BEST-P

モジュール仕様書

2023年4月

＜更新履歴＞

- 2021 年 6 月版　初版（2021.6.1）
- 2023 年 4 月版　Pump 台数制御 2019
  蓄電池 G2018
  媒体テストデータセット 2022 追加（2023.4.1）

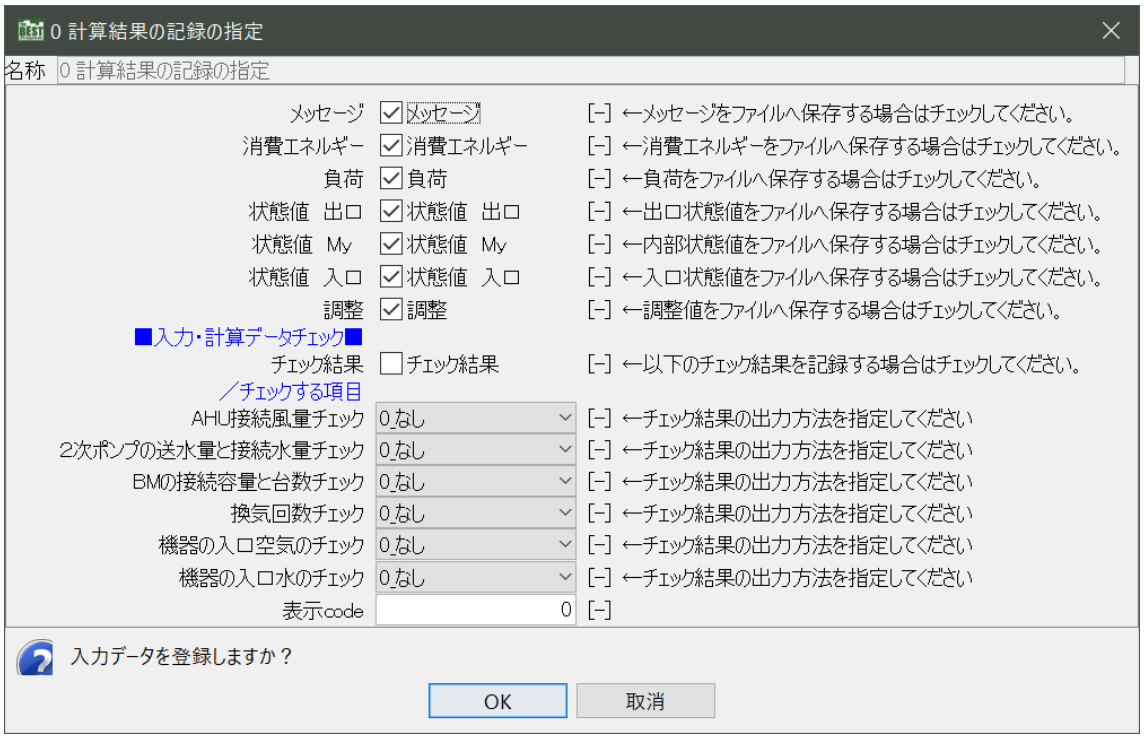
目次

「計算結果の記録の指定 2015」（場所：設備 2015／）

モジュール名	計算結果の記録の指定 2015
クラス	CheckPrintModule

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、設備モジュールからの計算結果の記録項目の範囲を指定する。

設備モジュールでは、記録するデータを次のように分類している。

メッセージ・・・・モジュールの計算過程での進捗状況の説明を記録する

消費エネルギー・・電力・ガスなどの消費エネルギーを記録する

負荷・・・・・・・モジュールが処理した熱量や要求熱量などを記録する

状態値　出口・・・モジュールの出口側の接続媒体の状態値を記録する

状態値　My・・・・モジュールの内部の状態値を記録する

状態値　入口・・・モジュールの入口側の接続媒体の状態値を記録する

調整・・・・・・・仮説調整（自動容量調整）の調整結果を記録する

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュ

ールからの出力項目」を下記に示す。

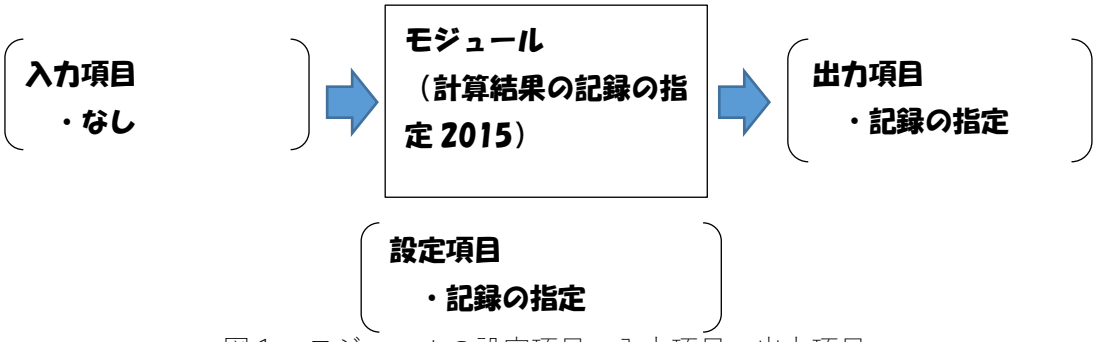


図１　モジュールの設定項目・入力項目・出力項目

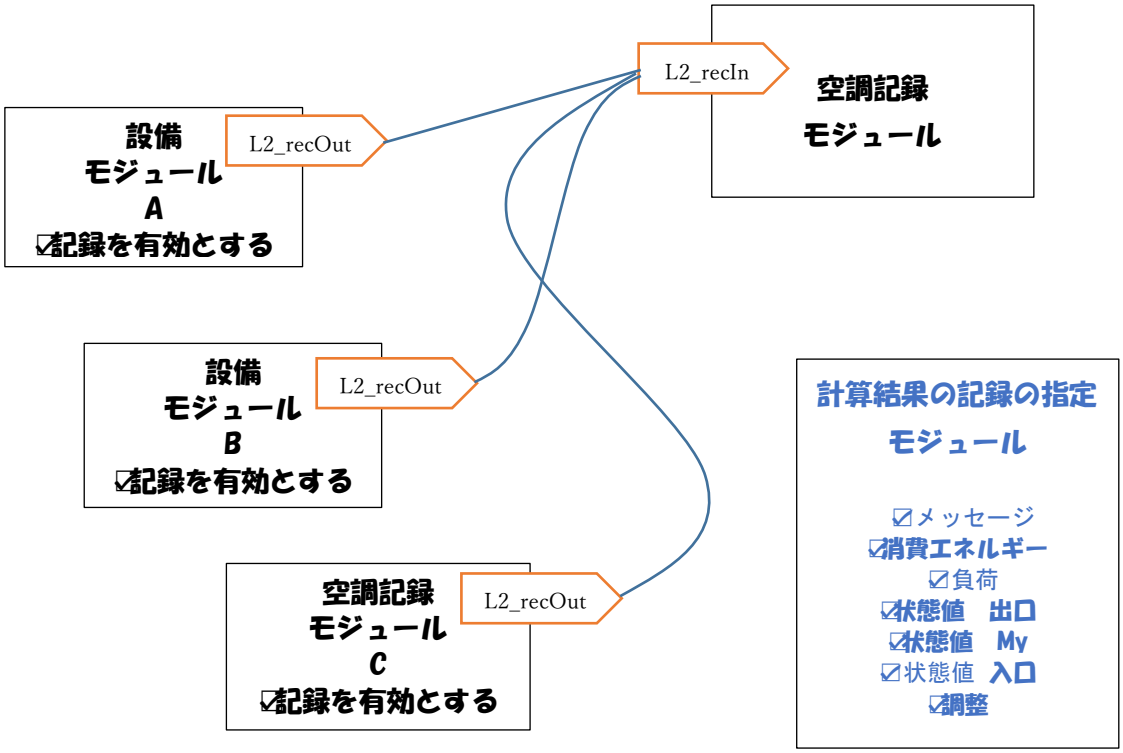
＊設備モジュールの記録がファイルへ出力される条件は次の通りである

①空調記録モジュールが登録されている

②設備モジュールの L2_recOut と空調記録モジュールの L2_recIn が接続されている

③設備モジュールの記録を有効とするが☑されている

④計算結果の記録の指定モジュールの出力範囲が☑されている



（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	―	―		
1	メッセージ	boolean	isPrintMessage	FALSE	[-]	―	―		メッセージをファイルへ保存する場合はチェックしてください。
2	消費エネルギー	boolean	isPrintEnergy	FALSE	[-]	―	―		消費エネルギーをファイルへ保存する場合はチェックしてください。
3	負荷	boolean	isPrintLoad	FALSE	[-]	―	―		負荷をファイルへ保存する場合はチェックしてください。
4	状態値　出口	boolean	isPrintStateOut	FALSE	[-]	―	―		出口状態値をファイルへ保存する場合はチェックしてください。
5	状態値　My	boolean	isPrintStateMy	FALSE	[-]	―	―		内部状態値をファイルへ保存する場合はチェックしてください。
6	状態値　入口	boolean	isPrintStateIn	FALSE	[-]	―	―		入口状態値をファイルへ保存する場合はチェックしてください。
7	調整	boolean	isPrintAdjust	FALSE	[-]	―	―		調整値をファイルへ保存する場合はチェックしてください。
	■入力・計算データチェック■	Label				―	―		
8	チェック結果	boolean	isPrintLOG	FALSE	[-]	―	―		以下のチェック結果を記録する場合はチェックしてください。

9	／チェックする項目	Label				―	―		
10	AHU 接続風量チェック	String	isLogD_AHUSum	0_なし、1_画面、2_ファイル、3_画面とファイル	[-]	―	―		チェック結果の出力方法を指定してください
11	２次ポンプの送水量と接続水量チェック	String	isLogD_2PWSum	0_なし、1_画面、2_ファイル、3_画面とファイル	[-]	―	―		チェック結果の出力方法を指定してください
12	BM の接続容量と台数チェック	String	isLogD_BMSum	0_なし、1_画面、2_ファイル、3_画面とファイル	[-]	―	―		チェック結果の出力方法を指定してください
13	換気回数チェック	String	isLogD_AirNV	0_なし、1_画面、2_ファイル、3_画面とファイル	[-]	―	―		チェック結果の出力方法を指定してください
14	機器の入口空気のチェック	String	isLogD_AirInC	0_なし、1_画面、2_ファイル、3_画面とファイル	[-]	―	―		チェック結果の出力方法を指定してください
15	機器の入口水のチェック	String	isLogD_WatInC	0_なし、1_画面、2_ファイル、3_画面とファイル	[-]	―	―		チェック結果の出力方法を指定してください
16	表示 code	String	numLanguage	0	[-]	―	―		0=日本語

（５）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ

（7）計算フロー・計算内容

全ての設備モジュールに対して、ここで指定した記録項目の範囲の制御を行う。

ここで指定がない場合は、機器などの設備モジュール側で「記録を有効とする」となっていても記録は行わない。

（8）データ範囲と範囲外の取扱い

チェック結果の記録については、誘導基準対応ツールの機能である。

以下の項目についてのチェックをする、しないの指定、チェックした記録を画面出力するかファイル保存するかの指定を行う。

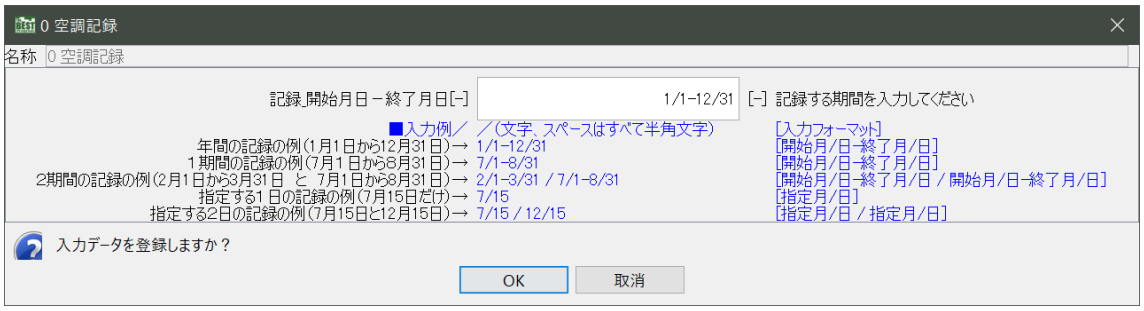
・空調機系統の定格 SA ファン風量と系統接続風量のチェック
・二次ポンプの定格流量と接続機器の定格流量合計値のチェック
・BM 室外機の定格能力と接続室内機の定格能力の合計値のチェック
・換気回数のチェック、機器への入口空気・水の状態値の異常チェック

「0 空調記録 2015」（場所：設備 2015／）

モジュール名	空調記録
クラス	AirSystemControl

（1）入力画面

・スペック

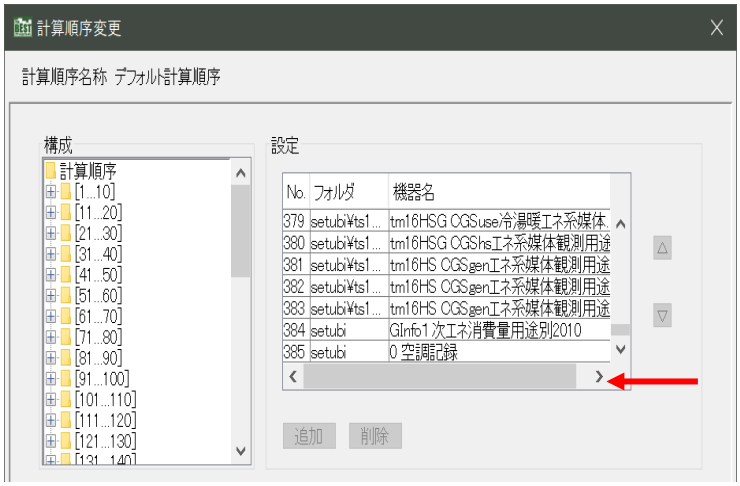


（2）モジュールの概要

本モジュールは、設備モジュールからの記録を制御するもので、各計算ステップごとに設備モジュールからの記録を出力するようメインフレーム側へ指示を出す。

設備モジュールの記録をファイル（best_result.csv）に出力するには、この空調記録モジュールを登録しておき、空調記録モジュールの記録ノード L2_recIn と設備モジュールの L2_recOut を接続しておく必要がある。

このモジュールの計算順序は最後とする。（下図のデフォルトの計算順序参照）



＊設備モジュールの記録する項目の範囲については、別途「計算結果の記録の指定」モジュールで設定する必要がある。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

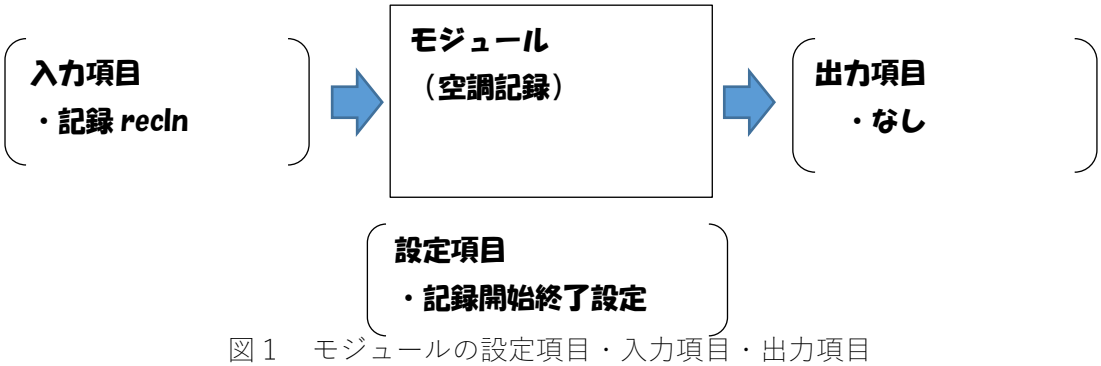


図１　モジュールの設定項目・入力項目・出力項目

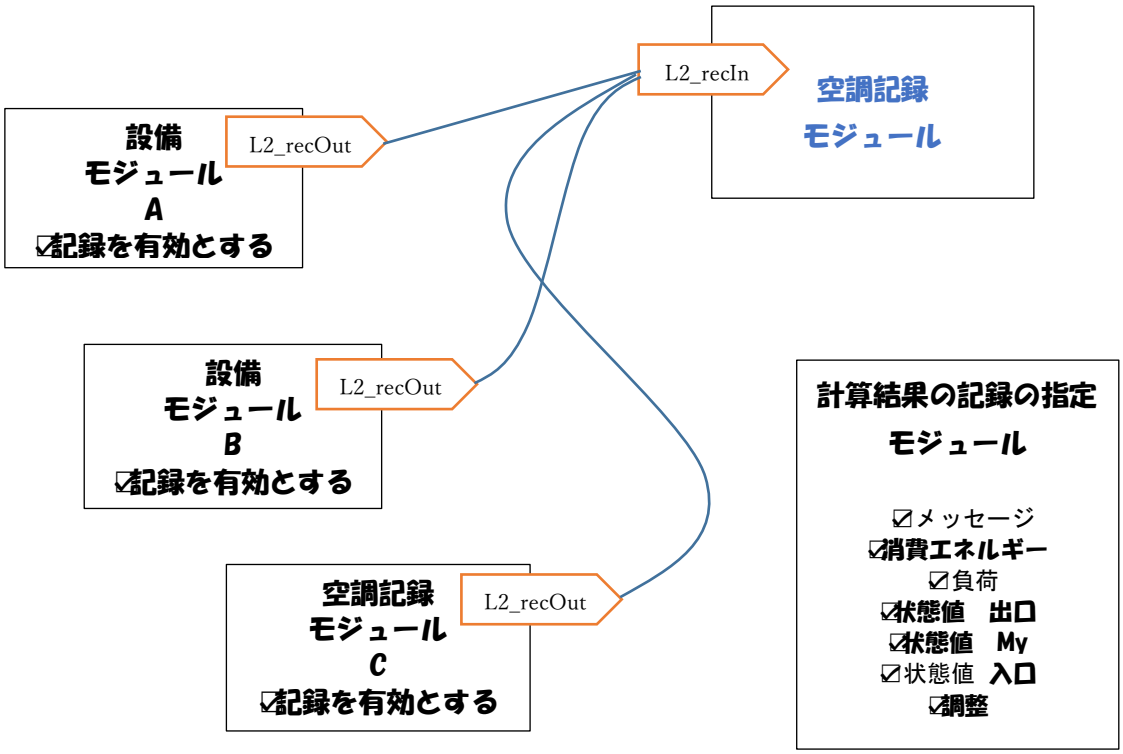
＊設備モジュールの記録がファイルへ出力される条件は次の通りである

①空調記録モジュールが登録されている

②設備モジュールの L2_recOut と空調記録モジュールの L2_recIn が接続されている

③設備モジュールの記録を有効とするが☑されている

④計算結果の記録の指定モジュールの出力範囲が☑されている



（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	Name		[-]	ー	ー		
1	記録_開始月日-終了月日	String	OPE_START_END	1/1-12/31	[-]	ー	ー		記録する期間を入力してください

〇記録_開始月日－終了月日の入力方法

年間の記録の例（1 月 1 日～12 月 31 日）→　1/1-12/31　[開始月/日-終了月/日]

１期間の記録の例（7 月 1 日～8 月 31 日）→　7/1-8/31　[開始月/日-終了月/日]

２期間の記録の例（2 月 1 日～3 月 31 日 と 7 月 1 日～8 月 31 日）→ 2/1-3/31 / 7/1-8/31 [開始月/日-終了月/日 / 開始月/日-終了月/日]

指定する 1 日の記録の例（7 月 15 日だけ）→　7/15　[指定月/日]

指定する 2 日の記録の例（7 月 15 日と 12 月 15 日）→　7/15 / 12/15　[指定月/日 / 指定月/日]

＊文字、スペースはすべて半角文字

＊[月/日-時：分]のフォーマットで入力する

＊複数の記録開始終了設定をする場合は「 / 」（半角スペース＋/+半角スペース）で区切る

＊記録開始終了設定は計算開始から終了までの範囲で計算進行の順番に設定する

（5）シーケンス接続（図 1 における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recIn	REC	-	記録	メモリ	入口	空調記録モジュールからの出力を受け取る。

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ

（7）計算フロー・計算内容

次の動作を行う。

各計算ステップごとに
現在の計算月日（時刻）が記録開始終了設定に該当するかチェックする
→該当する時、設備モジュールからの記録を出力するようメインフレーム側へ指示する
→該当しない場合、設備側からの記録をクリアするようメインフレーム側へ指示する

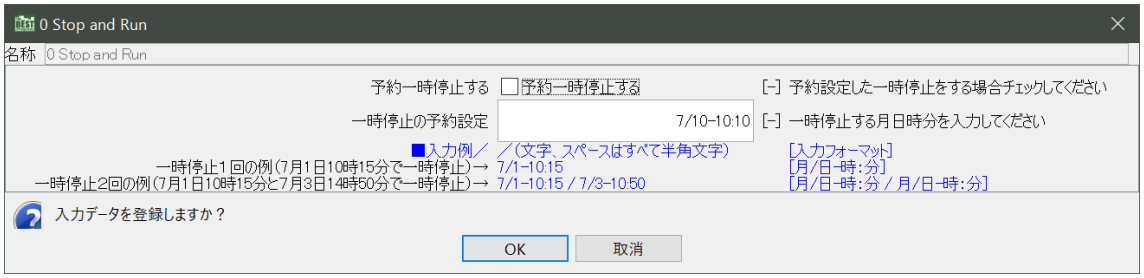
＊記録開始終了設定には年の指定がないため、助走計算の期間についても適用される。
このため、設定を 1/1-12/31 と年間とした場合、助走計算期間も記録を出力する。

「0 Stop and Run2015」（場所：設備 2015／）

モジュール名	Stop and Run
クラス	StopAndRunModule

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、設備の連成計算中に計算日時を表示して計算の進捗状況を確認することができる。また、計算の一時停止・再開や停止、計算速度などを操作することができる。

計算中のグラフ表示の値をチェックする場合に一時停止や計算速度を操作し、チェックした値が異常値であった場合は計算を停止する、などの使い方を想定している。

計算の一時停止については、あらかじめ一時停止させる日時を設定しておく「予約一時停止」機能がある。予約一時停止は複数の日時を設定可能であり、再開ボタンを押すことで次の予約一時停止日時まで計算が進み一時停止する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

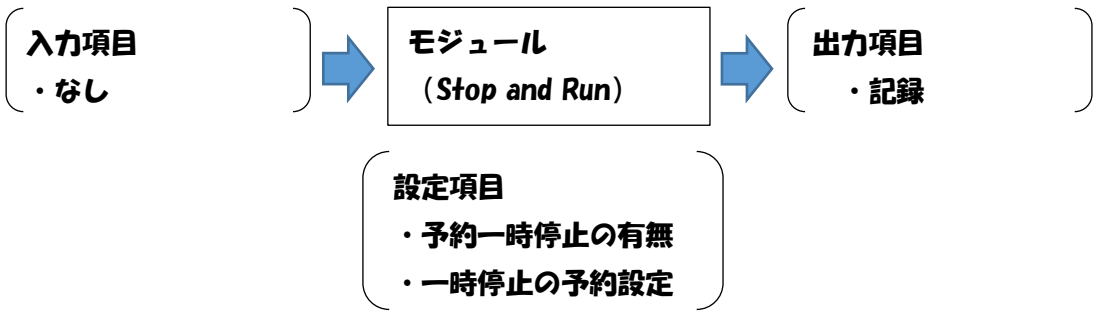


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	予約一時停止する	Boolean	isOPE_preSTOP	false	[-]	ー	ー		予約設定した一時停止をする場合にチェックしてください。
2	一時停止の予約設定	String	ope_preSTOP	7/10-10:10	[-]	ー	ー		一時停止する月日時分を入力してください。

〇一時停止の予約設定の入力方法

一時停止 1 回の例（7 月 1 日 10 時 15 分で一時停止）→　7/1-10:15　[月/日-時：分]

一時停止 2 回の例（7 月 1 日 10 時 15 分と 7 月 3 日 14 時 50 分で一時停止）→　7/1-10:15 / 7/3-10:50　[月/日-時：分 / 月/日-時：分]

＊文字、スペースはすべて半角文字

＊[月/日-時：分]のフォーマットで入力する

＊複数の予約設定をする場合は「 / 」（半角スペース＋/+半角スペース）で区切る

＊予約設定は計算開始から終了までの範囲で計算進行の順番に設定する

（5）シーケンス接続（図 1 における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。

（６）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	－	メッセージ

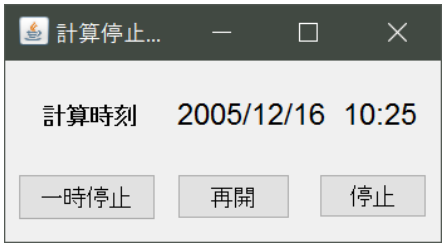
（７）計算フロー・計算内容

計算中に表示される画面の操作により次の動作を行う。

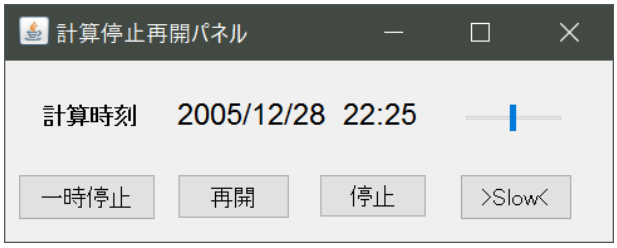
「一時停止」ボタンが押された時：計算を一時停止する（計算中の情報は保持している）
「再開」ボタンが押された時：計算を再開する
「停止」ボタンが押された時：計算を強制終了する
「＞SLOW＜」ボタンが押された時：
スライドバーの操作が有効となり、その値で計算速度を制御する。

（８）計算中に表示される画面の操作方法

計算を実行すると次の画面が表示される。



右枠を右へ拡大すると次のように「＞SLOW＜」ボタンとスライドバーが操作できる。



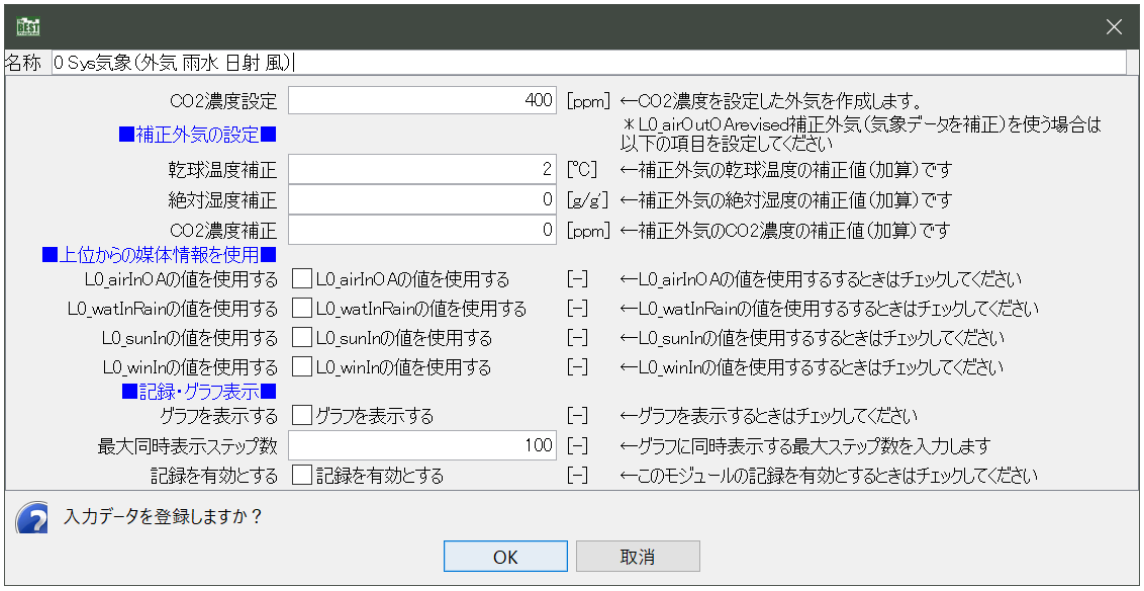
スライドバーの操作は、左方向が遅く、右方向が早くなり、左端で一時停止状態、右端で通常計算の速さとなる。

「0 Sys 気象（外気 雨水 日射 風)」(場所：設備 2015／)

モジュール名	システム用気象（外気 雨水 日射 風）
クラス	SystemWeatherModule20080909

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、気象データを設備モジュールへ伝達するモジュールである。

また、外気の CO2 濃度の設定や、乾球温度・絶対湿度・CO2 濃度を補正したものを出力するノードを備えている。

本モジュールは空調関連の各テンプレートに組み込まれており、テンプレート単位で条件を変更することができる。

例えば、気象データの乾球温度を＋5℃補正したものを BM などの室外機置き場の温度として室外機に与え、吸い込み温度が高温となった時の能力低下の計算を簡単に行うことができる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

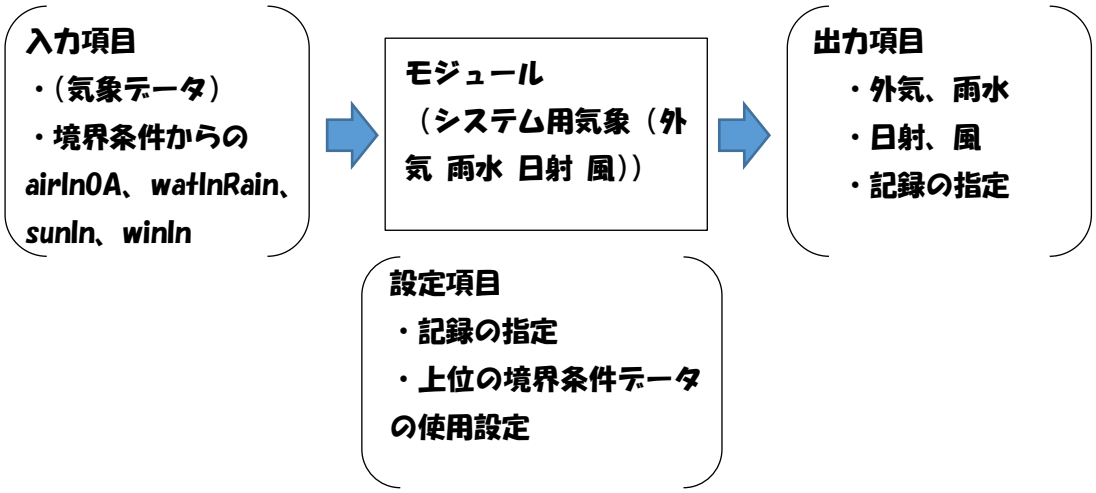


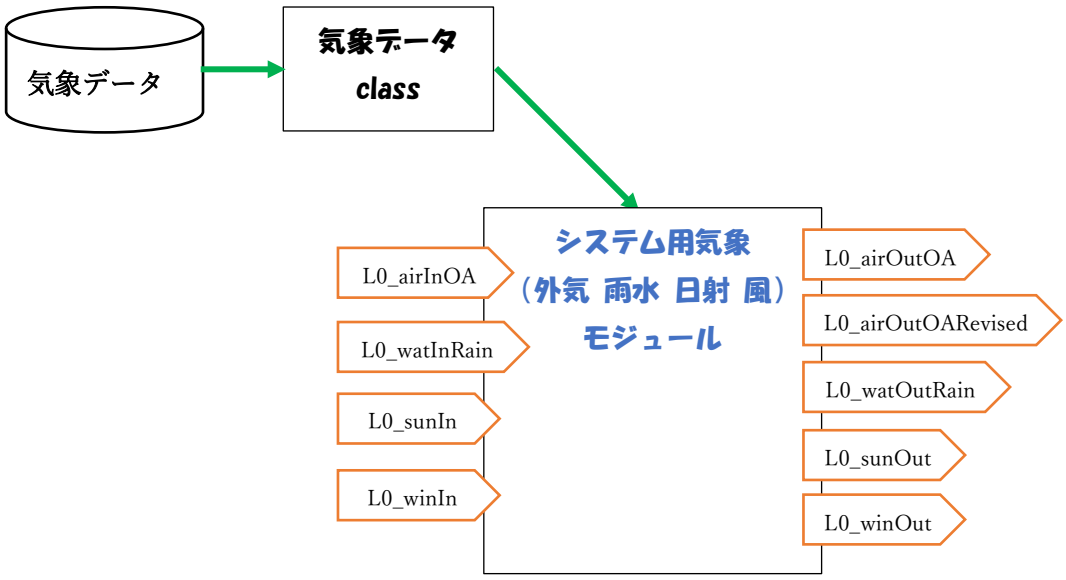
図１　モジュールの設定項目・入力項目・出力項目

気象データの流れについて次のような使い方がある

①　「共通／気象」画面で設定した拡張アメダスなどの気象データを取り込む場合

システム用気象モジュールを登録するだけで、自動で「共通／気象」画面で設定した気象データを取り込む。

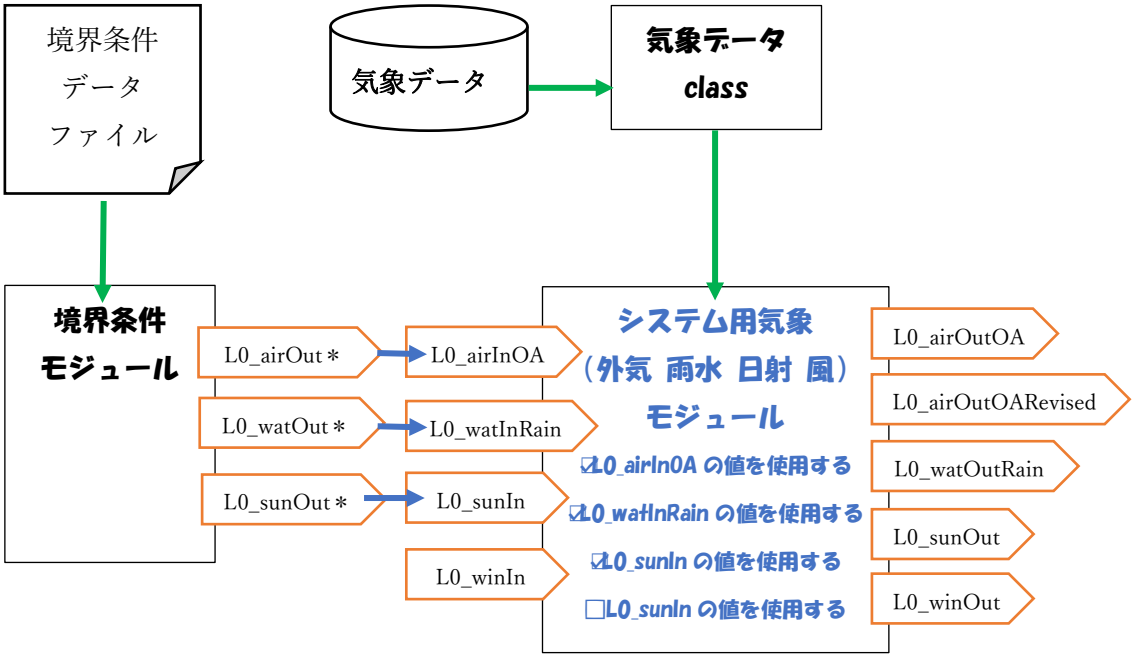
（下図のモジュール間の→はデータの流れを示すもので、ノード接続は不要である）



② 境界条件のデータファイルから気象データを取り込む場合

境界条件データファイルにセットした外気、雨水、日射を境界条件モジュールで取り込み、境界条件モジュールの各出力ノードとシステム用気象モジュールの対応する入力ノードを接続し、上位からの媒体情報を使用欄の使用ノードをチェックする。

下図の例では、境界条件モジュールから外気、雨水、日射を受けとりそれらの値を下位へ出力している。L0_winOut からは気象データ（class）の風情報が出力される。



（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	―	―		
1	CO2 濃度設定	Double	co2ppm	400	[ppm]	―	―		←CO2 濃度を設定した外気を作成します。
2	■補正外気の設定■					―	―		
3	乾球温度補正	Double	addTemp	0	[℃]	―	―		←補正外気の乾球温度の補正値（加算）です
4	絶対湿度補正	Double	addHumi	0	[g/g']	―	―		←補正外気の絶対湿度の補正値（加算）です
5	CO2 濃度補正	Double	addCO2ppm	0	[ppm]	―	―		←補正外気の CO2 濃度の補正値（加算）です
6	■上位からの媒体情報を使用■								
7	LO_airInOA の値を使用する	Boolean	isUSEairInOA	FALSE	[-]	―	―		←LO_airInOA の値を使用するするときはチェックしてください
8	LO_watInRain の値を使用する	Boolean	isUSEwatInRain	FALSE	[-]	―	―		←LO_watInRain の値を使用するするときはチェックしてください
9	LO_sunIn の値を使用する	Boolean	isUSEsunIn	FALSE		―	―		←LO_sunIn の値を使用するするときはチェックしてください
10	LO_winIn の値を使用す	Boolean	isUSEwinIn	FALSE	[-]	―	―		←LO_winIn の値を使用するする

	る								ときはチェックしてください
11	■記録・グラフ表示■								
12	グラフを表示する	Boolean	isGVisible	FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
13	最大同時表示ステップ数	Int	maxItemCount	100	[-]	ー	ー		←グラフに同時表示する最大ステップ数を入力します
14	記録を有効とする	Boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	外気	LO_airOutOA	airOutOA	-	状態	空気	出口	補正していない外気を出力する
3	外気（補正）	LO_airOutOArevised	airOutOArevised	-	状態	空気	出口	乾球温度、絶対湿度、CO2 濃度を補正した外気を出力する
4	雨	LO_watOutRain	watOutRain	-	状態	水	出口	雨を出力する
5	日射	LO_sunOut	sunOut	-	状態	日射	出口	日射を出力する
6	風	LO_winOut	winOut	-	状態	風	出口	風を出力する
7	上位からの外気	LO_airInOA	airInOA	-	状態	空気	入口	境界条件モジュールなどと接続する
8	上位からの雨	LO_watInRain	watInRain	-	状態	水	入口	境界条件モジュールなどと接続する
9	上位からの日射	LO_sunIn	sunIn	-	状態	日射	入口	境界条件モジュールなどと接続する
10	上位からの風	LO_winIn	winIn	-	状態	風	入口	境界条件モジュールなどと接続する

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
	外気乾球温度	温度	℃	出口
	外気絶対湿度	湿度	g/g	出口
	法線面直達日射量	日射量	W/m2	出口
	水平面天空日射量	日射量	W/m2	出口
	水平面全天日射量	日射量	W/m2	出口
	法線面直射照度	照度	Lx	出口
	水平面天空照度	照度	Lx	出口
	水平面全天照度	照度	Lx	出口
	水平面夜間放射量	放射量	W/m2	出口
	風向	風向	—	出口
	風速	風速	m/s	出口
	太陽方位角	方位	deg	出口
	太陽高度	高度	deg	出口
	降水量	雨量	mm	出口
	外気 CO2 濃度	CO2 濃度	ppm	出口

（7）計算フロー・計算内容

補正ノードへ補正した値をセットし出力する。

上位からの媒体情報を使用する場合は、出力側ノードへ上位からの値をセットして出力する。

（8）データ範囲と範囲外の取扱い

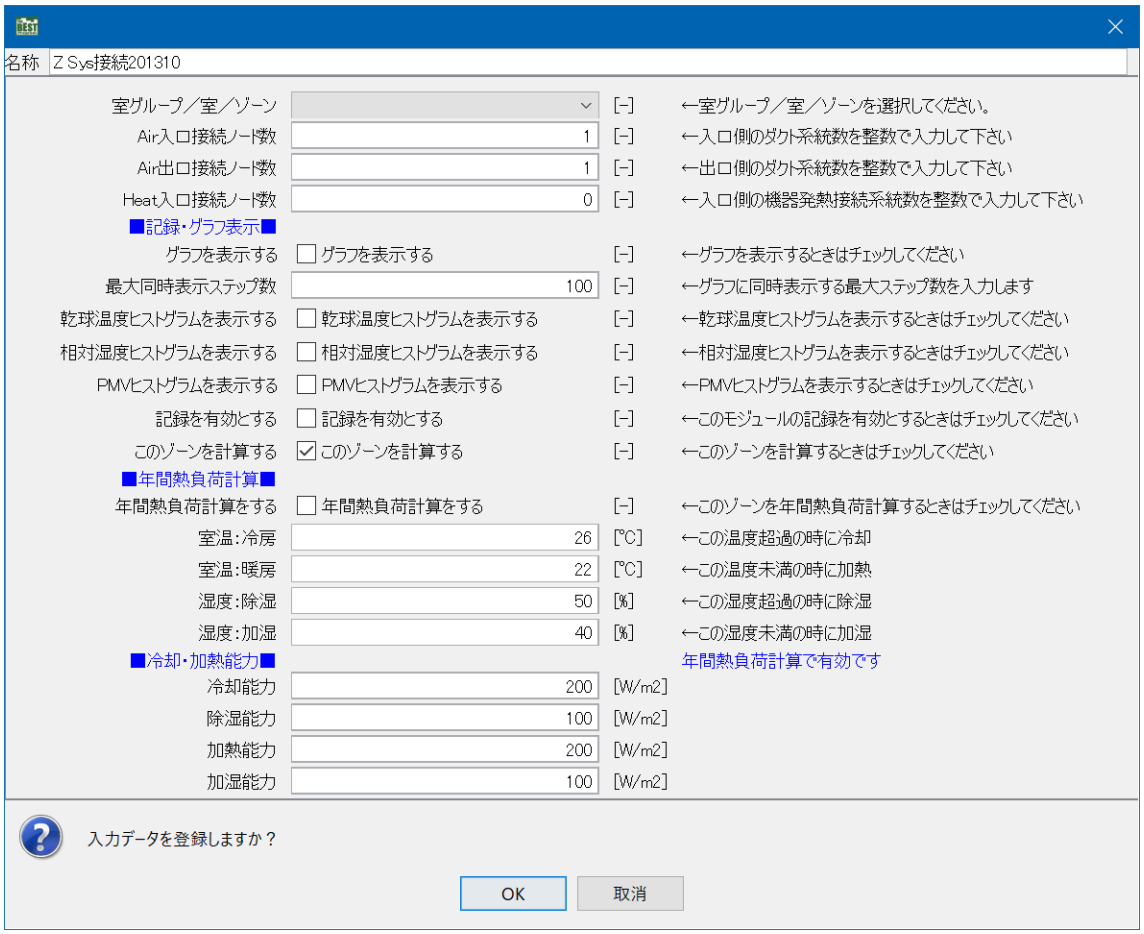
特になし。

「Z Sys 接続 201310」（場所：設備 2015／Z ゾーン 2015／）

モジュール名	Z Sys 接続 201310
クラス	ZoneforSystemModule201310*

（1）入力画面

・スペック



（2）モジュールの概要

ゾーンへ空調・還気機器の吸排気を接続し建物側のゾーンの計算に反映させます。

ゾーンの環境情報、照明消費電力、コンセント消費電力、在室人数を取得します。

行政版専用機能として年間負荷計算（PAL 計算）を備えています。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

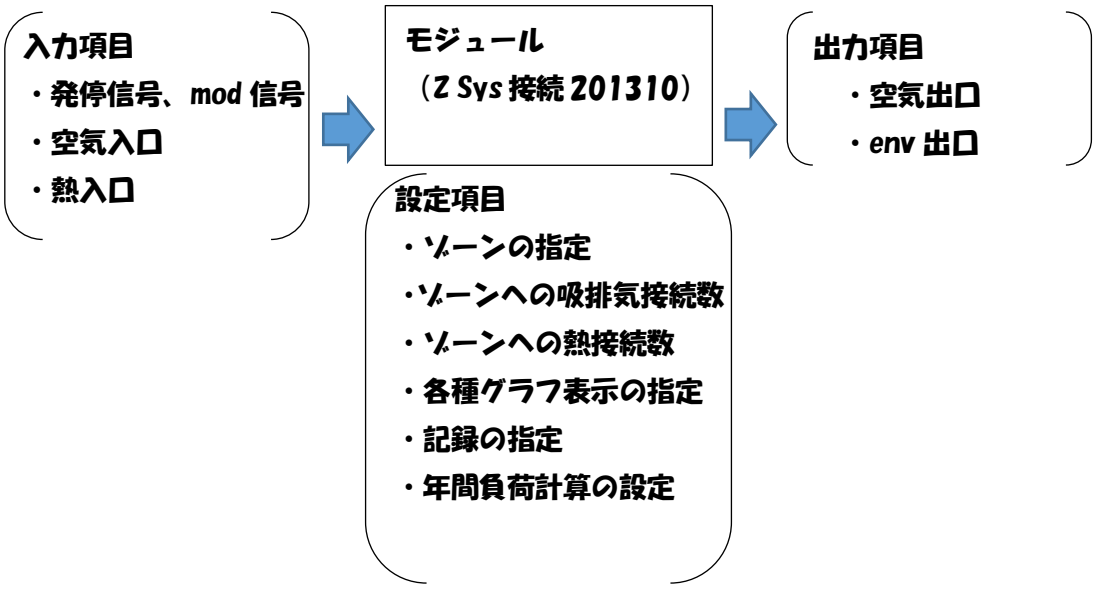


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	室グループ／室／ゾーン	String	coreSpace sysSpace	#RoomGroup	[-]	ー	ー		←室グループ／室／ゾーンを選択してください。
2	Air 入口接続ノード数	int	numberOfAirInlet	0	[-]	ー	0		←入口側のダクト系統数を整数で入力して下さい
3	Air 出口接続ノード数	int	numberOfAirOutlet	0	[-]	ー	0		←出口側のダクト系統数を整数で入力して下さい
4	Heat 入口接続ノード数	int	numberOfHeatInlet	0	[-]	ー	0		←入口側の機器発熱接続系統数を整数で入力して下さい
5	■記録・グラフ表示■					ー	ー		
6	グラフを表示する	boolean	isGVisible	FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
7	最大同時表示ステップ数	int	maxItemCount	100	[-]	ー	10		←グラフに同時表示する最大ステップ数を入力します
8	乾球温度ヒストグラムを表示する	boolean	isGDBVisible	FALSE	[-]	ー	ー		←乾球温度ヒストグラムを表示するときはチェックしてください
9	相対湿度ヒストグラムを表示する	boolean	isGRHVisible	FALSE	[-]	ー	ー		←相対湿度ヒストグラムを表示するときはチェックしてください
10	PMV ヒストグラムを表示する	boolean	isGPMVVisible	FALSE	[-]	ー	ー		←PMV ヒストグラムを表示するときはチェックしてください
11	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください
12	このゾーンを計算する	boolean	isCalc	TRUE	[-]	ー	ー		←このゾーンを計算するときはチェックしてください
13	■年間熱負荷計算■					ー	ー		＊行政版の機能（年間熱負荷計算：PAL 計算）
14	年間熱負荷計算をする	boolean	isCalcAnnualLoad	FALSE	[-]	ー	ー		←このゾーンを年間熱負荷計算するときはチェックしてください
15	室温：冷房	double	setTempC	26	[℃]	ー	ー		←この温度超過の時に冷却

16	室温：暖房	double	setTempH	22	[℃]	―	―		←この温度未満の時に加熱
17	湿度：除湿	double	setHumiRC	50	[%]	―	―		←この湿度超過の時に除湿
18	湿度：加湿	double	setHumiRH	40	[%]	―	―		←この湿度未満の時に加湿
19	■冷却・加熱能力■					―	―		
20	冷却能力	double	desCoolS	200	[W/m2]	―	―		
21	除湿能力	double	desCoolL	100	[W/m2]	―	―		
22	加熱能力	double	desHeatS	200	[W/m2]	―	―		
23	加湿能力	double	desHeatL	100	[W/m2]	―	―		

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	空気入口	L0_airIn	airIn	-	状態	空気	入口	
5	空気出口	L0_airOut	airOut	-	状態	空気	出口	
6	空気入口[]	L0_airInPair[]	airInPair[]	-	状態	空気	入口	
7	空気出口[]	L0_airOutPair[]	airOutPair[]	-	状態	空気	出口	
8	Env 出口	L0_envOut	envOut	-	状態	室環境	出口	
9	照明電力入口	L0_eleInLighting	eleInLighting	-	状態	電気	入口	
10	コンセント電力入口	L0_eleInConcent	eleInConcent	-	状態	電気	入口	
11	発熱入口[]	L0_heaIn[]	heaIn[]	-	状態	熱	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	ZSys_Message#-#-	メッセージ	—	メッセージ
2	ZSys_室出口乾球温度#℃#温度	ZSys_室出口乾球温度	℃	出口
3	ZSys_室出口質量流量#g/s#質量流量	ZSys_室出口質量流量	g/s	出口
4	ZSys_室出口絶対湿度#g/g'#絶対湿度	ZSys_室出口絶対湿度	g/g	出口
5	ZSys_室出口 CO2 濃度#ppm#濃度	ZSys_室出口 CO2 濃度	ppm	出口
6	ZSys_室入口乾球温度#℃#温度	ZSys_室入口乾球温度	℃	入口
7	ZSys_室入口質量流量#g/s#質量流量	ZSys_室入口質量流量	g/s	入口
8	ZSys_室入口絶対湿度#g/g'#絶対湿度	ZSys_室入口絶対湿度	g/g	入口
9	ZSys_室入口 CO2 濃度#ppm#濃度	ZSys_室入口 CO2 濃度	ppm	入口
10	ZAirSys_airInPir["+i+"]乾球温度#℃#温度	ZAirSys_airInPir["+i+"]乾球温度	℃	入口
11	ZAirSys_airInPir["+i+"]質量流量#g/s#質量流量	ZAirSys_airInPir["+i+"]質量流量	g/s	入口
12	ZAirSys_airInPir["+i+"]絶対湿度#g/g'#絶対湿度	ZAirSys_airInPir["+i+"]絶対湿度	g/g	入口
13	ZAirSys_airInPir["+i+"]CO2 濃度#ppm#濃度	ZAirSys_airInPir["+i+"]CO2 濃度	ppm	入口
14	ZAirSys_airOutPir["+i+"]質量流量#g/s#質量流量	ZAirSys_airOutPir["+i+"]質量流量	g/s	出口
15	ZSys_室 PMV#-#PMV	ZSys_室 PMV	—	My
16	ZSys_室作用温度#℃#作用温度	ZSys_室作用温度	℃	My
17	ZSys_室照明消費電力#W#電力 照明・コンセント 照明	ZSys_室照明消費電力	W	エネルギー消費
18	ZSys_室コンセント消費電力#W#電力 照明･コンセント コンセント	ZSys_室コンセント消費電力	W	エネルギー消費
19	ZSys_室在室人数#人#-	ZSys_室在室人数	人	My
20	ZSys_累積冷房顕熱負荷#MJ#-	ZSys_累積冷房顕熱負荷	MJ	年間熱負荷計算をするが有効の時
21	ZSys_累積冷房潜熱負荷#MJ#-	ZSys_累積冷房潜熱負荷	MJ	年間熱負荷計算をするが有効の時
22	ZSys_累積冷房全熱負荷#MJ#-	ZSys_累積冷房全熱負荷	MJ	年間熱負荷計算をするが有効の時
23	ZSys_累積冷房顕熱負荷#MJ/m2#-	ZSys_累積冷房顕熱負荷	MJ/m2	年間熱負荷計算をするが有効の時
24	ZSys_累積冷房潜熱負荷#MJ/m2#-	ZSys_累積冷房潜熱負荷	MJ/m2	年間熱負荷計算をするが有効の時
25	ZSys_累積冷房全熱負荷#MJ/m2#-	ZSys_累積冷房全熱負荷	MJ/m2	年間熱負荷計算をするが有効の時
26	ZSys_累積暖房顕熱負荷#MJ#-	ZSys_累積暖房顕熱負荷	MJ	年間熱負荷計算をするが有効の時
27	ZSys_累積暖房潜熱負荷#MJ#-	ZSys_累積暖房潜熱負荷	MJ	年間熱負荷計算をするが有効の時
28	ZSys_累積暖房全熱負荷#MJ#-	ZSys_累積暖房全熱負荷	MJ	年間熱負荷計算をするが有効の時
29	ZSys_累積暖房顕熱負荷#MJ/m2#-	ZSys_累積暖房顕熱負荷	MJ/m2	年間熱負荷計算をするが有効の時
30	ZSys_累積暖房潜熱負荷#MJ/m2#-	ZSys_累積暖房潜熱負荷	MJ/m2	年間熱負荷計算をするが有効の時

31	ZSys_累積暖房全熱負荷#MJ/m2#-	ZSys_累積暖房全熱負荷	MJ/m2	年間熱負荷計算をするが有効の時
32				

（７）計算フロー・計算内容

・接続された L0_airIn、L0_airInPair[＊]（空調機器からのゾーンへの送風)、L0_heatIn[＊]（機器の発熱）を指定したゾーンへ伝達し建築側のゾーンの計算に反映させます。

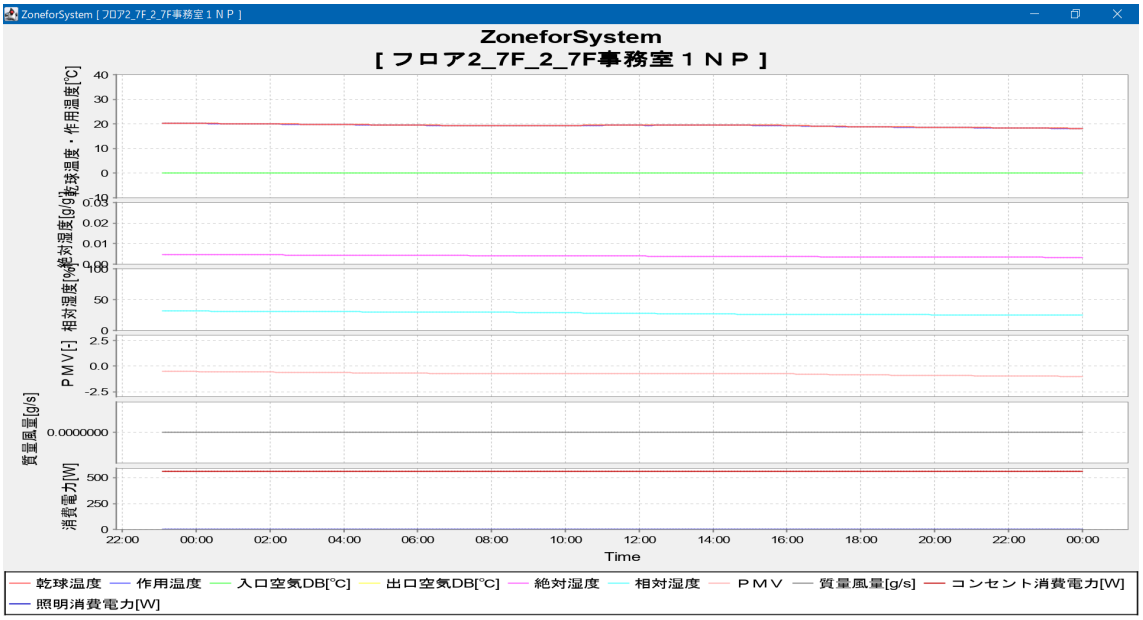
・指定したゾーンの空気を L0_airOut、L0_airOutPair[＊]（空調機器への還気）にセットします。

・指定したゾーンの環境情報を L0_envOut にセットします。

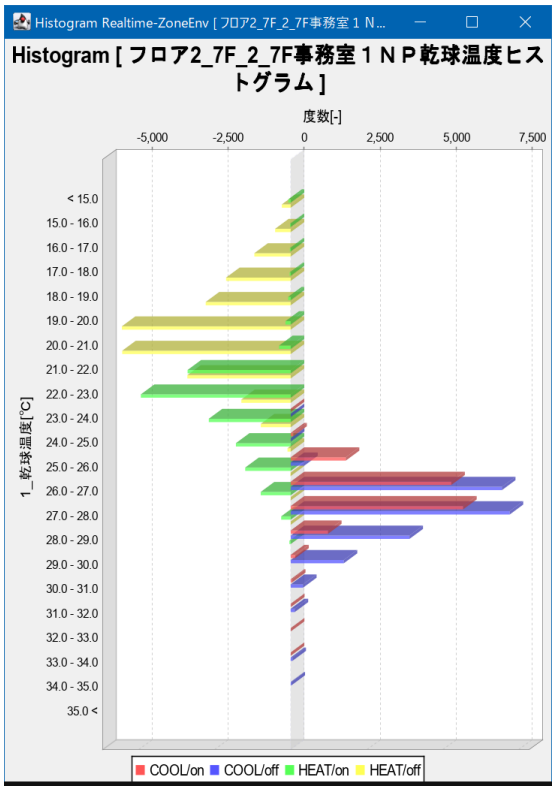
・指定したゾーンの照明消費電力とコンセント消費電力を L0_eleInLighting、L_eleInConcent にセットします。

・グラフを表示する

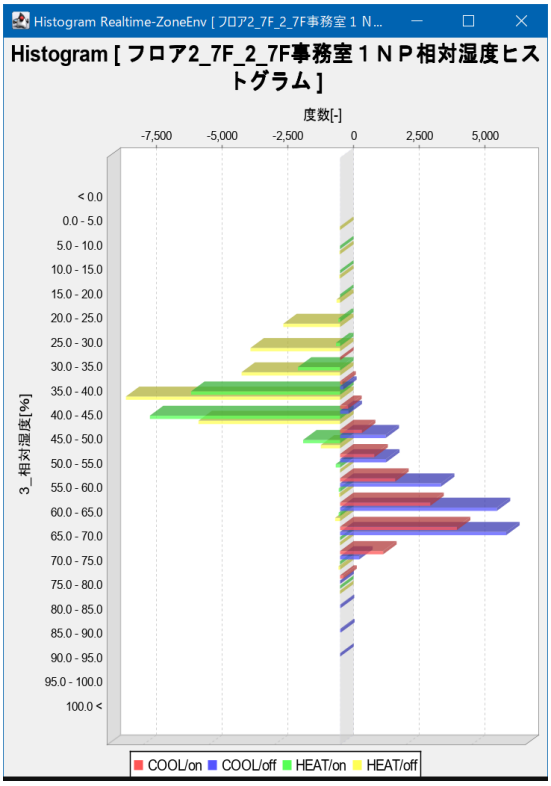
　次の乾球・作用温度、絶対湿度、相対湿度、PMV、風量、消費電力のトレンドグラフを計算中に表示します。最大同時表示ステップ数で横軸の幅を設定できます。



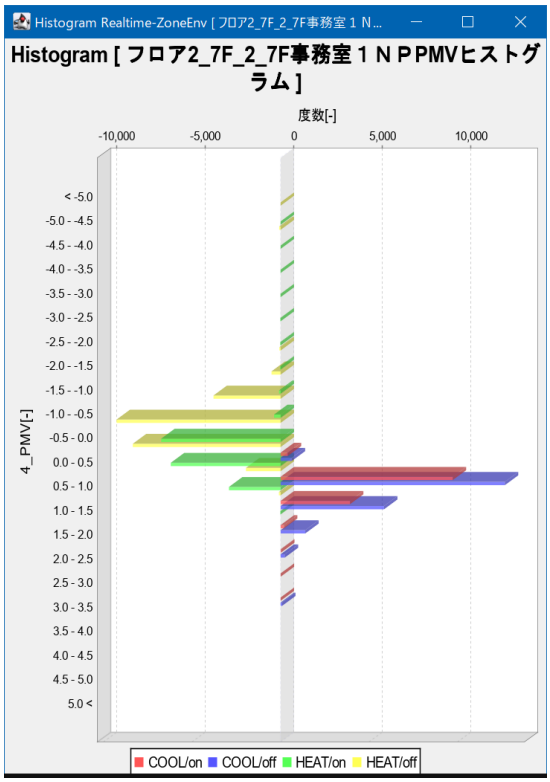
・乾球温度ヒストグラムを表示する



・相対湿度ヒストグラムを表示する



・PMV ヒストグラムを表示する



（8）データ範囲と範囲外の取扱い

年間負荷計算は、行政版の専用機能です。

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestHeat;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.building.load.SystemHeatGain;
import jp.or.ibec.best.domain.building.spaces.ISpace;
import jp.or.ibec.best.domain.building.spaces.LoadElementManager;
import jp.or.ibec.best.domain.building.spaces.ZoneElectric;
import jp.or.ibec.best.domain.building.spaces.ZoneEnv;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.log.BestLogger;
import jp.or.ibec.best.message.BestEngineMessageHandler;
import jp.or.ibec.best.message.BestEngineMessageParam;
import jp.or.ibec.best.message.constants.SystemMessageConstants;

/**
 * 空調吹出空気情報を建物側に渡し、リターン空気を出力するモジュール
 * env、照明、コンセント（機器）消費電力を出力、heatを接続可能
 * 行政版向けに累積負荷出力
 * @author  kohri HIROSHI NINOMIYA 20080529  /20080909 /20081212
 *
 * 記録の有効無効判断を追加
 * 20081013/空調時間帯が異なる場合  if(airInPair[i].getFlowRate() > 0.){   //2008.10.13kohri修正
 * 20090101/新規ZoneResult に対応
 * 20090505/ゾーン名称取扱い変更
 * 20101111/温度、湿度、PMVのヒストグラム表示追加
 *
 * 20130801 SystemHeatGain  に切替
 *
 * 20131030 ZoneAirCalc201310  を使用//ISpaceへのairの負荷の受渡し方法を変更
 */
public class ZoneforSystemModule201310 extends AbstractBestModule implements
      IBestMetaModule, jp.or.ibec.best.dk.test.IProbeForUT {

   private final String moduleName = "(ZoneforSystemModule201310) ";

   //接続ノード
   private final String S_NODE_airOut = "L0_airOut";
   private final String S_NODE_airIn  = "L0_airIn";

   //入口  接続名称は  L0_airInPair[0],L0_airInPair[1]・・・・とする
   //出口  接続名称は  L0_airOutPair[0],L0_airOutPair[1]・・・・とする
   private String[] S_NODE_airInPair;
   private String[] S_NODE_airOutPair;
   //入口  接続名称は  L0_heaIn[0],L0_heaIn[1]・・・・とする
   private String[] S_NODE_heaIn;

   private final String S_NODE_envOut = "L0_envOut";
   private final String S_NODE_eleInLighting = "L0_eleInLighting";
   private final String S_NODE_eleInConcent  = "L0_eleInConcent";

   private final String C_NODE_swcIn = "L1_swcIn";
   private final String C_NODE_modIn = "L1_modIn";

   private final String R_NODE = "L2_recOut";
```

```
//外部定義項目
//仕様
private final String SPEC_name = "名称";
private final String SPEC_grzName = "室グループ／室／ゾーン";
//
private final String SPEC_NumberOfAirInlet  = "[]Air入口接続ノード数";
private final String SPEC_NumberOfAirOutlet = "[]Air出口接続ノード数";
private final String SPEC_NumberOfHeatInlet = "[]Heat入口接続ノード数";
private final String SPEC_isGVisible   = "グラフを表示する";
private final String SPEC_maxItemCount = "最大同時表示ステップ数";
private final String SPEC_isGDBVisible = "乾球温度ヒストグラムを表示する";
private final String SPEC_isGRHVisible = "相対湿度ヒストグラムを表示する";
private final String SPEC_isGPMVVisible= "PMVヒストグラムを表示する";
private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする
private final String SPEC_isRecordPAL = "PALの記録を有効とする";//PALの記録を有効とする

private final String SPEC_isCalc = "このゾーンを計算する";//行政ツール用

private final String SPEC_isCalcAnnualLoad = "年間熱負荷計算をする";//行政ツール用
private final String SPEC_setTempC  = "室温度冷却[℃]";
private final String SPEC_setTempH  = "室温度加熱[℃]";
private final String SPEC_setHumiRC = "室湿度除湿[-]";
private final String SPEC_setHumiRH = "室湿度加湿[-]";
private final String SPEC_desCoolS = "冷却能力[W/m2]";
private final String SPEC_desCoolL = "除湿能力[W/m2]";
private final String SPEC_desHeatS = "加熱能力[W/m2]";
private final String SPEC_desHeatL = "加湿能力[W/m2]";

//記録
private final String RECORD_message    = "ZSys_Message#-#-";
private final String RECORD_T_airOut   = "ZSys_室出口乾球温度#℃#温度";
private final String RECORD_M_airOut   = "ZSys_室出口質量流量#g/s#質量流量";
private final String RECORD_X_airOut   = "ZSys_室出口絶対湿度#g/g'#絶対湿度";
private final String RECORD_CO2_airOut = "ZSys_室出口CO2濃度#ppm#濃度";
private final String RECORD_T_airIn    = "ZSys_室入口乾球温度#℃#温度";
private final String RECORD_M_airIn    = "ZSys_室入口質量流量#g/s#質量流量";
private final String RECORD_X_airIn    = "ZSys_室入口絶対湿度#g/g'#絶対湿度";
private final String RECORD_CO2_airIn  = "ZSys_室入口CO2濃度#ppm#濃度";

private String[] RECORD_T_airInPair = null;
private String[] RECORD_M_airInPair = null;
private String[] RECORD_X_airInPair = null;
private String[] RECORD_CO2_airInPair = null;
private String[] RECORD_M_airOutPair = null;//出口のT、X、COはairOutに同じ

private final String RECORD_P_envOut = "ZSys_室PMV#-#PMV";
private final String RECORD_O_envOut = "ZSys_室作用温度#℃#作用温度";

private final String RECORD_PPE_eleInLighting = "ZSys_室照明消費電力#W#電力 照明・コンセント 照明";
private final String RECORD_PPE_eleInConcent  = "ZSys_室コンセント消費電力#W#電力 照明・コンセント コンセント";
private final String RECORD_nHumanStep = "ZSys_室在室人数#人#-";

private final String RECORD_cSHL    = "ZSys_累積冷房顕熱負荷#MJ#-";
private final String RECORD_cLHL    = "ZSys_累積冷房潜熱負荷#MJ#-";
private final String RECORD_cTHL    = "ZSys_累積冷房全熱負荷#MJ#-";
private final String RECORD_cSHL1m2 = "ZSys_累積冷房顕熱負荷#MJ/m2#-";
private final String RECORD_cLHL1m2 = "ZSys_累積冷房潜熱負荷#MJ/m2#-";
private final String RECORD_cTHL1m2 = "ZSys_累積冷房全熱負荷#MJ/m2#-";
private final String RECORD_hSHL    = "ZSys_累積暖房顕熱負荷#MJ#-";
private final String RECORD_hLHL    = "ZSys_累積暖房潜熱負荷#MJ#-";
private final String RECORD_hTHL    = "ZSys_累積暖房全熱負荷#MJ#-";
private final String RECORD_hSHL1m2 = "ZSys_累積暖房顕熱負荷#MJ/m2#-";
private final String RECORD_hLHL1m2 = "ZSys_累積暖房潜熱負荷#MJ/m2#-";
private final String RECORD_hTHL1m2 = "ZSys_累積暖房全熱負荷#MJ/m2#-";

private double cSHL1m2 = 0.;//累積冷房顕熱負荷[J/m2]
private double cLHL1m2 = 0.;//累積冷房潜熱負荷[J/m2]
private double cTHL1m2 = 0.;//累積冷房全熱負荷[J/m2]
private double hSHL1m2 = 0.;//累積暖房顕熱負荷[J/m2]
private double hLHL1m2 = 0.;//累積暖房潜熱負荷[J/m2]
private double hTHL1m2 = 0.;//累積暖房全熱負荷[J/m2]
```

```
private double floorArea = 0;//床面積[m2]

private double timeInterval;

//接続熱媒など
private BestAir[] airInPair  = null;
private BestAir[] airOutPair  = null;
private BestAir   airOut = null;
private BestAir   airIn = null;
private BestAir   airInMix  = null;

private BestHeat[]  heaIn = null;

private ZoneEnv   envOut = null;
private BestElectricity eleInLighting = null;
private BestElectricity eleInConcent = null;

//制御信号など
private int swcIn;               //運転モード
private int modIn;

//仕様など
private String name;          //名称
private int numberOfAirInlet;    //Air入口  接続数
private int numberOfAirOutlet;      //Air出口  接続数
private int numberOfHeatInlet;      //Heat入口接続数
//
private boolean isGVisible = false;//このグラフを表示する=true
private int  maxItemCount = 100;//最大同時表示ステップ数
private boolean isGDBVisible = false;//DBこのグラフを表示する=true
private boolean isGRHVisible = false;//RHこのグラフを表示する=true
private boolean isGPMVVisible = false;//PMVこのグラフを表示する=true
private boolean isRecord = false;//記録を有効とする=true
private boolean isRecordPAL = false;//PALの記録を有効とする=true
private boolean isCalc = true;//このゾーンを計算する  行政ツール用

private boolean isCalcAnnualLoad = false;//"年間熱負荷計算をする";//行政ツール用
private double setTempC;// = "室温冷房[℃]";
private double setTempH;// = "室温暖房[℃]";
private double setHumiRC;// = "室湿度冷房[-]";
private double setHumiRH;// = "室湿度暖房[-]";
private double desCoolS;// = "冷却能力[W/m2]";
private double desCoolL;// = "除湿能力[W/m2]";
private double desHeatS;// = "加熱能力[W/m2]";
private double desHeatL;// = "加湿能力[W/m2]";
//private double airOut_maxFlowRate;      //出口  最大質量流量[g/s]

private ArrayList<ZoneElectric> zonesElectric;  //建物側電力情報インスタンス

private SystemHeatGain systemHeatGain; //建物側インスタンス

private StringBuffer message = new StringBuffer();


private ISpace coreSpace   ;//建物側インスタンス

private int[] dateTime; //0:年、1:月、2:日、3:年間通算日、4:曜日、5:時、6:分、7:秒

//グラフ表示など
private GraphJFrameZoneforSystem20090101 gZone = null;
//
private GraphRealtimeZoneEnvHistogramJFrame20080909 gZoneEnvhDB = null;
private GraphRealtimeZoneEnvHistogramJFrame20080909 gZoneEnvhRH = null;
private GraphRealtimeZoneEnvHistogramJFrame20080909 gZoneEnvhPMV = null;

private boolean isCalcAirInPair = false;//20130318
private boolean isCalcAirOutPair = false;//20130318
private boolean isCalcHeaIn = false;//20130318

private double nHumanStep;//各ステップの在室人数
private double metHuman;//met
```

```
// private ISpace coreSpace   ;//建物側インスタンス
   private SystemISpace sysSpace = null;

   private ZoneAirSysCalc201310 zoneAirSysCalc = ZoneAirSysCalc201310.getInstance();


   private double zoneVolume = 0;
   private double zoneCO2ppm = 500.;


   @Override
   public void setProfile(BestSpecs spec) {
      if(spec == null) {
         return;
      }

      Map<String, String> map = spec.getSpec();
      if(map == null) {
         return;
      }

      //isCalcを取得
      this.isCalc = spec.getSpecValue( this.SPEC_isCalc, true );

      if( !this.isCalc ){
         return;
      }

      //名称を取得
      this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

      //numberOfAirInletを取得
      this.numberOfAirInlet = spec.getSpecValue( this.SPEC_NumberOfAirInlet, 0 );

      //出口ｎ系統のBestAirの初期化
      if( this.numberOfAirInlet > 0 ){
         this.isCalcAirInPair = true;//20130318
         this.airInPair = new BestAir[this.numberOfAirInlet];
         this.S_NODE_airInPair = new String[this.numberOfAirInlet];
         this.RECORD_T_airInPair   = new String[this.numberOfAirInlet];
         this.RECORD_M_airInPair   = new String[this.numberOfAirInlet];
         this.RECORD_X_airInPair   = new String[this.numberOfAirInlet];
         this.RECORD_CO2_airInPair = new String[this.numberOfAirInlet];

         for( int i=0; i<this.airInPair.length; i++ ){
            this.S_NODE_airInPair[i] = new String( "LO_airInPair[" + i + "]" );
            this.RECORD_T_airInPair[i] = new String( "ZAirSys_airInPir["+i+"]乾球温度#℃#温度" );
            this.RECORD_M_airInPair[i] = new String( "ZAirSys_airInPir["+i+"]質量流量#g/s#質量流量" );
            this.RECORD_X_airInPair[i] = new String( "ZAirSys_airInPir["+i+"]絶対湿度#g/g'#絶対湿度" );
            this.RECORD_CO2_airInPair[i] = new String( "ZAirSys_airInPir["+i+"]CO2濃度#ppm#濃度" );
         }
      }


      //numberOfAirOutletを取得
      this.numberOfAirOutlet = spec.getSpecValue( this.SPEC_NumberOfAirOutlet, 0 );

      //出口ｎ系統のBestAirの初期化
      if( this.numberOfAirOutlet > 0 ){
         this.isCalcAirOutPair = true;//20130318
         this.airOutPair = new BestAir[this.numberOfAirOutlet];
         this.S_NODE_airOutPair = new String[this.numberOfAirOutlet];
         this.RECORD_M_airOutPair = new String[this.numberOfAirOutlet];

         for( int i=0; i<this.airOutPair.length; i++ ){
            this.S_NODE_airOutPair[i] = new String( "LO_airOutPair[" + i + "]" );
            this.RECORD_M_airOutPair[i] = new String( "ZAirSys_airOutPir["+i+"]質量流量#g/s#質量流量" );
         }
      }
```

```
        //numberOfHeatInletを取得
        this.numberOfHeatInlet = spec.getSpecValue( this.SPEC_NumberOfHeatInlet, 0 );

        //出口ｎ系統のBestAirの初期化
        if( this.numberOfHeatInlet > 0 ){
           this.isCalcHeaIn = true;
           this.heaIn = new BestHeat[this.numberOfHeatInlet];
           this.S_NODE_heaIn = new String[this.numberOfHeatInlet];

           for( int i=0; i<this.heaIn.length; i++ ){
              this.S_NODE_heaIn[i] = new String( "LO_heaIn[" + i + "]" );
           }
        }



        //室グループ、ゾーン名の取得
        String grzName = null;
        grzName = spec.getSpecValue( this.SPEC_grzName, "" );

        this.coreSpace = ZoneGRZName.getCoreSpace( grzName, this.name );

        this.sysSpace = this.zoneAirSysCalc.getSystemISpace( grzName, this.coreSpace );


        //SystemHeatGainインスタンスを建物側に登録
        this.systemHeatGain = new SystemHeatGain();
        this.systemHeatGain.setProfile(
              ZoneGRZName.getMultiSpaceName( grzName, this.name ),
              ZoneGRZName.getZoneName( grzName, this.name ) );

        //室床面積[㎡]
        this.floorArea = this.systemHeatGain.getZone().getValue(ISpace.FLOORAREA);


        this.zoneVolume = this.sysSpace.getISpace().getValue( ISpace.FLOORAREA ) *
    this.sysSpace.getISpace().getValue( ISpace.CEILINGHEIGHT );
        this.sysSpace.setVolume( this.zoneVolume );
        this.sysSpace.setCO2ppm( this.zoneCO2ppm );

        name = ZoneGRZName.getMultiSpaceName( grzName, this.name ) + "_" + ZoneGRZName.getZoneName( grzName, this.name );

        //MultiSpaceNameとZoneNameを指定したとき
        //該当するゾーンの電力情報インスタンスを取得
      // ISpace zone = SpaceManager.getISpace( multiSpaceName, zoneName );
        ISpace zone = this.systemHeatGain.getZone();
        if( zone != null ){
           ZoneElectric zoneElectric=zone.getZoneElectric();
           if( zoneElectric != null ){
              zonesElectric = new ArrayList<ZoneElectric>();
              zonesElectric.add(zoneElectric);
           }
        }

        //Heat
      // systemHeat=(AbstractSystemHeat)coreSpace.getLoadElement(LoadElementManager.SYSTEMHEAT);
    //    heatCRL = new HeatCRL();
    //    heatCRLPAL = new HeatCRLPAL();

        //isGVisibleを取得
        this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

        // 最大同時表示ステップ数
        this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

        //isGDBVisibleを取得
        this.isGDBVisible = spec.getSpecValue( this.SPEC_isGDBVisible, false );

        //isGRHVisibleを取得
        this.isGRHVisible = spec.getSpecValue( this.SPEC_isGRHVisible, false );

        //isGPMVVisibleを取得
```

```
        this.isGPMVVisible = spec.getSpecValue( this.SPEC_isGPMVVisible, false );


        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

        //isRecordPALを取得
        this.isRecordPAL = spec.getSpecValue( this.SPEC_isRecordPAL, false );

        //SPEC_isCalcAnnualLoad = "年間熱負荷計算をする";//行政ツール用
        this.isCalcAnnualLoad = spec.getSpecValue( this.SPEC_isCalcAnnualLoad, false );

        //SPEC_setTempC = "室温冷房[℃]";
        this.setTempC = spec.getSpecValue( this.SPEC_setTempC, 26. );

        //SPEC_setTempH = "室温暖房[℃]";
        this.setTempH = spec.getSpecValue( this.SPEC_setTempH, 22. );

        this.sp   = this.setTempC;
        this.h_sp = this.setTempH;

        //SPEC_setHumiRC = "室湿度冷房[-]";
        this.setHumiRC = spec.getSpecValue( this.SPEC_setHumiRC, 0.5 );

        //SPEC_setHumiRH = "室湿度暖房[-]";
        this.setHumiRH = spec.getSpecValue( this.SPEC_setHumiRH, 0.4 );

        this.l_sp   = this.setHumiRC * 100.;
        this.h_l_sp = this.setHumiRH * 100.;

        //SPEC_desCoolS = "冷却能力[W/m2]";
        this.desCoolS = spec.getSpecValue( this.SPEC_desCoolS, 200. );

        //SPEC_desCoolL = "除湿能力[W/m2]";
        this.desCoolL = spec.getSpecValue( this.SPEC_desCoolL, 100. );

        //SPEC_desHeatS = "加熱能力[W/m2]";
        this.desHeatS = spec.getSpecValue( this.SPEC_desHeatS, 200. );

        //SPEC_desHeatL = "加湿能力[W/m2]";
        this.desHeatL = spec.getSpecValue( this.SPEC_desHeatL, 100. );

    }

    public void initialize(IBestStateMessage stateNodes,
            IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
            IBestDomainSchedule schedule) {
        //状態ノードを取得
        super.sm = stateNodes;

        //制御ノードを取得
        super.cm = commandNodes;

        //記録ノードを取得
        super.rm = recordNodes;

        if( !this.isCalc ){
            return;
        }

        //接続ノード　出口
        //airOut
        this.airOut = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOut );
/*      if( super.sm.getState( super.getConnectionNode( this.S_NODE_airOut)) != null ){
            this.airOut
            = (BestAir) super.sm.getState( super.getConnectionNode( this.S_NODE_airOut));
        }else{
            this.airOut = new BestAir();
            super.sm.setState( super.getConnectionNode( this.S_NODE_airOut ), this.airOut );
        }
*/
        //airOutPair[ ]
```

```
        //if( this.numberOfAirOutlet > 0 ){
        if( this.isCalcAirOutPair ){//20130318
           for( int i=0; i< this.airOutPair.length; i++ ){
              this.airOutPair[i] = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutPair[i] );
           }
        }

        //envOut
        if( super.sm.getState( super.getConnectionNode( this.S_NODE_envOut)) != null ){
           this.envOut
           = (ZoneEnv) super.sm.getState( super.getConnectionNode( this.S_NODE_envOut));
        }else{
           this.envOut = new ZoneEnv();
           super.sm.setState( super.getConnectionNode( this.S_NODE_envOut ), this.envOut );
        }


        //入口
        //airIn
        this.airIn = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airIn );

        //airInPair[ ]
        //if( this.numberOfAirInlet > 0 ){
        if( this.isCalcAirInPair ){//20130318
           for( int i=0; i< this.airInPair.length; i++ ){
              this.airInPair[i] = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInPair[i] );
           }
        }

        //heaIn[ ]
        //if( this.numberOfHeatInlet > 0 ){
        if( this.isCalcHeaIn ){//20130318
           for( int i=0; i< this.heaIn.length; i++ ){
              this.heaIn[i] = BestHeat.bindnode( super.transferMapState, super.sm, this.S_NODE_heaIn[i] );
           }
        }

        //eleInLighting
        this.eleInLighting = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleInLighting );

        //eleInConcent
        this.eleInConcent = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleInConcent );


        //グラフ表示の準備
        if( this.isGVisible ){
           gZone = new GraphJFrameZoneforSystem20090101( this.name, this.maxItemCount, 0 * 1.5, 0 * 1.5 );
        }

        if( this.isGDBVisible ){
           this.gZoneEnvhDB = new GraphRealtimeZoneEnvHistogramJFrame20080909( this.name+"乾球温度ヒストグラム", 15, 35,
20, "5_棒３Ｄグラフ", "1_乾球温度[℃]", true, true);
        }

        if( this.isGRHVisible ){
           this.gZoneEnvhRH = new GraphRealtimeZoneEnvHistogramJFrame20080909( this.name+"相対湿度ヒストグラム", 0, 100,
20, "5_棒３Ｄグラフ", "3_相対湿度[%]", true, true);
        }

        if( this.isGPMVVisible ){
           this.gZoneEnvhPMV = new GraphRealtimeZoneEnvHistogramJFrame20080909( this.name+"PMVヒストグラム", -5, 5, 20, "5_
棒３Ｄグラフ", "4_PMV[-]", true, true);
        }


        if( null != this.sysSpace.getISpace().getLoadElementManager().getLoadElement(LoadElementManager.HUMAN)){
           this.nHumanStep =

     ((jp.or.ibec.best.domain.building.load.Human)this.sysSpace.getISpace().getLoadElementManager().getLoadElement(LoadElem
entManager.HUMAN)).getNHuman(false);
        }else{
           this.nHumanStep = 0;
```

```
        }

        //
        this.airInMix = new BestAir();

        if( this.isCalc ){
           this.zoneAirSysCalc.setZoneIDandAir( this.sysSpace, this.airInMix );
        }
     }


     public void outputs() {

        if( !this.isCalc ){
           return;
        }

        //接続　入口状態値の取得
        this.airIn
        =(BestAir)super.sm.getState( super.getConnectionNode( this.S_NODE_airIn ));

        //if( this.numberOfAirInlet > 0 ){
        if( this.isCalcAirInPair ){//20130318
           for( int i=0; i< this.airInPair.length; i++ ){
              this.airInPair[i]
              = (BestAir) super.sm.getState( super.getConnectionNode( this.S_NODE_airInPair[i]));
           }
        }

        //airIn＊＊　の混合空気
        BestAir.calcMixedAirHX( this.airIn, this.airInPair, this.airInMix );

        //if( this.numberOfHeatInlet > 0 ){
        if( this.isCalcHeaIn ){//20130318
           for( int i=0; i< this.heaIn.length; i++ ){
              this.heaIn[i]
              = (BestHeat) super.sm.getState( super.getConnectionNode( this.S_NODE_heaIn[i]));
           }
        }

        this.swcIn = super.cm.getCommand(super.getConnectionNode( this.C_NODE_swcIn ));
        this.modIn = super.cm.getCommand(super.getConnectionNode( this.C_NODE_modIn ));

        if( Airswc.isOFF( this.swcIn ) ){ //空気が流れていない時
           //
           //System.out.println( "zonesystem swcIn = 0");
           if( this.isRecord ){
              this.message.append( "(C)swc停止" );
           }
        }else{   //空気が流れている時

        }


        //室状態値の取得
        if( this.coreSpace == null ){
           BestEngineMessageHandler.addMessage( new BestEngineMessageParam( SystemMessageConstants.CAL_ERROR2 ,
                 new String[]{ "( "+this.name+" ) [ 室／室グループ／室ゾーン ]", "ゾーンが負荷に未接続"} ));
           BestLogger.info( "(W)"+AirSystemControl.getTimeStr()
                 + "("+moduleName+"_"+"setProfile"+")["+name+"]"
                 + "coreSpace" +"(システムAir接続用： "+this.name+" ) [ 室グループ／室／ゾーン ]"+ "ゾーンが負荷に未接続
" );
        }
        double[] tx=sysSpace.getISpace().getValues(ISpace.SPACETX);
        if(tx==null) {
           System.out.println("SpaceModule >>Error<< spaceTX data are not defined !!");
           return;
        }
        //
        this.airOut.setTempDB(tx[0]);
        this.airOut.setHumi(tx[1]);
        this.airOut.setFlowRate( this.airIn.getFlowRate());
```

```
        this.airOut.setCO2ppm( this.sysSpace.getCO2ppm());
        //ノードに設定
        super.sm.setState(super.getConnectionNode( this.S_NODE_airOut ),
              this.airOut );

        if( this.isCalcAirOutPair ){//20130318
           for( int i=0; i<this.airOutPair.length; i++ ){
              this.airOutPair[i].setTempDB(tx[0]);
              this.airOutPair[i].setHumi(tx[1]);
              this.airOutPair[i].setFlowRate( this.airInPair[i].getFlowRate());
              this.airOutPair[i].setCO2ppm( this.sysSpace.getCO2ppm());
              //ノードに設定
              super.sm.setState(super.getConnectionNode( this.S_NODE_airOutPair[i] ),
                    this.airOutPair[i] );
           }
        }

        //ZoneEnv
        this.envOut.setEnv( this.sysSpace.getISpace().getZoneEnv().getTa(),
              this.sysSpace.getISpace().getZoneEnv().getXa(),
              this.sysSpace.getISpace().getZoneEnv().getPMV(),
              this.sysSpace.getISpace().getZoneEnv().getOT() );
        //ノードに設定
        super.sm.setState( super.getConnectionNode( this.S_NODE_envOut), this.envOut );

        //Human
        //System.out.println( "nHumanStep ZONE name="+this.name);
        if( null != this.sysSpace.getISpace().getLoadElementManager().getLoadElement(LoadElementManager.HUMAN)){
           this.nHumanStep =

   ((jp.or.ibec.best.domain.building.load.Human)this.sysSpace.getISpace().getLoadElementManager().getLoadElement(LoadElem
entManager.HUMAN)).getNHuman(true);
        }else{
           this.nHumanStep = 0;
        }
        //System.out.println( "zone="
+this.coreSpace.getName(ISpace.MULTISPACE)+"/"+this.coreSpace.getName(ISpace.ROOM)+"/"+this.coreSpace.getName(ISpace.SPAC
E)+"/"+ "nHuman=" + nHuman );
        this.sysSpace.setNHuman( this.nHumanStep );

        if( null != this.sysSpace.getISpace().getLoadElementManager().getLoadElement(LoadElementManager.HUMAN)){
           this.metHuman =

   ((jp.or.ibec.best.domain.building.load.Human)this.sysSpace.getISpace().getLoadElementManager().getLoadElement(LoadElem
entManager.HUMAN)).getMet();
        }else{
           this.metHuman = 0;
        }
        this.sysSpace.setMet( this.metHuman );

        //Heat ->> update()

        //Lighting Concent
        //システム側時間間隔が建物側の時間間隔と同じか大きいとき
        //ゾーン別電力量を集計して、出力側インスタンスに設定
        if( this.zonesElectric == null ){
           this.eleInLighting.setActivePower( 0 );
           this.eleInLighting.setReactivePower( 0 );
           this.eleInConcent.setActivePower( 0 );
           this.eleInConcent.setReactivePower( 0 );

        }else
        if( this.zonesElectric.get(0).getIntegTimeStep() > 0. ){
           double[] totalEpl=new double[2];
           for( ZoneElectric zoneElectric : this.zonesElectric ){
              double[] epl0=zoneElectric.getElectricPowerLoad();
              for(int i=0; i<2; i++) totalEpl[i]+=epl0[i];
           }
           this.eleInLighting.setActivePower( totalEpl[0] );
           this.eleInLighting.setReactivePower( totalEpl[0] );
           this.eleInConcent.setActivePower( totalEpl[1] );
           this.eleInConcent.setReactivePower( totalEpl[1] );
```

```
            //System.out.println( "Lighting, Concent = " + totalEpl[0] + "," + totalEpl[1] );
        }

        //20081217
        super.sm.setState(super.getConnectionNode( this.S_NODE_eleInLighting), this.eleInLighting);
        super.sm.setState(super.getConnectionNode( this.S_NODE_eleInConcent ), this.eleInConcent );


        //記録
        //本モジュールで設定した運転スケジュールを使用する
        dateTime = BestTimeManager.getDateWeatherTime();

        if( this.isRecordPAL  && super.rm != null ){
            //PAL記録する時は、PAL以外は記録しない
        }else if( this.isRecord && super.rm != null ){
            this.record();
        }

        //message.setLength(0);
    }

    /**
     * 記録
     */
    private void record(){
        if( CheckPrintModule.isPrintMessage ){
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                    this.RECORD_message, this.name, this.message.toString() );
        }

        if( CheckPrintModule.isPrintEnergy ){
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                    this.RECORD_PPE_eleInLighting, this.name,
                    AirFormat.df_2( this.eleInLighting.getActivePower() )); //"ZoneforSystem室照明消費電力#W#電力 照明・コンセント 照明";
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                    this.RECORD_PPE_eleInConcent, this.name,
                    AirFormat.df_2( this.eleInConcent.getActivePower() )); //"ZoneforSystem室コンセント消費電力#W#電力 照明・コンセント コンセント";
        }

        //if( CheckPrintModule.isPrintLoad ){
        //}

        if( CheckPrintModule.isPrintStateOut ){
            //出口
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                    this.RECORD_T_airOut, this.name,
                    AirFormat.df_2( this.airOut.getTempDB() ));
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                    this.RECORD_X_airOut, this.name,
                    AirFormat.df_4( this.airOut.getHumi() ));
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                    this.RECORD_M_airOut, this.name,
                    AirFormat.df_2( this.airOut.getFlowRate() ));
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                    this.RECORD_CO2_airOut, this.name,
                    AirFormat.df_0( this.airOut.getCO2ppm() ));
            //
            //if( this.numberOfAirOutlet > 0 ){
            if( this.isCalcAirOutPair ){//20130318
                for( int i=0; i<airOutPair.length; i++ ){
                    super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                            this.RECORD_M_airOutPair[i], this.name,
                            AirFormat.df_2( this.airOutPair[i].getFlowRate() ));
                }
            }
        }

        if( CheckPrintModule.isPrintStateMy ){
            //ZoneEnv
```

```
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
              this.RECORD_P_envOut,this.name,
              AirFormat.df_2( this.envOut.getPMV() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
              this.RECORD_O_envOut,this.name,
              AirFormat.df_2( this.envOut.getOT() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
              this.RECORD_nHumanStep, this.name,
              AirFormat.df_2( this.nHumanStep )); //"ZoneforSystem室在室人数";
    }

    if( CheckPrintModule.isPrintStateIn ){
        //入口
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
               this.RECORD_T_airIn, this.name,
               AirFormat.df_2( this.airIn.getTempDB() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
              this.RECORD_X_airIn, this.name,
              AirFormat.df_4( this.airIn.getHumi() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
              this.RECORD_M_airIn, this.name,
              AirFormat.df_2( this.airIn.getFlowRate() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
              this.RECORD_CO2_airIn, this.name,
              AirFormat.df_0( this.airIn.getCO2ppm() ));
        //
        //if( this.numberOfAirInlet > 0 ){
        if( this.isCalcAirInPair ){//20130318
            for( int i=0; i<airInPair.length; i++ )   {
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                      this.RECORD_T_airInPair[i], this.name,
                      AirFormat.df_2( this.airInPair[i].getTempDB()) );
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                      this.RECORD_X_airInPair[i], this.name,
                      AirFormat.df_5( this.airInPair[i].getHumi() ));
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                      this.RECORD_M_airInPair[i], this.name,
                      AirFormat.df_2( this.airInPair[i].getFlowRate() ));
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                      this.RECORD_CO2_airInPair[i], this.name,
                      AirFormat.df_0( this.airInPair[i].getCO2ppm() ));
            }
        }

        //Heat
        //if( this.numberOfHeatInlet > 0 ){
        if( this.isCalcHeaIn ){//20130318
            for( int i=0; i<heaIn.length; i++ )
            {
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                  this.S_NODE_heaIn[i] + "室機器対流熱負荷#W#負荷", this.name,
                  AirFormat.df_2( this.heaIn[i].getConvectiveHeatRate() ));
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                  this.S_NODE_heaIn[i] + "室機器放射熱負荷#W#負荷", this.name,
                  AirFormat.df_2( this.heaIn[i].getRadiativeHeatRate() ));
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                  this.S_NODE_heaIn[i] + "室機器潜熱負荷#W#負荷", this.name,
                  AirFormat.df_2( this.heaIn[i].getLatentHeatRate() ));
            }
        }
    }

}


public void update() {
    //if( super.sm == null || S_NODE_aIn == null) return;

    if( !this.isCalc ){
        return;
    }
```

```
      this.zoneAirSysCalc.countUp();
      //Heat
      //if( super.sm == null) return;
      //入力ノードから機器類の発熱熱取得を取得
   // heatCRL.clear();
      //if( this.numberOfHeatInlet > 0 ){
      if( this.isCalcHeaIn ){//20130318
         for( int i=0; i< this.heaIn.length; i++ ){
            this.heaIn[i]
            = (BestHeat) super.sm.getState( super.getConnectionNode( this.S_NODE_heaIn[i]));
            //2008.12.21kohri修正
            //heatCRL.addHeat( this.heaIn[i].getSensibleHeat(), this.heaIn[i].getLatentHeat(),
this.heaIn[i].getTempLevel());
            //heatCRL.addHeat( this.heaIn[i].getSensibleHeatHigh(), this.heaIn[i].getSensibleHeat(),
this.heaIn[i].getLatentHeat());
            this.systemHeatGain.integrateHeatGain( this.heaIn[i].getConvectiveHeatRate(),
                  this.heaIn[i].getRadiativeHeatRate(),
                  this.heaIn[i].getLatentHeatRate(),
                  BestTimeManager.getCurrentInterval());
         }
      }
   // systemHeat.addHeatGain(heatCRL, BestTimeManager.getCurrentInterval());

      //PAL
      if( this.isCalcAnnualLoad ){
         sensibleHeatRateHigh =0;
         sensibleHeatRate = 0;
         latentHeatRate = 0;
         //double dt=0;
         //double dtcb = 0;
         //double dthb = 0;
         //double c1 = 1;
         //double h1 = 1;

         if( Airswc.isON( this.swcIn )){
            this.calc_PID_cool( this.swcIn );
            this.calc_PID_heat( this.swcIn );
            this.calc_PID_I_cool( this.swcIn );
            this.calc_PID_I_heat( this.swcIn );

            //Sensible
            if(  this.valOutCtrl > 0 ){
               //前回の操作が加熱操作の時はハンチング回避のため冷却しない
               if( this.isPlusPreValHeat ){
                  this.isPlusPreValCool = false;
                  this.isPlusPreValHeat = false;
                  sensibleHeatRate = 0;
               }else{
                  this.isPlusPreValCool = true;
                  sensibleHeatRate = - this.valOutCtrl * this.floorArea * this.desCoolS ;
               }
               //System.out.println( this.name+" 冷房 "+dt+ " /"+this.valOutCtrl + " /"+sensibleHeatRate );
            }else if( this.h_valOutCtrl > 0 ){
               //前回の操作が冷却操作の時はハンチング回避のため加熱しない
               if( this.isPlusPreValCool ){
                  this.isPlusPreValCool = false;
                  this.isPlusPreValHeat = false;
                  sensibleHeatRate = 0;
               }else{
                  this.isPlusPreValHeat = true;
                  sensibleHeatRate = this.h_valOutCtrl * this.floorArea * this.desHeatS ;
               }
               //System.out.println( this.name+" 暖房 "+dt);
            }else{
               this.isPlusPreValHeat = false;
               this.isPlusPreValCool = false;
               sensibleHeatRate = 0;
            }

            //Latent
            if( this.h_I_valOutCtrl > 0 ){
               //前回の操作が除湿操作の時はハンチング回避のため加湿しない
```

```
            if( this.isPlusPreValCoolL ){
               this.isPlusPreValCoolL = false;
               this.isPlusPreValHeatL = false;
               latentHeatRate = 0;
            }else{
               this.isPlusPreValHeatL = true;
               latentHeatRate = this.h_l_valOutCtrl * this.floorArea * this.desHeatL ;
            }
         }else if(  this.l_valOutCtrl > 0 ){
            //前回の操作が加湿操作の時はハンチング回避のため除湿しない
            if( this.isPlusPreValHeatL ){
               this.isPlusPreValCoolL = false;
               this.isPlusPreValHeatL = false;
               latentHeatRate = 0;
            }else{
               this.isPlusPreValCoolL = true;
               latentHeatRate = - this.l_valOutCtrl * this.floorArea * this.desCoolL ;
            }
         }else {
            this.isPlusPreValHeatL = false;
            this.isPlusPreValCoolL = false;
            latentHeatRate = 0;
         }

      }else{
         this.calc_PID_cool( this.swcIn );
         this.calc_PID_heat( this.swcIn );
         this.calc_PID_l_cool( this.swcIn );
         this.calc_PID_l_heat( this.swcIn );

         this.isPlusPreValHeatL = false;
         this.isPlusPreValCoolL = false;
         sensibleHeatRate = 0;
         latentHeatRate   = 0;
         //System.out.println( this.name+" 中間or停止 "+this.envOut.getTa());
      }

      //室負荷計算
      double[] spLoad=new double[3];
      //double[] tx=coreSpace.getValues(ISpace.SPACETX);
      //spLoad[0]=-(coeffT[2]+coeffT[0]*tx[0]); //室顕熱負荷（冷房：正）
      //spLoad[1]=-(coeffX[2]+coeffX[0]*tx[1]); //室潜熱負荷（除湿：正）
      //spLoad[2]=spLoad[0]+spLoad[1];        //室全熱負荷
      spLoad[0]=-(sensibleHeatRate);    //室顕熱負荷（冷房：正）
      spLoad[1]=-(latentHeatRate);   //室潜熱負荷（除湿：正）
      spLoad[2]=spLoad[0]+spLoad[1];          //室全熱負荷
      //建物側へ室負荷値を設定
      coreSpace.setValues(ISpace.SPACELOAD, spLoad);

   // heatCRLPAL.clear();
   // systemHeat.clear();
      //heatCRLPAL.addHeat(sensibleHeatRateHigh, sensibleHeatRate, latentHeatRate);
   // heatCRLPAL.addHeat( sensibleHeatRate, latentHeatRate, 0);
   // systemHeat.addHeatGain(heatCRLPAL, BestTimeManager.getCurrentInterval());
      this.systemHeatGain.integrateHeatGain(
            0.3*sensibleHeatRateHigh+0.5*sensibleHeatRate,
            0.7*sensibleHeatRateHigh+0.5*sensibleHeatRate,
            sensibleHeatRate,
            BestTimeManager.getCurrentInterval());
   }

   //記録
   //本モジュールで設定した運転スケジュールを使用する
   //dateTime = BestTimeManager.getDateWeatherTime();
   this.timeInterval = BestTimeManager.getCurrentInterval();
   //double[] spaceEnv = this.coreSpace.getValues(ISpace.SPACEENV);
   //
   //this.isRecordPAL = true;
   //
   if( this.isCalcAnnualLoad && this.isRecord  && super.rm != null ){
      //PALを記録する時は　月末の24時のみ
      //System.out.print( BestTimeManager.getDateTime() + " this.timeInterval ="+this.timeInterval + " / ");
```

```
        //顕熱
        if( this.coreSpace.getZoneResult().getSpLoadS()>0 ){
           //System.out.print( " S-COOL/ "+(int)this.coreSpace.getZoneResult().getSpLoadS());
           this.cSHL1m2 += ( this.coreSpace.getZoneResult().getSpLoadS() * this.timeInterval);
        }else{
           //System.out.print( " S-HEAT/ "+(int)this.coreSpace.getZoneResult().getSpLoadS());
           this.hSHL1m2 += ( this.coreSpace.getZoneResult().getSpLoadS() * this.timeInterval);
        }
        //潜熱
        if( this.coreSpace.getZoneResult().getSpLoadL() >0 ){
           //System.out.print( " L-HEAT/ "+(int)this.coreSpace.getZoneResult().getSpLoadL());
           this.cLHL1m2 += ( this.coreSpace.getZoneResult().getSpLoadL() * this.timeInterval);
        }else{
           //System.out.print( " L-HEAT/ "+(int)this.coreSpace.getZoneResult().getSpLoadL());
           this.hLHL1m2 += ( this.coreSpace.getZoneResult().getSpLoadL() * this.timeInterval);
        }
        //全熱
        if( ( this.coreSpace.getZoneResult().getSpLoadS() + this.coreSpace.getZoneResult().getSpLoadL() )>0){
           //System.out.print( " T-HEAT/ "+(int)(this.coreSpace.getZoneResult().getSpLoadS() +
this.coreSpace.getZoneResult().getSpLoadL()));
           this.cTHL1m2 +=(( this.coreSpace.getZoneResult().getSpLoadS() + this.coreSpace.getZoneResult().getSpLoadL() )
* this.timeInterval );
        }else{
           //System.out.print( " T-HEAT/ "+(int)(this.coreSpace.getZoneResult().getSpLoadS() +
this.coreSpace.getZoneResult().getSpLoadL()));
           this.hTHL1m2 +=(( this.coreSpace.getZoneResult().getSpLoadS() + this.coreSpace.getZoneResult().getSpLoadL() )
* this.timeInterval );
        }
        //System.out.println( " / ");


        if( this.isRecordPAL ){
           if( dateTime[5]==24 ){
              if( dateTime[6]==0 ){
                 int endMD = dateTime[1] * 100 + dateTime[2];
                 if( endMD==131 || endMD==228 || endMD==331 ||
                       endMD==430 || endMD==531 || endMD==630 ||
                       endMD==731 || endMD==831 || endMD==930 ||
                       endMD==1031 || endMD==1130 || endMD==1231 ){
                    this.record2();
                    //System.out.println( "month / day = " + dateTime[1]+" / " +dateTime[2] );
                 }
              }
           }
        }else{
           record3();
        }
     }else if( this.isRecord  && super.rm != null ){
        if( Airmod.isCOOL( this.modIn )){
           //冷房
           this.cSHL1m2 += ( this.coreSpace.getZoneResult().getSpLoadS() * this.timeInterval);
           this.cLHL1m2 += ( this.coreSpace.getZoneResult().getSpLoadL() * this.timeInterval);
           this.cTHL1m2 +=(( this.coreSpace.getZoneResult().getSpLoadS() + this.coreSpace.getZoneResult().getSpLoadL() )
* this.timeInterval );
        }else if( Airmod.isHEAT( this.modIn)){
           //暖房
           this.hSHL1m2 += ( this.coreSpace.getZoneResult().getSpLoadS() * this.timeInterval);
           this.hLHL1m2 += ( this.coreSpace.getZoneResult().getSpLoadL() * this.timeInterval);
           this.hTHL1m2 +=(( this.coreSpace.getZoneResult().getSpLoadS() + this.coreSpace.getZoneResult().getSpLoadL() )
* this.timeInterval );
        }
        if( this.isRecordPAL ){
           if( dateTime[5]==24 ){
              if( dateTime[6]==0 ){
                 int endMD = dateTime[1] * 100 + dateTime[2];
                 if( endMD==131 || endMD==228 || endMD==331 ||
                       endMD==430 || endMD==531 || endMD==630 ||
                       endMD==731 || endMD==831 || endMD==930 ||
                       endMD==1031 || endMD==1130 || endMD==1231 ){
                    this.record2();
                    //System.out.println( "month / day = " + dateTime[1]+" / " +dateTime[2] );
                 }
```

```
                }
            }
        }else{
            record3();
        }
    }

    //グラフデータ追加
    if( this.isGVisible ){
        gZone.addData( BestTimeManager.getDateWeatherTime(), this.coreSpace.getZoneEnv(), this.airIn, this.airOut,
this.eleInConcent, this.eleInLighting );
    }

    if( this.isGDBVisible ){
        this.gZoneEnvhDB.addData(this.swcIn, this.modIn, this.envOut );
    }

    if( this.isGRHVisible ){
        this.gZoneEnvhRH.addData(this.swcIn, this.modIn, this.envOut );
    }

    if( this.isGPMVVisible ){
        this.gZoneEnvhPMV.addData(this.swcIn, this.modIn, this.envOut );
    }

    message.setLength(0);
  }

  double sensibleHeatRateHigh;
  double sensibleHeatRate;
  double latentHeatRate;

  //private double valObs; //観測値
  private double valOutCtrl;//制御量
  private double h_valOutCtrl;//制御量
  private double l_valOutCtrl;//制御量
  private double h_l_valOutCtrl;//制御量

  private boolean isPlusPreValCool =false;
  private boolean isPlusPreValHeat =false;
  private boolean isPlusPreValCoolL =false;
  private boolean isPlusPreValHeatL =false;
  //cool
  private double sp;      //設定値
  private double propGain;   //比例ゲイン[b/a]
  private double inteTime;   //積分時間
  private double diffTime;   //微分時間
  private double ctrlOff; // off時の操作量[b]
  private double ctrlRef; // 操作量の参照値[b]
  private double ctrlMin; // 操作量の最小値[b]
  private double ctrlMax; // 操作量の最大値[b]
  private double dtime;      //計算時間間隔
  //
  private int consecutiveStepOn;      // OnOff!=0となってから何ステップ目か
  private double pvPrevious;       // 前ステップの制御量[a]
  private double pvPrevPrev;       // 前々ステップの制御量[a]
  private double ctrlPrevious;     // 前ステップの操作量[b]
  private double kp;               // 比例ゲインに相当[b/a]
  private double ki;               // 積分ゲイン[b/a]
  private double kd;               // 微分ゲイン[b/a]
  int ctrlMode ; //=1:  設定値を上回ると操作量を増やす
   //=-1: 設定値を上回ると操作量を減らす

  private void calc_PID_cool( int swcIn ){
    //PID
    double valObs; //観測値
    //this.sp = 26;
    this.propGain = 0.03;
    this.dtime = 300;
    this.inteTime = 300;
    this.diffTime = 300;
    this.ctrlMode = 1;
```

```
    this.ctrlMax = 1.;
    this.ctrlMin = 0;
    this.ctrlOff = 0;
    this.ctrlRef = 0.1;

    valObs = this.envOut.getTa();

    if( this.inteTime < 0.001 ){
       this.ki = 0.0 ;   //積分時間=0とした場合には積分項目無しとして扱う
    }else{
       this.ki = this.propGain * this.dtime / this.inteTime ;
    }

    this.kp = this.propGain - this.ki / 2.0 ;

    //０での除算を避ける
    //(deltaTが0だった場合、kdに0が入ってしまうので修正が必要)
    if( this.inteTime != 0 ){
       this.kd = this.propGain * diffTime / dtime;
    }


    // 操作量の計算
    if( Airswc.isOFF( swcIn )){
       this.consecutiveStepOn = 0;
       this.valOutCtrl = this.ctrlOff;
    }else{
       // 制御量
       if( this.consecutiveStepOn == 1 ){
          if( ( valObs - this.sp ) * this.ctrlMode > 0 ){
             this.valOutCtrl = this.ctrlRef;
          }else{
             //スタート時に条件内であれば何もしない
             this.valOutCtrl = this.ctrlOff;
             this.consecutiveStepOn = 0;
          }
       }else if( this.consecutiveStepOn == 2 ){
          this.valOutCtrl =  this.ctrlPrevious
             + this.propGain * ( valObs - this.pvPrevious ) * this.ctrlMode;
       }else{
          double du
          = this.kp * ( valObs - this.pvPrevious ) + this.ki * ( valObs - this.sp )
                + this.kd * ( valObs + this.pvPrevPrev - 2.0 * this.pvPrevious );
          this.valOutCtrl = this.ctrlPrevious + du * this.ctrlMode;
       }
       // 制御量の上下限設定
       if( this.valOutCtrl > this.ctrlMax){
          this.valOutCtrl = this.ctrlMax;
       }else if( this.valOutCtrl < this.ctrlMin){
          this.valOutCtrl = this.ctrlMin;
       }
    }

    // 過去の制御量の保管
    this.pvPrevPrev = this.pvPrevious;
    this.pvPrevious = valObs;
    // 操作量の保管
    this.ctrlPrevious = this.valOutCtrl;
    // 制御継続ステップ数の更新
    this.consecutiveStepOn++;
}

//heat
private double h_sp;    //設定値
private double h_propGain; //比例ゲイン[b/a]
private double h_inteTime; //積分時間
private double h_diffTime; //微分時間
private double h_ctrlOff;  // off時の操作量[b]
private double h_ctrlRef;  // 操作量の参照値[b]
private double h_ctrlMin;  // 操作量の最小値[b]
private double h_ctrlMax;  // 操作量の最大値[b]
private double h_dtime;    //計算時間間隔
```

```
//
private int h_consecutiveStepOn;    // OnOff!=0となってから何ステップ目か
private double h_pvPrevious;        // 前ステップの制御量[a]
private double h_pvPrevPrev;        // 前々ステップの制御量[a]
private double h_ctrlPrevious;      // 前ステップの操作量[b]
private double h_kp;              // 比例ゲインに相当[b/a]
private double h_ki;              // 積分ゲイン[b/a]
private double h_kd;              // 微分ゲイン[b/a]
int h_ctrlMode ;  //=1:  設定値を上回ると操作量を増やす

private void calc_PID_heat( int swcIn ){
   //PID
   double valObs; //観測値
   //this.h_sp = 22;
   this.h_propGain = 0.03;
   this.h_dtime = 300;
   this.h_inteTime = 300;
   this.h_diffTime = 300;
   this.h_ctrlMode = -1;
   this.h_ctrlMax = 1.;
   this.h_ctrlMin = 0;
   this.h_ctrlOff = 0;
   this.h_ctrlRef = 0.05;

   valObs = this.envOut.getTa();

   if( this.h_inteTime < 0.001 ){
      this.h_ki = 0.0 ; //積分時間=0とした場合には積分項目無しとして扱う
   }else{
      this.h_ki = this.h_propGain * this.h_dtime / this.h_inteTime ;
   }

   this.h_kp = this.h_propGain - this.h_ki / 2.0 ;

   //０での除算を避ける
   //(deltaTが0だった場合、kdに0が入ってしまうので修正が必要)
   if( this.h_inteTime != 0 ){
      this.h_kd = this.h_propGain * h_diffTime / h_dtime;
   }


   // 操作量の計算
   if( Airswc.isOFF( swcIn )){
      this.h_consecutiveStepOn = 0;
      this.h_valOutCtrl = this.h_ctrlOff;
   }else{
      // 制御量
      if( this.h_consecutiveStepOn == 1 ){
         if( ( valObs - this.h_sp ) * this.h_ctrlMode > 0 ){
            this.h_valOutCtrl = this.h_ctrlRef;
         }else{
            //スタート時に条件内であれば何もしない
            this.h_valOutCtrl = this.h_ctrlOff;
            this.h_consecutiveStepOn = 0;
         }
      }else if( this.h_consecutiveStepOn == 2 ){
         this.h_valOutCtrl =  this.h_ctrlPrevious
            + this.h_propGain * ( valObs - this.h_pvPrevious ) * this.h_ctrlMode;
      }else{
         double du
         = this.h_kp * ( valObs - this.h_pvPrevious ) + this.h_ki * ( valObs - this.h_sp )
               + this.h_kd * ( valObs + this.h_pvPrevPrev - 2.0 * this.h_pvPrevious );
         this.h_valOutCtrl = this.h_ctrlPrevious + du * this.h_ctrlMode;
      }
      // 制御量の上下限設定
      if( this.h_valOutCtrl > this.h_ctrlMax){
         this.h_valOutCtrl = this.h_ctrlMax;
      }else if( this.h_valOutCtrl < this.h_ctrlMin){
         this.h_valOutCtrl = this.h_ctrlMin;
      }
   }
```

```
    // 過去の制御量の保管
    this.h_pvPrevPrev = this.h_pvPrevious;
    this.h_pvPrevious = valObs;
    // 操作量の保管
    this.h_ctrlPrevious = this.h_valOutCtrl;
    // 制御継続ステップ数の更新
    this.h_consecutiveStepOn++;
}

//l_cool
private double l_sp;    //設定値
private double l_propGain; //比例ゲイン[b/a]
private double l_inteTime; //積分時間
private double l_diffTime; //微分時間
private double l_ctrlOff;  // off時の操作量[b]
private double l_ctrlRef;  // 操作量の参照値[b]
private double l_ctrlMin;  // 操作量の最小値[b]
private double l_ctrlMax;  // 操作量の最大値[b]
private double l_dtime;    //計算時間間隔
//
private int l_consecutiveStepOn;    // OnOff!=0となってから何ステップ目か
private double l_pvPrevious;        // 前ステップの制御量[a]
private double l_pvPrevPrev;        // 前々ステップの制御量[a]
private double l_ctrlPrevious;      // 前ステップの操作量[b]
private double l_kp;              // 比例ゲインに相当[b/a]
private double l_ki;              // 積分ゲイン[b/a]
private double l_kd;              // 微分ゲイン[b/a]
int l_ctrlMode ;  //=1:  設定値を上回ると操作量を増やす
 //=-1: 設定値を上回ると操作量を減らす

/**
 * 除湿用
 */
private void calc_PID_l_cool( int swcIn ){
    //PID
    double valObs; //観測値
    //this.l_sp = 50;
    this.l_propGain = 0.01;
    this.l_dtime = 300;
    this.l_inteTime = 300;
    this.l_diffTime = 300;
    this.l_ctrlMode = 1;
    this.l_ctrlMax = 1.;
    this.l_ctrlMin = 0;
    this.l_ctrlOff = 0;
    this.l_ctrlRef = 0.01;

    valObs = this.envOut.getRh();

    if( this.l_inteTime < 0.001 ){
        this.l_ki = 0.0 ; //積分時間=0とした場合には積分項目無しとして扱う
    }else{
        this.l_ki = this.l_propGain * this.l_dtime / this.l_inteTime ;
    }

    this.l_kp = this.l_propGain - this.l_ki / 2.0 ;

    //０での除算を避ける
    //(deltaTが0だった場合、kdに0が入ってしまうので修正が必要)
    if( this.l_inteTime != 0 ){
        this.l_kd = this.l_propGain * l_diffTime / l_dtime;
    }


    // 操作量の計算
    if( Airswc.isOFF( swcIn )){
        this.l_consecutiveStepOn = 0;
        this.l_valOutCtrl = this.l_ctrlOff;
    }else{
        // 制御量
        if( this.l_consecutiveStepOn == 1 ){
            if( ( valObs - this.l_sp ) * this.l_ctrlMode > 0 ){
```

```
            this.l_valOutCtrl = this.l_ctrlRef;
         }else{
            //スタート時に条件内であれば何もしない
            this.l_valOutCtrl = this.l_ctrlOff;
            this.l_consecutiveStepOn = 0;
         }
      }else if( this.l_consecutiveStepOn == 2 ){
         this.l_valOutCtrl =  this.l_ctrlPrevious
            + this.l_propGain * ( valObs - this.l_pvPrevious ) * this.l_ctrlMode;
      }else{
         double du
         = this.l_kp * ( valObs - this.l_pvPrevious ) + this.l_ki * ( valObs - this.l_sp )
               + this.l_kd * ( valObs + this.l_pvPrevPrev - 2.0 * this.l_pvPrevious );
         this.l_valOutCtrl = this.l_ctrlPrevious + du * this.l_ctrlMode;
      }
      // 制御量の上下限設定
      if( this.l_valOutCtrl > this.l_ctrlMax){
         this.l_valOutCtrl = this.l_ctrlMax;
      }else if( this.l_valOutCtrl < this.l_ctrlMin){
         this.l_valOutCtrl = this.l_ctrlMin;
      }
   }

   // 過去の制御量の保管
   this.l_pvPrevPrev = this.l_pvPrevious;
   this.l_pvPrevious = valObs;
   // 操作量の保管
   this.l_ctrlPrevious = this.l_valOutCtrl;
   // 制御継続ステップ数の更新
   this.l_consecutiveStepOn++;
}

//l_heat
private double h_l_sp;     //設定値
private double h_l_propGain;  //比例ゲイン[b/a]
private double h_l_inteTime;  //積分時間
private double h_l_diffTime;  //微分時間
private double h_l_ctrlOff;   // off時の操作量[b]
private double h_l_ctrlRef;   // 操作量の参照値[b]
private double h_l_ctrlMin;   // 操作量の最小値[b]
private double h_l_ctrlMax;   // 操作量の最大値[b]
private double h_l_dtime;     //計算時間間隔
//
private int h_l_consecutiveStepOn;     // OnOff!=0となってから何ステップ目か
private double h_l_pvPrevious;         // 前ステップの制御量[a]
private double h_l_pvPrevPrev;         // 前々ステップの制御量[a]
private double h_l_ctrlPrevious;    // 前ステップの操作量[b]
private double h_l_kp;              // 比例ゲインに相当[b/a]
private double h_l_ki;              // 積分ゲイン[b/a]
private double h_l_kd;              // 微分ゲイン[b/a]
int h_l_ctrlMode ;   //=1:  設定値を上回ると操作量を増やす

/**
 * 加湿用
 */
private void calc_PID_l_heat( int swcIn ){
   //PID
   double valObs; //観測値
   //this.h_l_sp = 40;
   this.h_l_propGain = 0.01;
   this.h_l_dtime = 300;
   this.h_l_inteTime = 300;
   this.h_l_diffTime = 300;
   this.h_l_ctrlMode = -1;
   this.h_l_ctrlMax = 1.;
   this.h_l_ctrlMin = 0;
   this.h_l_ctrlOff = 0;
   this.h_l_ctrlRef = 0.01;

   valObs = this.envOut.getRh();

   if( this.h_l_inteTime < 0.001 ){
```

```
      this.h_l_ki = 0.0 ;  //積分時間=0とした場合には積分項目無しとして扱う
    }else{
      this.h_l_ki = this.h_l_propGain * this.h_l_dtime / this.h_l_inteTime ;
    }

    this.h_l_kp = this.h_l_propGain - this.h_l_ki / 2.0 ;

    //０での除算を避ける
    //(deltaTが0だった場合、kdに0が入ってしまうので修正が必要)
    if( this.h_l_inteTime != 0 ){
      this.h_l_kd = this.h_l_propGain * h_l_diffTime / h_l_dtime;
    }


    // 操作量の計算
    if( Airswc.isOFF( swcIn )){
      this.h_l_consecutiveStepOn = 0;
      this.h_l_valOutCtrl = this.h_l_ctrlOff;
    }else{
      // 制御量
      if( this.h_l_consecutiveStepOn == 1 ){
        if( ( valObs - this.h_l_sp ) * this.h_l_ctrlMode > 0 ){
          this.h_l_valOutCtrl = this.h_l_ctrlRef;
        }else{
          //スタート時に条件内であれば何もしない
          this.h_l_valOutCtrl = this.h_l_ctrlOff;
          this.h_l_consecutiveStepOn = 0;
        }
      }else if( this.h_l_consecutiveStepOn == 2 ){
        this.h_l_valOutCtrl =  this.h_l_ctrlPrevious
          + this.h_l_propGain * ( valObs - this.h_l_pvPrevious ) * this.h_l_ctrlMode;
      }else{
        double du
        = this.h_l_kp * ( valObs - this.h_l_pvPrevious ) + this.h_l_ki * ( valObs - this.h_l_sp )
            + this.h_l_kd * ( valObs + this.h_l_pvPrevPrev - 2.0 * this.h_l_pvPrevious );
        this.h_l_valOutCtrl = this.h_l_ctrlPrevious + du * this.h_l_ctrlMode;
      }
      // 制御量の上下限設定
      if( this.h_l_valOutCtrl > this.h_l_ctrlMax){
        this.h_l_valOutCtrl = this.h_l_ctrlMax;
      }else if( this.h_l_valOutCtrl < this.h_l_ctrlMin){
        this.h_l_valOutCtrl = this.h_l_ctrlMin;
      }
    }

    // 過去の制御量の保管
    this.h_l_pvPrevPrev = this.h_l_pvPrevious;
    this.h_l_pvPrevious = valObs;
    // 操作量の保管
    this.h_l_ctrlPrevious = this.h_l_valOutCtrl;
    // 制御継続ステップ数の更新
    this.h_l_consecutiveStepOn++;
  }



  /**
   * 記録
   */
  private void record2(){
    //System.out.println( "zone 20081212 print in");
    super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
          this.RECORD_cSHL1m2, this.name,
          this.cSHL1m2/1000000. ); //
    super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
          this.RECORD_hSHL1m2, this.name,
          this.hSHL1m2/1000000. ); //
    super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
          this.RECORD_cTHL1m2, this.name,
          this.cTHL1m2/1000000. ); //
    super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
          this.RECORD_hTHL1m2, this.name,
```

```
            this.hTHL1m2/1000000. ); //

        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_cSHL, this.name,
                this.cSHL1m2 * this.floorArea/1000000. ); //
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_hSHL, this.name,
                this.hSHL1m2 * this.floorArea/1000000. ); //
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_cTHL, this.name,
                this.cTHL1m2 * this.floorArea/1000000. ); //
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_hTHL, this.name,
                this.hTHL1m2 * this.floorArea/1000000. ); //

        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                "test", this.name,
                this.sensibleHeatRate/1000000. ); //
    }
    private void record3(){
        //this.isRecord = false;
        if( this.isRecord  && super.rm != null ){
            if( CheckPrintModule.isPrintLoad ){

                //System.out.println( "zone 20081212 print in");
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_cSHL1m2, this.name,
                        this.cSHL1m2/1000000. ); //
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_cLHL1m2, this.name,
                        this.cLHL1m2/1000000. ); //
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_cTHL1m2, this.name,
                        this.cTHL1m2/1000000. ); //
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_hSHL1m2, this.name,
                        this.hSHL1m2/1000000. ); //
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_hLHL1m2, this.name,
                        this.hLHL1m2/1000000. ); //
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_hTHL1m2, this.name,
                        this.hTHL1m2/1000000. ); //

                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_cSHL, this.name,
                        this.cSHL1m2 * this.floorArea/1000000. ); //
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_cLHL, this.name,
                        this.cLHL1m2 * this.floorArea/1000000. ); //
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_cTHL, this.name,
                        this.cTHL1m2 * this.floorArea/1000000. ); //
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_hSHL, this.name,
                        this.hSHL1m2 * this.floorArea/1000000. ); //
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_hLHL, this.name,
                        this.hLHL1m2 * this.floorArea/1000000. ); //
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        this.RECORD_hTHL, this.name,
                        this.hTHL1m2 * this.floorArea/1000000. ); //


                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        "test", this.name,
                        this.sensibleHeatRate/1000000. ); //
            }
        }
    }

    public Object viewInternal(TestCommand cmd) {
```

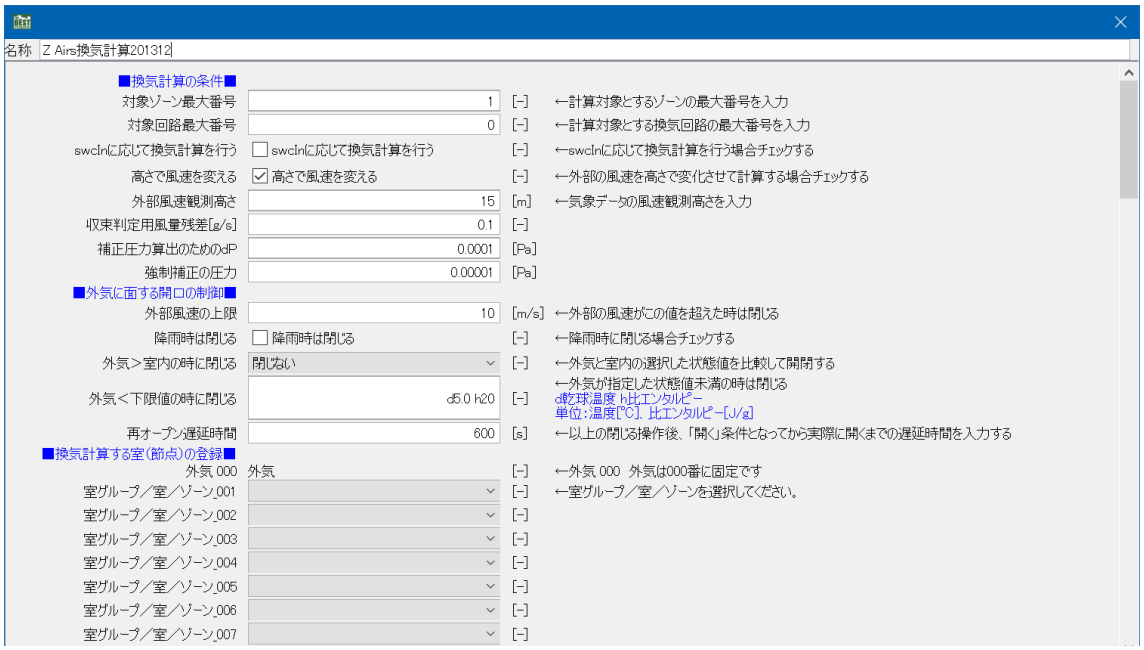
```
        List<Object> result = new ArrayList<Object>();
        result.add(super.cm);
        result.add(super.rm);
        result.add(super.sm);
        return result;
    }
}
```

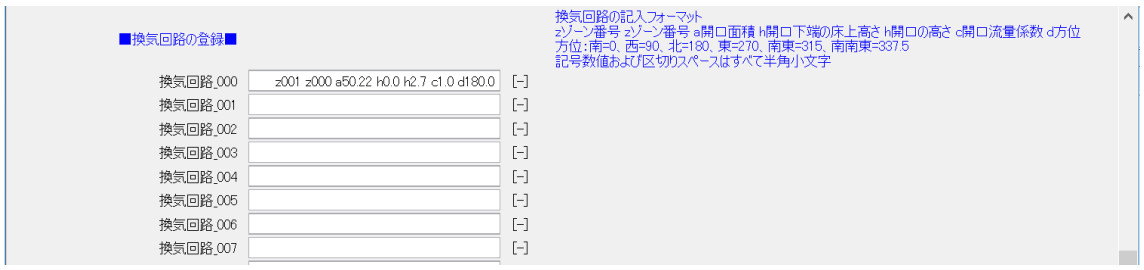
「Z Airs 換気計算 201312」（場所：設備 2015／Z ゾーン 2015／）

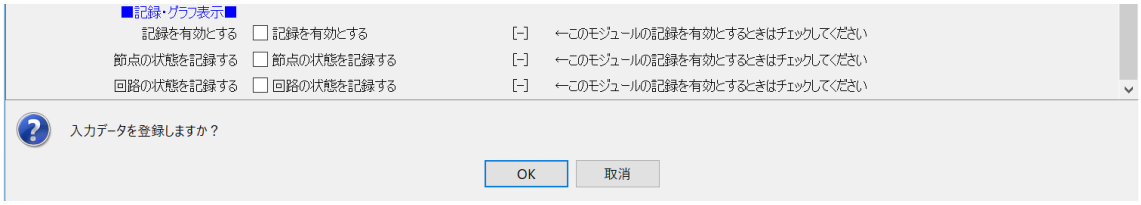
モジュール名	Z Airs 換気計算 201312
クラス	ZoneAirsVentModule201312*

（1）入力画面

・スペック







（２）モジュールの概要

還気回路網計算を行うモジュールである。

回路数、節点数に制限はない。入力画面は各３０個の入力枠を用意しているが、足りない場合は、入力画面の定義ファイルを修正して使用すること。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

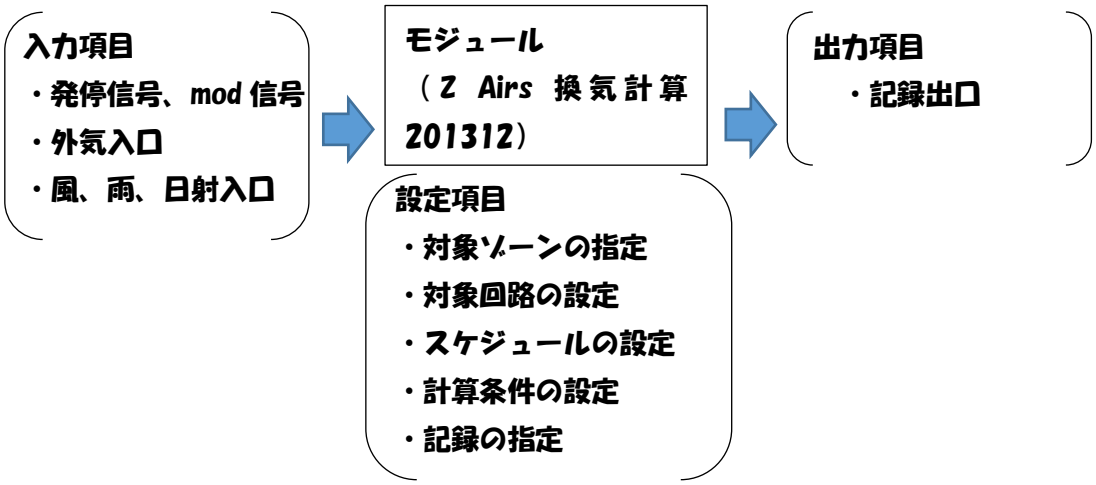


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	isVentRecord	ー	[-]	ー	ー		
1	■換気計算の条件■					ー	ー		
2	対象ゾーン最大番号	int	numZone	1	[-]	ー	ー		←計算対象とするゾーンの最大番号を入力
3	対象回路最大番号	int	numVent	0	[-]	ー	ー		←計算対象とする換気回路の最大番号を入力
4	swcIn に応じて換気計算を行う	boolean	isVentOpe_swcIn	FALSE	[-]	ー	ー		←swcIn に応じて換気計算を行う場合チェックする
5	高さで風速を変える	boolean	isChange_V_winheight	TRUE	[-]	ー	ー		←外部の風速を高さで変化させて計算する場合チェックする
6	外部風速観測高さ	int	measureHeight_V_win	15	[m]	ー	ー		←外部の風速を高さで変化させて計算する場合チェックする
7	収束判定用風量残差[g/s]	double	vLimit	0.1	[-]	1	0		
8	補正圧力算出のための dP	double	dP	0.0001	[Pa]	10	0		
9	強制補正の圧力	double	hoseidP	0.00001	[Pa]	1	0		
10	■外気に面する開口の制御■					ー	ー		
11	外部風速の上限	double	maxV_OA	10	[m/s]	ー	0		←外部の風速がこの値を超えた時は閉じる
12	降雨時は閉じる	boolean	isCloseRain	FALSE	[-]	ー	ー		←降雨時に閉じる場合チェックする
13	外気＞室内の時に閉じる	String	compareOutIn	閉じない、乾球温度、比エンタルピ、乾球温度または比エンタルピ、乾球温度かつ比エンタルピ	[-]	ー	ー		←外気と室内の選択した状態値を比較して開閉する

14	外気＜下限値の時に閉じる	String	closeLowerValue	d5.0 h20	[-]	―	―		<html>←外気が指定した状態値未満の時は閉じる<font color=blue> d 乾球温度 h 比エンタルピー 単位：温度[℃]、比エンタルピー[J/g]
15	再オープン遅延時間	double	delaySeconds	600	[s]	―	0		←以上の閉じる操作後、「開く」条件となってから実際に開くまでの遅延時間を入力する
16	■換気計算する室（節点）の登録■					―	―		
17	外気_000	String		外気	[-]	―	―		←外気_000　外気は 000 番に固定です
18	室グループ／室／ゾーン_001	String	sysSpace[]	#RoomGroup	[-]	―	―		←室グループ／室／ゾーンを選択してください。
19	室グループ／室／ゾーン_002	String	sysSpace[]	#RoomGroup	[-]	―	―		
20	室グループ／室／ゾーン_003	String	sysSpace[]	#RoomGroup	[-]	―	―		
21	室グループ／室／ゾーン_004	String	sysSpace[]	#RoomGroup	[-]	―	―		
22	室グループ／室／ゾーン_005	String	sysSpace[]	#RoomGroup	[-]	―	―		
23	室グループ／室／ゾーン_006	String	sysSpace[]	#RoomGroup	[-]	―	―		
24	室グループ／室／ゾーン_007	String	sysSpace[]	#RoomGroup	[-]	―	―		
25	室グループ／室／ゾーン_008	String	sysSpace[]	#RoomGroup	[-]	―	―		
26	室グループ／室／ゾーン_009	String	sysSpace[]	#RoomGroup	[-]	―	―		
27	室グループ／室／ゾーン_010	String	sysSpace[]	#RoomGroup	[-]	―	―		
28	室グループ／室／ゾーン_011	String	sysSpace[]	#RoomGroup	[-]	―	―		←室グループ／室／ゾーンを選択してください。
29	室グループ／室／ゾーン_012	String	sysSpace[]	#RoomGroup	[-]	―	―		
30	室グループ／室／ゾーン_013	String	sysSpace[]	#RoomGroup	[-]	―	―		
31	室グループ／室／ゾーン_014	String	sysSpace[]	#RoomGroup	[-]	―	―		
32	室グループ／室／ゾーン_015	String	sysSpace[]	#RoomGroup	[-]	―	―		
33	室グループ／室／ゾーン_016	String	sysSpace[]	#RoomGroup	[-]	―	―		
34	室グループ／室／ゾーン_017	String	sysSpace[]	#RoomGroup	[-]	―	―		

35	室グループ／室／ゾーン_018	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
36	室グループ／室／ゾーン_019	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
37	室グループ／室／ゾーン_020	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
38	室グループ／室／ゾーン_021	String	sysSpace[]	#RoomGroup	[-]	ー	ー		←室グループ／室／ゾーンを選択してください。
39	室グループ／室／ゾーン_022	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
40	室グループ／室／ゾーン_023	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
41	室グループ／室／ゾーン_024	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
42	室グループ／室／ゾーン_025	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
43	室グループ／室／ゾーン_026	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
44	室グループ／室／ゾーン_027	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
45	室グループ／室／ゾーン_028	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
46	室グループ／室／ゾーン_029	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
47	室グループ／室／ゾーン_030	String	sysSpace[]	#RoomGroup	[-]	ー	ー		
48	■換気回路の登録■					ー	ー		<html><font color=blue>換気回路の記入フォーマット z ゾーン番号 z ゾーン番号 a 開口面積 h 開口下端の床上高さ h 開口上端の床上高さ c 開口流量係数 d 方位[度] 方位：南=0、西=90、北=180、東=270、南東=315、南南東=337.5 記号数値および区切りスペースはすべて半角小文字 単位：面積[m2]、高さ[m]、流量係数[0～1]
49	換気回路_000	String	zvSpec[]	z001 z000 a50.22 h0.0 h2.7 c1.0 d180.0	[-]	ー	ー		
50	換気回路_001	String	zvSpec[]		[-]	ー	ー		
51	換気回路_002	String	zvSpec[]		[-]	ー	ー		

52	換気回路_003	String	zvSpec[]		[-]	—	—		
53	換気回路_004	String	zvSpec[]		[-]	—	—		
54	換気回路_005	String	zvSpec[]		[-]	—	—		
55	換気回路_006	String	zvSpec[]		[-]	—	—		
56	換気回路_007	String	zvSpec[]		[-]	—	—		
57	換気回路_008	String	zvSpec[]		[-]	—	—		
58	換気回路_009	String	zvSpec[]		[-]	—	—		
59	換気回路_010	String	zvSpec[]		[-]	—	—		
60	換気回路_011	String	zvSpec[]		[-]	—	—		
61	換気回路_012	String	zvSpec[]		[-]	—	—		
62	換気回路_013	String	zvSpec[]		[-]	—	—		
63	換気回路_014	String	zvSpec[]		[-]	—	—		
64	換気回路_015	String	zvSpec[]		[-]	—	—		
65	換気回路_016	String	zvSpec[]		[-]	—	—		
66	換気回路_017	String	zvSpec[]		[-]	—	—		
67	換気回路_018	String	zvSpec[]		[-]	—	—		
68	換気回路_019	String	zvSpec[]		[-]	—	—		
69	換気回路_020	String	zvSpec[]		[-]	—	—		
70	換気回路_021	String	zvSpec[]		[-]	—	—		
71	換気回路_022	String	zvSpec[]		[-]	—	—		
72	換気回路_023	String	zvSpec[]		[-]	—	—		
73	換気回路_024	String	zvSpec[]		[-]	—	—		

74	換気回路_025	String	zvSpec[]		[-]	ー	ー		
75	換気回路_026	String	zvSpec[]		[-]	ー	ー		
76	換気回路_027	String	zvSpec[]		[-]	ー	ー		
77	換気回路_028	String	zvSpec[]		[-]	ー	ー		
78	換気回路_029	String	zvSpec[]		[-]	ー	ー		
79	換気回路_030	String	zvSpec[]		[-]	ー	ー		
80	■換気回路のスケジュールの登録■					ー	ー		<html><font color=blue>換気回路の開閉（開放率）をスケジュールにより行う場合はそのスケジュールを選定してください swcIn 連動を指定の場合、ここで指定したスケジュールは swcIn が on の時に適用します。
81	換気回路_000_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		←DailyAnnualSchedule のスケジュール名を入力してください。
82	換気回路_001_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
83	換気回路_002_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
84	換気回路_003_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
85	換気回路_004_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
86	換気回路_005_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
87	換気回路_006_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
88	換気回路_007_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
89	換気回路_008_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
90	換気回路_009_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		

91	換気回路_010_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
92	換気回路_011_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
93	換気回路_012_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
94	換気回路_013_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
95	換気回路_014_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
96	換気回路_015_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
97	換気回路_016_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
98	換気回路_017_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
99	換気回路_018_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
100	換気回路_019_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
101	換気回路_020_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
102	換気回路_021_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
103	換気回路_022_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
104	換気回路_023_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
105	換気回路_024_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
106	換気回路_025_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
107	換気回路_026_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		

108	換気回路_027_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
109	換気回路_028_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
110	換気回路_029_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
111	換気回路_030_スケジュール	String	zvSchName[]	#Userm、時刻変動スケジュール	[-]	ー	ー		
112	■記録・グラフ表示■					ー	ー		
113	＊グラフを表示する	boolean		FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
114	＊最大同時表示ステップ数	int		100	[-]	ー	10		←グラフに同時表示する最大ステップ数を入力します
115	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください
116	節点の状態を記録する	boolean	isZoneRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください
117	回路の状態を記録する	boolean	isVentRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	外気導入入口	L0_airInOA	airInOA	-	状態	空気	入口	
5	風情報入口	L0_winIn	winIn	-	状態	風	入口	
6	雨情報入口	L0_watInRain	watInRain	-	状態	水	入口	
7	日射情報入口	L0_sunIn	sunIn	-	状態	太陽日射	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	spaceMessage#-#-	メッセージ	—	メッセージ
2	ventIn_混合乾球温度#℃#温度	ventIn_混合乾球温度	℃	入口
3	ventIn_混合絶対湿度#g/g'#絶対湿度	ventIn_混合絶対湿度	g/g	入口
4	ventIn_混合 CO2 濃度#ppm#濃度	ventIn_混合 CO2 濃度	ppm	入口
5	ventIn_合計風量#g/s#質量流量	ventIn_合計風量	g/s	入口
6	ventIn_合計風量#m3/h#体積流量	ventIn_合計風量	M3/h	入口
7	ventIn_混合密度#g/L#密度	ventIn_混合密度	g/L	入口
8	ventIn_換気回数#回/h#-	ventIn_換気回数	回/h	入口
9	ventOut_合計風量#g/s#質量流量	ventOut_合計風量	g/s	出口
10	ventOut_合計風量#m3/h#体積流量	ventOut_合計風量	M3/h	出口
11	zone_温度#℃#温度	zone_温度	℃	My
12	zone_絶対湿度#g/g'#絶対湿度	zone_絶対湿度	g/g	My
13	zone_CO2 濃度#ppm#濃度	zone_CO2 濃度	ppm	My
14	zone_密度#g/L#密度	zone_密度	g/L	My
15	zone_床面圧力#Pa#圧力	zone_床面圧力	Pa	My
16	外部風向#-#風向	外部風向	—	My
17	外部風速#m/s#風速	外部風速	m/s	My
18	外気_換気回数#回/h#-	外気_換気回数	回/h	My
19	降雨量#-#-	降雨量	—	My

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import javax.vecmath.GMatrix;
import javax.vecmath.GVector;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestSun;
import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.DO.BestWind;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.building.manager.ScheduleManager;
import jp.or.ibec.best.domain.building.manager.SpaceManager;
import jp.or.ibec.best.domain.building.schedule.DailyAnnualSchedule;
import jp.or.ibec.best.domain.building.spaces.ISpace;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * 空調吹出空気情報を建物側に渡し、リターン空気を出力するモジュール
 * 換気回路網による換気量を計算しモジュールへその換気を与えるモジュール
 * @author  kohri  NIROSHI NINONIYA/20120505
 * 20160830 nino 差圧の積分処理部分不具合修正
 */
public class ZoneAirsVentModule201312 extends AbstractBestModule implements
      IBestMetaModule, jp.or.ibec.best.dk.test.IProbeForUT {

   private final String moduleName = "(ZoneAirsVentModule201312) ";

   //外部定義項目
   private final String SPEC_name0 = "名称"; //20071122nino
   private final String SPEC_vLimit  = "収束判定用風量残差[g/s]";
   private final String SPEC_dP      = "補正圧力dP[Pa]";
   private final String SPEC_hoseidP = "強制補正圧力[Pa]";
   private final String SPEC_numZone = "対象ゾーン最大番号[-]";
   private final String SPEC_numVent = "対象回路最大番号[-]";
   private final String SPEC_isVentOpe_swcIn = "swcInに応じて換気計算を行う[-]";
   private final String SPEC_isChange_V_winheight = "高さで風速を変える[-]";
   private final String SPEC_measureHeight_V_win  = "外部風速観測高さ[m]";

   private final String SPEC_maxV_OA         = "外部風速上限[m/s]";
   private final String SPEC_isCloseRain     = "降雨時に閉じる[-]";
   private final String SPEC_compareOutIn    = "内外状態値比較し閉じる[-]";
   private final String SPEC_closeLowerValue = "外気状態値の下限[-]";
   private final String SPEC_delaySeconds    = "再オープン遅延時間[s]";

   private String[] SPEC_zvName = null;
   private String[] SPEC_zvSpec = null;
   private String[] SPEC_zvSchName = null;//"スケジュール名[-]";

   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする
   private final String SPEC_isZoneRecord = "節点の状態を記録する[-]";
   private final String SPEC_isVentRecord = "回路の状態を記録する[-]";

   private final String C_NODE_swcIn  = "L1_swcIn" ;

   private final String C_NODE_modIn  = "L1_modIn" ;

   private final String S_NODE_airInOA  = "L0_airInOA";
```

```
    private final String S_NODE_winIn    = "LO_winIn";
    private final String S_NODE_watInRain= "LO_watInRain";
    private final String S_NODE_sunIn    = "LO_sunIn";

    private String R_NODE = "L2_recOut";

    private final String RECORD_message = "spaceMessage#-#-";
    private final String RECORD_T_aIn = "ventIn_混合乾球温度#℃#温度";
    private final String RECORD_X_aIn = "ventIn_混合絶対湿度#g/g'#絶対湿度";
    private final String RECORD_CO2aIn= "ventIn_混合CO2濃度#ppm#濃度";
    private final String RECORD_M_aIn = "ventIn_合計風量#g/s#質量流量";
    private final String RECORD_V_aIn = "ventIn_合計風量#m3/h#体積流量";
    private final String RECORD_D_aIn = "ventIn_混合密度#g/L#密度";
    private final String RECORD_NV_aIn= "ventIn_換気回数#回/h#-";
    private final String RECORD_M_aOut= "ventOut_合計風量#g/s#質量流量";
    private final String RECORD_V_aOut= "ventOut_合計風量#m3/h#体積流量";
    private final String RECORD_T_aMy = "zone_温度#℃#温度";
    private final String RECORD_X_aMy = "zone_絶対湿度#g/g'#絶対湿度";
    private final String RECORD_CO2aMy= "zone_CO2濃度#ppm#濃度";
    private final String RECORD_D_aMy = "zone_密度#g/L#密度";
    private final String RECORD_PRef  = "zone_床面圧力#Pa#圧力";
//  private final String RECORD_T_aOA = "外気_温度#℃#温度";
//  private final String RECORD_X_aOA = "外気_絶対湿度#g/g'#絶対湿度";
//  private final String RECORD_D_aOA = "外気_密度#g/L#密度";
    private final String RECORD_Dir_win = "外部風向#-#風向";
    private final String RECORD_Vel_win = "外部風速#m/s#風速";
    private final String RECORD_NV_all= "外気_換気回数#回/h#-";
    private final String RECORD_MMrain = "降雨量#-#-";

    private StringBuffer message = new StringBuffer();

    private String name0 = null;
    private String[] name = null; //名称
    private double vLimit;//判定残差風量[g/s]
    private double dP;//ヤコビアンを作成するための微小圧力[Pa]
    private double hoseidP;//強制補正の微小圧力[Pa]
    private int numZone;
    private int numVent;
    private boolean isRecord = false;//記録を有効とする=true
    private boolean isZoneRecord = false;
    private boolean isVentRecord = false;

    private boolean isVentOpe_swcIn = false;//
    private boolean isChange_V_winheight = true;// = "高さで風速を変える[-]";
    private double measureHeight_V_win;// = "外部風速観測高さ[m]";

    private double maxV_OA;// = "外部風速上限[m/s]";
    private boolean isCloseRain = false;// "降雨時に閉じる[-]";
    private String compareOutIn = null;//"内外状態値比較し閉じる[-]";
    private String closeLowerValue = null;//"外気状態値の下限[-]";
    private double delaySeconds;// ="再オープン遅延時間[s]";

    private String[] zvName = null;
    private String[] zvSpec = null;
    private String[] zvSchName = null;
    private ISpace[] coreSpace = null;//建物側インスタンス
    private SystemISpace[] sysSpace = null;//20131217

    private ZoneAirSysCalc201310 zoneAirSysCalc = ZoneAirSysCalc201310.getInstance();//20131217

    /**
     * 上位からのon/off信号入力
     */
    private int swcIn;   //熱源運転状態 In

    /**
     * 上位からのmode信号入力
     */
    private int modIn;//熱源運転モード In
    private BestAir airInOA = null;
    private BestAir[] z_airIn = null;//流入空気を混合して一つのairInとしたもの
    private BestAir[][] v_airOutIn = null;//換気経路の空気出入り Out:[][0] In:[][1]
```

```
private BestWind winIn = null;
private BestWater watInRain = null;
private BestSun sunIn = null;
private BestAir[] airMy = null;

private double[] airOutFlowRate = null;

private int[] openZoneName1 = null;//1番目のz***
private int[] openZoneName2 = null;//2番目のz***
private double[] openArea = null;//開口面積a***[m2]
private double[] openHLabovezoneReference = null;//1番目のゾーン床面からの開口下端の高さ[m]
private double[] openHHabovezoneReference = null;//1番目のゾーン床面からの開口上端の高さ[m]
private double[] openHeight = null;//開口の高さ[m]
private double[] openCoef = null;//開口の流量係数[-]
private double[] openDir = null;//開口の方位[度]

private double[] opendPsL = null;//ps1L - ps2L;//開口下端のZone1基準の圧力差[Pa]
private double[] opendPsH = null;//ps1H - ps2H;//開口上端のZone1基準の圧力差[Pa]

private double[] openDelaySeconds = null;
private boolean[] isThisCloseNow = null;//

private double[] zonePReference = null;//ゾーンの床面圧力[Pa]
private double[] zonePOldReference = null;//ゾーンの床面圧力[Pa]
//private double[] dPReference = null;

private double[] zoneFloorArea = null;
private double[] zoneCeilingHeight = null;
private double[] zoneFloorLevel = null;
private double[] zoneVolume = null;

private double zoneVolumeSum;//ゾーンの体積の合計

private int compareOutInNum;//状態値内外比較
private double closeLowerValueDB;//下限値DB
private double closeLowerValueH;//下限値H


private static final double CP = 1.0;
private static final double R  = 2500.;
private static double SPa  = 101330.;//[Pa]
private static double g = 9.81;//[m/s2]

private boolean isBalance = false;

double matrixSum = 0;
double matrixSumOld = 0;

private int startSubCount = 150;
private int maxSubCount = 150;

private DailyAnnualSchedule[] das = null;
private boolean[] isAvailableDAS = null;


@Override
public void setProfile(BestSpecs spec) {
   if(spec==null) return;
   Map<String, String> map=spec.getSpec();
   if(map==null) {
      System.out.println("(E) ZoneAirModule : null spec map !!");
      return;
   }

   // 機器名称
   this.name0 = spec.getSpecValue( this.SPEC_name0, this.moduleName );

   //SPEC_vlimit = "収束判定用風量残差[g/s]"
   this.vLimit = spec.getSpecValue( this.SPEC_vLimit, 0.00001 );

   //dp圧力補正を取得
   this.dP = spec.getSpecValue( this.SPEC_dP, 0.1 );
```

```
//hoseidp強制補正圧力を取得
this.hoseidP = spec.getSpecValue( this.SPEC_hoseidP, 0.0001 );

//numZone対象ゾーン最大番号を取得
this.numZone = spec.getSpecValue( this.SPEC_numZone, 1 );

//numVentSPEC_numVentを取得
this.numVent = spec.getSpecValue( this.SPEC_numVent, 1 );

this.SPEC_zvName = new String[ this.numZone + 1 ];
this.SPEC_zvSpec = new String[ this.numVent + 1 ];
this.SPEC_zvSchName = new String[ this.numVent + 1 ];
this.das = new DailyAnnualSchedule[ this.numVent + 1 ];
this.isAvailableDAS = new boolean[ this.numVent + 1 ];

for( int i=0; i<this.SPEC_zvName.length; i++ ){
   if( i<10 ){
      this.SPEC_zvName[ i ] = "ZV00"+i;
   }else if( i<100 ){
      this.SPEC_zvName[ i ] = "ZV0"+i;
   }else{
      this.SPEC_zvName[ i ] = "ZV"+i;
   }
}
for( int i=0; i<this.SPEC_zvSpec.length; i++ ){
   if( i<10 ){
      this.SPEC_zvSpec[ i ] = "ZVS00"+i;
      this.SPEC_zvSchName[ i ] = "ZVSS00"+i;
   }else if( i<100 ){
      this.SPEC_zvSpec[ i ] = "ZVS0"+i;
      this.SPEC_zvSchName[ i ] = "ZVSS0"+i;
   }else{
      this.SPEC_zvSpec[ i ] = "ZVS"+i;
      this.SPEC_zvSchName[ i ] = "ZVSS"+i;
   }
}

this.name      = new String[ this.numZone + 1 ];
this.zvName    = new String[ this.numZone + 1 ];
this.coreSpace = new ISpace[ this.numZone + 1 ];
this.sysSpace  = new SystemISpace[ this.numZone + 1 ];//20131217

this.zonePReference    = new double[ this.zvName.length ];
this.zonePOldReference = new double[ this.zvName.length ];
//this.dPReference       = new double[ this.zvName.length ];
this.zoneFloorArea     = new double[ this.zvName.length ];
this.zoneCeilingHeight = new double[ this.zvName.length ];
this.zoneFloorLevel    = new double[ this.zvName.length ];
this.zoneVolume        = new double[ this.zvName.length ];

this.zvSpec = new String[ this.numVent + 1 ];
this.zvSchName = new String[ this.numVent + 1 ];

this.openZoneName1 = new int[ this.zvSpec.length ];
this.openZoneName2 = new int[ this.zvSpec.length ];
this.openArea                 = new double[ this.zvSpec.length ];
this.openHLabovezoneReference = new double[ this.zvSpec.length ];
this.openHHabovezoneReference = new double[ this.zvSpec.length ];
this.openHeight               = new double[ this.zvSpec.length ];
this.openCoef                 = new double[ this.zvSpec.length ];
this.openDir                  = new double[ this.zvSpec.length ];
this.opendPsL                 = new double[ this.zvSpec.length ];
this.opendPsH                 = new double[ this.zvSpec.length ];
this.openDelaySeconds         = new double[ this.zvSpec.length ];
this.isThisCloseNow           = new boolean[ this.zvSpec.length ];

//外気
this.zvName[0] = "換気外気";
this.zoneFloorArea[0]    = 0;
this.zoneFloorLevel[0]   = 0;
this.zoneCeilingHeight[0]= 0;
```

```
this.zoneVolume[0]       = 0;
this.zonePReference[0] = SPa;
this.name[0] = "換気外気";

for( int i=1; i<this.zvName.length; i++ ){
   String multiSpaceName=null;
   String zoneName=null;

   if(null!=map.get(this.SPEC_zvName[i])){
      zvName[i]=(String)map.get(this.SPEC_zvName[i]);
   }else{
      zvName[i] = "";
   }

   if( !zvName[i].contains( "/" )){
      continue;
   }

   //multiSpaceName = grzName.split("/")[0];
   //zoneName       = grzName.split("/")[2];
   if( null != zvName[i].split("/")[0] ){
      multiSpaceName = zvName[i].split("/")[0];
   }else{
      multiSpaceName = "";
   }

   if( null != zvName[i].split("/")[2] ){
      zoneName       = zvName[i].split("/")[2];
   }else{
      zoneName = "";
   }
   //名称を取得
   name[i] = multiSpaceName + "_" + zoneName;

   if( SpaceManager.getISpace(multiSpaceName, zoneName) != null ){
      //this.coreSpace[i] = SpaceManager.getISpace(multiSpaceName, zoneName);
      this.coreSpace[i] = ZoneGRZName.getCoreSpace( this.zvName[i], this.name[i] );
      this.sysSpace[i] = this.zoneAirSysCalc.getSystemISpace( this.zvName[i], this.coreSpace[i]);//20131217

      this.zoneFloorArea[i]    = this.coreSpace[i].getValue( ISpace.FLOORAREA );
      this.zoneFloorLevel[i]   = this.coreSpace[i].getValue( ISpace.FLOORLEVEL );
      this.zoneCeilingHeight[i]= this.coreSpace[i].getValue( ISpace.CEILINGHEIGHT );
      this.zoneVolume[i] = this.zoneFloorArea[i] * this.zoneCeilingHeight[i];

      this.zonePReference[i]    = SPa - this.zoneFloorLevel[i] * 1.2 * g;
      this.zonePOldReference[i] = SPa - this.zoneFloorLevel[i] * 1.2 * g;

      this.sysSpace[i].setVolume( this.zoneVolume[i] );//20131217
      this.sysSpace[i].setCO2ppm( 500. );//20131217

   }else{
      this.zoneFloorArea[i]    = 0;
      this.zoneFloorLevel[i]   = 0;
      this.zoneCeilingHeight[i]= 0;
      this.zoneVolume[i]       = 0;

      this.zonePReference[i] = SPa;
      this.zonePOldReference[i] = SPa;
   }

   this.zoneVolumeSum += this.zoneVolume[i];
}

//換気の設定条件の取り込み
for( int i=0; i<this.zvSpec.length; i++ ){

   if( null != map.get( this.SPEC_zvSpec[i] ) ){
      zvSpec[i] = (String)map.get( this.SPEC_zvSpec[i] );
      String[] str = zvSpec[i].split( " " );
      this.openZoneName1[i] = Integer.parseInt( str[0].substring( 1 ) );
      this.openZoneName2[i] = Integer.parseInt( str[1].substring( 1 ) );
      this.openArea[i]   = Double.parseDouble( str[2].substring( 1 ) );
```

```
            this.openHLabovezoneReference[i] = Double.parseDouble( str[3].substring( 1 ) );
            this.openHeight[i] = Double.parseDouble( str[4].substring( 1 ) );
            this.openHHabovezoneReference[i] = this.openHeight[i] + this.openHLabovezoneReference[i];
            this.openCoef[i]   = Double.parseDouble( str[5].substring( 1 ) );
            this.openDir[i]    = Double.parseDouble( str[6].substring( 1 ) );
        }else{
            zvSpec[i] = "";
            this.openZoneName1[i] = -1;
            this.openZoneName2[i] = -1;
            this.openArea[i]      = 0;
            this.openHLabovezoneReference[i] = 0;
            this.openHHabovezoneReference[i] = 0;
            this.openHeight[i]    = 0;
            this.openCoef[i]      = 0;
            this.openDir[i]       = 0;
        }

        //初期化
        this.opendPsL[i] = 0;
        this.opendPsH[i] = 0;
        this.openDelaySeconds[i] = 0;
        this.isThisCloseNow[i] = false;

        //
        this.zvSchName[i] = spec.getSpecValue( this.SPEC_zvSchName[i], "" );
        if( this.zvSchName[i].equals( "" )){
            this.isAvailableDAS[i] = false;
        }else{
            this.isAvailableDAS[i] = true;
            this.das[i] = ScheduleManager.getDailyScheduleManager().getDailyAnnualSchedule( this.zvSchName[i] );
        }

    }




    //isVentOpe_swcIn = "swcInに応じて換気計算を行う[-]";
    this.isVentOpe_swcIn = spec.getSpecValue( this.SPEC_isVentOpe_swcIn, false );

    //isChange_V_winheight = true;// = "高さで風速を変える[-]";
    this.isChange_V_winheight = spec.getSpecValue( this.SPEC_isChange_V_winheight, true );

    //measureHeight_V_win;// = "外部風速観測高さ[m]";
    this.measureHeight_V_win = spec.getSpecValue( this.SPEC_measureHeight_V_win, 15. );

    //SPEC_maxV_OA = "外部風速上限[m/s]";
    this.maxV_OA = spec.getSpecValue( this.SPEC_maxV_OA, 15. );

    //SPEC_isCloseRain = "降雨時に閉じる[-]";
    this.isCloseRain = spec.getSpecValue( this.SPEC_isCloseRain, true );

    //SPEC_compareOutIn = "内外状態値比較し閉じる[-]";
    this.compareOutIn = spec.getSpecValue( this.SPEC_compareOutIn, "" );

    if( this.compareOutIn.endsWith( "閉じない" )){
        this.compareOutInNum = 0;
    }else if( this.compareOutIn.endsWith( "乾球温度" )){
        this.compareOutInNum = 1;
    }else if( this.compareOutIn.endsWith( "比エンタルピ" )){
        this.compareOutInNum = 2;
    }else if( this.compareOutIn.endsWith( "乾球温度または比エンタルピ" )){
        this.compareOutInNum = 3;
    }else if( this.compareOutIn.endsWith( "乾球温度かつ比エンタルピ" )){
        this.compareOutInNum = 4;
    }



    //SPEC_closeLowerValue ="外気状態値の下限[-]";
    this.closeLowerValue = spec.getSpecValue( this.SPEC_closeLowerValue, "" );
```

```
    String[] strCLV = this.closeLowerValue.split( " " );
    this.closeLowerValueDB = Double.parseDouble( strCLV[0].substring( 1 ) );
    this.closeLowerValueH  = Double.parseDouble( strCLV[1].substring( 1 ) );

    //SPEC_delaySeconds ="再オープン遅延時間[s]";
    this.delaySeconds = spec.getSpecValue( this.SPEC_delaySeconds, 600. );

    //isRecordを取得
    this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

    //isZoneRecordを取得
    this.isZoneRecord = spec.getSpecValue( this.SPEC_isZoneRecord, false );

    //isVentRecordを取得
    this.isVentRecord = spec.getSpecValue( this.SPEC_isVentRecord, false );

}


public void initialize(IBestStateMessage stateNodes,
      IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
      IBestDomainSchedule schedule) {
    super.sm = stateNodes   ;     //状態ノードを取得
    super.cm = commandNodes ;     //制御ノードを取得
    super.rm = recordNodes  ;     //記録ノードを取得

    this.airInOA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInOA );

    this.winIn = BestWind.bindnode( super.transferMapState, super.sm, this.S_NODE_winIn );

    this.watInRain = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInRain );

    this.sunIn = BestSun.bindnode( super.transferMapState, super.sm, this.S_NODE_sunIn );


    //
    this.z_airIn = new BestAir[ this.zvName.length ];
    for( int i=0; i<this.z_airIn.length; i++ ){
      this.z_airIn[i] = new BestAir();
      //
      if( i>0 ){
        this.zoneAirSysCalc.setZoneIDandAir( this.sysSpace[i], this.z_airIn[i] );//20131217
      }
    }

    //換気経路の空気出入り Out:[][0] In:[][1]
    this.v_airOutIn = new BestAir[ this.zvSpec.length ][ 2 ];
    for( int i=0; i<this.v_airOutIn.length; i++ ){
      for( int j=0; j<this.v_airOutIn[i].length; j++ ){
        this.v_airOutIn[i][j] = new BestAir();
      }
    }

    this.airMy = new BestAir[ this.zvName.length ];
    for( int i=0; i<this.airMy.length; i++ ){
      this.airMy[i] = new BestAir();
    }

    this.airOutFlowRate = new double[ this.zvName.length ];
    for( int i=0; i<this.airOutFlowRate.length; i++ ){
      this.airOutFlowRate[i] = 0;
    }
}



public void outputs() {
    if( super.sm == null || super.rm == null) return;

    if( this.numZone <= 0 ){
      return;
    }
```

```
this.swcIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ));
this.modIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn ));

this.airInOA
=(BestAir)super.sm.getState( super.getConnectionNode( this.S_NODE_airInOA ));
this.winIn
=(BestWind)super.sm.getState( super.getConnectionNode( this.S_NODE_winIn ));
this.watInRain
=(BestWater)super.sm.getState( super.getConnectionNode( this.S_NODE_watInRain ));
this.sunIn
=(BestSun)super.sm.getState( super.getConnectionNode( this.S_NODE_sunIn ));

//外気
airMy[0].copyAllState( this.airInOA );
//System.out.println( "OA.humi=" + airMy[0].getHumi() );

//部屋の数だけ  airMy状態値の設定
for(int i=1; i<this.coreSpace.length; i++){

   //if( this.coreSpace[i] == null ){
   if( this.sysSpace[i] == null ){
      continue;
   }
   //室状態値の取得
   //double[] tx = coreSpace[i].getValues( ISpace.SPACETX );
   double[] tx = this.sysSpace[i].getISpace().getValues( ISpace.SPACETX );
   if(tx==null) {
      System.out.println("SpaceModule >>Error<< spaceTX data are not defined !!");
      return;
   }

   if( tx[1] < 0 ){
      System.out.println( "tx[1] Humi "+tx[1]+"< 0 ***** --> 0   tx[0]="+tx[0] );
      tx[1] = 0;
   }
   airMy[i].setTempDB(tx[0]);
   airMy[i].setHumi(tx[1]);
   airMy[i].setFlowRate( 0 );
   airMy[i].setCO2ppm( this.sysSpace[i].getCO2ppm() );//20131217

   //テスト条件
   //airMy[i].setTempDB( 40-5*i );
   //airMy[i].setHumi(tx[1]);
   //airMy[i].setFlowRate( 0 );
// System.out.print( airMy[i].getDensity()+", " );

}

//swcInの時換気計算する場合で  swcInがoffの時
if( this.isVentOpe_swcIn && Airswc.isOFF( this.swcIn ) ){

   for( int i=0; i<this.airOutFlowRate.length; i++ ){
      this.z_airIn[i].copyAllState( this.airMy[i] );
      this.z_airIn[i].setFlowRate( 0. );

      this.airOutFlowRate[i]=0;
   }

   for( int i=0; i<this.v_airOutIn.length; i++ ){
      this.v_airOutIn[i][0].copyAllState( this.airMy[this.openZoneName1[i]]);
      this.v_airOutIn[i][1].copyAllState( this.airMy[this.openZoneName2[i]]);
      this.v_airOutIn[i][0].setFlowRate( 0.);
      this.v_airOutIn[i][1].setFlowRate( 0.);
   }

   this.message.append( "(C)swcIn換気計算停止" );

   //記録
   if( this.isRecord  && super.rm != null ){
      this.record();
   }
```

```
            return;
        }

        //外部開口を制御する条件の判定
        for( int i=0; i<this.zvSpec.length; i++ ){

            this.isThisCloseNow[i] = false;

            if( this.openZoneName2[i] != 0 ){
                continue;  //外気に面する開口でない場合はパスする
            }

            int numZ1 = this.openZoneName1[i];
            boolean isCloseNow = false;

            //外気に面する開口の開閉制御
            //降雨の時に閉じる
            if( this.isCloseRain && this.watInRain.getFlowRate() > 0 ){
                isCloseNow = true;
            }

            //外部風速が上限値超えた時に閉じる
            if( this.winIn.getVelocity() > this.maxV_OA ){  //------------------代表風速でよいか？？？？？
                isCloseNow = true;
            }
            //外気＞室の時に閉じる
            if( this.compareOutInNum == 0 ){
                //閉じない
            }else if(  this.compareOutInNum == 1 && this.airMy[ numZ1 ].getTempDB() < this.airInOA.getTempDB() ){
                //乾球温度
                isCloseNow = true;
            }else if(  this.compareOutInNum == 2 && this.airMy[ numZ1 ].getEnthalpy() < this.airInOA.getEnthalpy() ){
                //比エンタルピ
                isCloseNow = true;
            }else if(  this.compareOutInNum == 3 && this.airMy[ numZ1 ].getTempDB() < this.airInOA.getTempDB() ||
this.airMy[ numZ1 ].getEnthalpy() < this.airInOA.getEnthalpy() ){
                //乾球温度または比エンタルピー
                isCloseNow = true;
            }else if(  this.compareOutInNum == 4 && this.airMy[ numZ1 ].getTempDB() < this.airInOA.getTempDB() &&
this.airMy[ numZ1 ].getEnthalpy() < this.airInOA.getEnthalpy() ){
                //乾球温度かつ比エンタルピー
                isCloseNow = true;
            }

            if( this.closeLowerValueDB > this.airInOA.getTempDB()
                  || this.closeLowerValueH > this.airInOA.getEnthalpy() ){
                //外気下限値DBより低い
                isCloseNow = true;
            }

            if( isCloseNow ){
                this.openDelaySeconds[i] = this.delaySeconds * 2;//outputs()は  2回実行されるので
                this.isThisCloseNow[i] = true;
            }

            if( this.openDelaySeconds[i] > 0 ){
                this.isThisCloseNow[i] = true;
            }

            this.openDelaySeconds[i] -= BestTimeManager.getCurrentInterval();

        }



        //換気回路の数だけ
        this.isBalance = false;

        double[] matrix = new double[ (this.zvName.length-1) * (this.zvName.length-1) ];
        //double[] vector = new double[ this.zvName.length-1 ];
```

```
// for( int i=0; i<this.zonePReference.length; i++ ){
//    this.zonePReference[i]   = SPa - this.zoneFloorLevel[i] * this.airInOA.getDensity() * g;
// }


   double[] zone_dQ     = new double[ this.zvName.length ];//ゾーンの出入り風量収支[m3/s]
   double[] zone_dQOld  = new double[ this.zvName.length ];//ゾーンの出入り風量収支[m3/s]
   double[] zone_dQm    = new double[ this.zvName.length-1 ];//ゾーンの出入り風量収支[m3/s]
   double[] zone_dM     = new double[ this.zvName.length ];//ゾーンの出入り風量収支[g/s]
   double[] zone_dMOld  = new double[ this.zvName.length ];//ゾーンの出入り風量収支[g/s]
   double[] zone_dMm    = new double[ this.zvName.length-1 ];//ゾーンの出入り風量収支[g/s]

   double[] zone_dQPlus = new double[ this.zvName.length ];//+dPを与えた時のゾーンの出入り風量収支[m3/s]
   double[] zone_dQMinus= new double[ this.zvName.length ];//-dPを与えた時のゾーンの出入り風量収支[m3/s]
   double[] zone_dMPlus = new double[ this.zvName.length ];//+dPを与えた時のゾーンの出入り風量収支[g/s]
   double[] zone_dMMinus= new double[ this.zvName.length ];//-dPを与えた時のゾーンの出入り風量収支[g/s]
   double[] zonePR      = new double[ this.zvName.length ];//ゾーンの床面の圧力[Pa]
   //double dpMax = 1.;//ヤコビアンを作成するための微小圧力[Pa]
   //double dpMin = 0.0001;//ヤコビアンを作成するための微小圧力[Pa]
   boolean[] isB = new boolean[ this.zvName.length ];
   double zone_dQMax = 0;
   //double zone_dQMaxOld1 = 0;
   //double zone_dQMaxOld2 = 0;
   //int num_zone_dQMax = 0;

   double zone_dMMax = 0;
   double zone_dMMaxOld1 = 0;
   double zone_dMMaxOld2 = 0;
   int num_zone_dMMax = 0;

   int count = 0;
   for( int i=0; i<zone_dQ.length; i++ ){
      zone_dQ[i] = 0;
      zone_dQOld[i] = 0;
      zone_dM[i] = 0;
      zone_dMOld[i] = 0;
   }


   while( !this.isBalance ){

      //zone_dQMaxOld2 = zone_dQMaxOld1;
      //zone_dQMaxOld1 = zone_dQMax;
      zone_dQMax = 0;
      zone_dMMaxOld2 = zone_dMMaxOld1;
      zone_dMMaxOld1 = zone_dMMax;
      zone_dMMax = 0;
      this.isBalance = true;
      count++;

      for( int j=0; j<this.zvName.length; j++ ){
         zonePR[j] = this.zonePReference[j];
         zone_dQOld[j] = zone_dQ[j];
         zone_dMOld[j] = zone_dM[j];
      }


      this.calcZone_dQ( zonePR, zone_dQ, zone_dM );

      //System.out.print( "zone_dQ  風量収支[m3/s]=");
      for( int i=1; i<zone_dQ.length; i++ ){//**************
         isB[i] = true;
         if( zone_dQMax < Math.abs( zone_dQ[i] ) ){
            zone_dQMax = Math.abs( zone_dQ[i] );
            //num_zone_dQMax = i;
         }
         if( zone_dMMax < Math.abs( zone_dM[i] ) ){
            zone_dMMax = Math.abs( zone_dM[i] );
            num_zone_dMMax = i;
         }
```

```
            //System.out.print( AirFormat.df_5( zone_dQ[i] ) + ", ");
//////          if( zone_dQ[i] > this.vLimit || zone_dQ[i]< -this.vLimit ){
            if( zone_dM[i] > this.vLimit || zone_dM[i]< -this.vLimit ){
               this.isBalance = false;
               isB[i] = false;
            }
            zone_dQ[i] = - zone_dQ[i];
            zone_dM[i] = - zone_dM[i];
         }
         //System.out.println( "");

         if( this.isBalance ){
            //System.out.println( "isBalance = true " );
            break;
         }

         for( int j=1; j<this.zvName.length; j++ ){//**************
            //System.out.print( "(while) j="+j );

            double jvalue = zonePR[j];
            //System.out.print( "  jvalue="+jvalue );

            zonePR[j] = jvalue + dP;// j 番にdpを加える

            //System.out.println( "zone_dQPlus");
            this.calcZone_dQ( zonePR, zone_dQPlus, zone_dMPlus );

            zonePR[j] = jvalue - dP;// j 番からdpを引く

            //System.out.println( "zone_dQMinus");
            this.calcZone_dQ( zonePR, zone_dQMinus, zone_dMMinus );

            zonePR[j] = jvalue;// j 番を元に戻す

            for( int i=1; i<this.zvName.length; i++ ){
               //結果をmatrixのデータへ

//////            matrix[ (i-1) * (this.zvName.length-1) + j-1 ] = ( zone_dQPlus[i] - zone_dQMinus[i] ) / 2. / dP;
               matrix[ (i-1) * (this.zvName.length-1) + j-1 ] = ( zone_dMPlus[i] - zone_dMMinus[i] ) / 2. / dP;

               this.matrixSum += matrix[ (i-1) * (this.zvName.length-1) + j-1 ];
            }
         }

         //逆行列を求める前のチェック
//////      if( this.matrixSum == 0 || this.matrixSum == this.matrixSumOld || count > 50 || ( zone_dQMax ==
zone_dQMaxOld2 && count > 100 ) ){
         if( this.matrixSum == 0 || this.matrixSum == this.matrixSumOld || count > 500 || ( zone_dMMax == zone_dMMaxOld2
&& count > 100 ) ){
         //if( this.matrixSum == 0 || count > 50 ){


            double rm = Math.random();

            if( count > 50 ){

               boolean isBalanceSub = false;
               int subCount = 0;

               while( !isBalanceSub ){

                  subCount++;
                  zone_dQMax = 0;
                  zone_dMMax = 0;
                  isBalanceSub = true;

                  for( int i=0; i<zone_dQ.length; i++ ){
                     zone_dQOld[i] = zone_dQ[i];
                     zone_dMOld[i] = zone_dM[i];
                  }

                  for( int i=0; i<zonePR.length; i++ ){
```

```
                zonePR[i] = this.zonePReference[i];
            }

            //残風量の計算
            this.calcZone_dQ( zonePR, zone_dQ, zone_dM );

            //System.out.print( "zone_dQ  Sub風量収支[m3/s]=");
            for( int i=1; i<zone_dQ.length; i++ ){//**************
                isB[i] = true;
                if( zone_dQMax < Math.abs( zone_dQ[i] ) ){
                    zone_dQMax = Math.abs( zone_dQ[i] );
                    //num_zone_dQMax = i;
                }
                if( zone_dMMax < Math.abs( zone_dM[i] ) ){
                    zone_dMMax = Math.abs( zone_dM[i] );
                    num_zone_dMMax = i;
                }
/////             System.out.print( zone_dQ[i] + ", ");
                //System.out.print( zone_dM[i] + ", ");
//////                if( zone_dQ[i] > this.vLimit || zone_dQ[i]< -this.vLimit ){
                if( zone_dM[i] > this.vLimit || zone_dM[i]< -this.vLimit ){
                    isBalanceSub = false;
                    isB[i] = false;
                }
            }
            //System.out.println( "");

            if( isBalanceSub ){
                //System.out.println( "isBalance = true " );
                this.maxSubCount = this.startSubCount;
                break;
            }

            //最大残風量ゾーンの圧力の見直し
/////         double d_pref = this.zonePReference[ num_zone_dQMax ] - this.zonePOldReference[ num_zone_dQMax ];
            double d_pref = this.zonePReference[ num_zone_dMMax ] - this.zonePOldReference[ num_zone_dMMax ];
/////         double d_Q    = zone_dQ[ num_zone_dQMax ] - zone_dQOld[ num_zone_dQMax ];
            double d_M    = zone_dM[ num_zone_dMMax ] - zone_dMOld[ num_zone_dMMax ];
/////         System.out.println( "subCount ="+subCount+" num_zone_dQMax="+num_zone_dQMax+" zone_dQMax="+zone_dQMax+"
zonePOldRef="+this.zonePOldReference[ num_zone_dQMax ]+" zonePRef="+this.zonePReference[ num_zone_dQMax ]+" d_pref
="+d_pref + " d_Q="+d_Q );
            //System.out.println( "subCount ="+subCount+" num_zone_dMMax="+num_zone_dMMax+"
zone_dMMax="+zone_dMMax+" zonePOldRef="+this.zonePOldReference[ num_zone_dMMax ]+"
zonePRef="+this.zonePReference[ num_zone_dMMax ]+" d_pref ="+d_pref + " d_M="+d_M );
            for( int i=0; i<zonePR.length; i++ ){
                this.zonePOldReference[i] = this.zonePReference[i];
            }
            double ddp;
/////         if( d_Q != 0 ){
/////             ddp = Math.abs( d_pref / d_Q  * zone_dQ[ num_zone_dQMax ] / 2. );
            if( d_M != 0 ){
                ddp = Math.abs( d_pref / d_M  * zone_dM[ num_zone_dMMax ] / 2. );
            }else{
                ddp = Math.abs( d_pref / 2. );
            }
            if( ddp == 0 ){
                ddp = this.hoseidP * Math.random();
                //break;
            }
/////         if( zone_dQ[ num_zone_dQMax ] > 0 ){
/////             this.zonePReference[ num_zone_dQMax ] += ddp;
/////         }else if( zone_dQ[ num_zone_dQMax ] < 0 ){
/////             this.zonePReference[ num_zone_dQMax ] -= ddp;
/////         }
            if( zone_dM[ num_zone_dMMax ] > 0 ){
                this.zonePReference[ num_zone_dMMax ] += ddp;
            }else if( zone_dM[ num_zone_dMMax ] < 0 ){
                this.zonePReference[ num_zone_dMMax ] -= ddp;
            }

            if( subCount > this.maxSubCount ){
                //System.out.println( "+++++maxSubCount ="+ this.maxSubCount + "--> +1" );
```

```
                this.maxSubCount +=1;
                break;
            }

         } //End of sub while

         System.out.print( "強制補正  count >="+ count );
         count = 0;

         this.matrixSum = 0.999994 + count;
         System.out.println( "" );

      }else

      if( this.matrixSum == 0 ){
         System.out.print( "強制補正  matrixSum== 0 count="+ count + " rm="+rm);
         if( rm > 0.25 ){
            for( int i=1; i<this.zonePReference.length; i++ ){
               this.zonePOldReference[i] = this.zonePReference[i];
               //zone_dQ の値の正負は逆である←→これに合わせて強制補正
/////            if( zone_dQ[i] > 0 && !isB[i] ){
               if( zone_dM[i] > 0 && !isB[i] ){
/////               this.zonePReference[i] -= this.hoseidP * Math.random() * zone_dQ[i] / zone_dQMax;
                  this.zonePReference[i] -= this.hoseidP * Math.random() * zone_dM[i] / zone_dMMax;
                  //System.out.print( "-"+i);
/////            }else if( zone_dQ[i] < 0 && !isB[i]  ){
               }else if( zone_dM[i] < 0 && !isB[i]  ){
/////               this.zonePReference[i] -= this.hoseidP * Math.random() * zone_dQ[i] / zone_dQMax;
                  this.zonePReference[i] -= this.hoseidP * Math.random() * zone_dM[i] / zone_dMMax;
                  //System.out.print( "+"+i);
               }else{
               }
            }
         }else if( rm > 0.05 ){
            for( int i=1; i<this.zonePReference.length; i++ ){
               this.zonePOldReference[i] = this.zonePReference[i];
               //zone_dQ の値の正負は逆である←→これに合わせて強制補正
/////            if( zone_dQ[i] > 0 && !isB[i]  ){
               if( zone_dM[i] > 0 && !isB[i]  ){
                  this.zonePReference[i] -= this.hoseidP * Math.random();
                  //System.out.print( "-"+i);
/////            }else if( zone_dQ[i] < 0 && !isB[i]  ){
               }else if( zone_dM[i] < 0 && !isB[i]  ){
                  this.zonePReference[i] += this.hoseidP * Math.random();
                  //System.out.print( "+"+i);
               }else{
               }
            }
         }else{
            for( int i=1; i<this.zonePReference.length; i++ ){
               this.zonePOldReference[i] = this.zonePReference[i];
               //zone_dQ の値の正負は逆である←→これに合わせて強制補正
/////            if( zone_dQ[i] > 0 && !isB[i]  ){
               if( zone_dM[i] > 0 && !isB[i]  ){
                  this.zonePReference[i] = this.zonePReference[i] * 0.8 + SPa * 0.2;
                  //System.out.print( "-"+i);
/////            }else if( zone_dQ[i] < 0 && !isB[i]  ){
               }else if( zone_dM[i] < 0 && !isB[i]  ){
                  this.zonePReference[i] = this.zonePReference[i] * 0.8 + SPa * 0.2;
                  //System.out.print( "+"+i);
               }else{
               }
            }
         }
         System.out.println( "" );

         this.matrixSum = 0.999991;
         //count = 0;
      }else

/////   if( zone_dQMax == zone_dQMaxOld2 && count > 100 ){
      if( zone_dMMax == zone_dMMaxOld2 && count > 100 ){
```

```
/////          System.out.print( "強制補正  zone_dQMax == zone_dQMaxOld2 count="+ count );
               System.out.print( "強制補正  zone_dMMax == zone_dMMaxOld2 count="+ count );
               for( int i=1; i<this.zonePReference.length; i++ ){
                  this.zonePOldReference[i] = this.zonePReference[i];
                  //zone_dQ の値の正負は逆である←→これに合わせて強制補正
/////               if( zone_dQ[i] > 0 && !isB[i]  ){
                  if( zone_dM[i] > 0 && !isB[i]  ){
                     this.zonePReference[i] = this.zonePReference[i] * rm + SPa * ( 1 - rm );
                     //System.out.print( "-"+i);
/////               }else if( zone_dQ[i] < 0 && !isB[i]  ){
                  }else if( zone_dM[i] < 0 && !isB[i]  ){
                     this.zonePReference[i] = this.zonePReference[i] * rm + SPa * ( 1 - rm );
                     //System.out.print( "+"+i);
                  }else{
                  }
               }
               System.out.println( "" );

               this.matrixSum = 0.999992;
               count = 0;

            }else

            if( this.matrixSum == this.matrixSumOld ){
               System.out.print( "強制補正  matrixSum==matrixSumOld count="+ count );
               if( rm > 0.01 ){
                  for( int i=1; i<this.zonePReference.length; i++ ){
                     this.zonePOldReference[i] = this.zonePReference[i];
                     //zone_dQ の値の正負は逆である←→これに合わせて強制補正
/////                  if( zone_dQ[i] > 0 && !isB[i]  ){
                     if( zone_dM[i] > 0 && !isB[i]  ){
/////                     this.zonePReference[i] -= this.hoseidP * Math.random() * zone_dQ[i] / zone_dQMax;
                        this.zonePReference[i] -= this.hoseidP * Math.random() * zone_dM[i] / zone_dMMax;
                        //System.out.print( "-"+i);
/////                  }else if( zone_dQ[i] < 0 && !isB[i]  ){
                     }else if( zone_dM[i] < 0 && !isB[i]  ){
/////                     this.zonePReference[i] -= this.hoseidP * Math.random() * zone_dQ[i] / zone_dQMax;
                        this.zonePReference[i] -= this.hoseidP * Math.random() * zone_dM[i] / zone_dMMax;
                        //System.out.print( "+"+i);
                     }else{
                     }
                  }
               }else if( rm > 0.15 ){
                  for( int i=1; i<this.zonePReference.length; i++ ){
                     this.zonePOldReference[i] = this.zonePReference[i];
                     //zone_dQ の値の正負は逆である←→これに合わせて強制補正
/////                  if( zone_dQ[i] > 0 && !isB[i]  ){
                     if( zone_dM[i] > 0 && !isB[i]  ){
/////                     this.zonePReference[i] -= this.hoseidP * zone_dQ[i] / zone_dQMax;
                        this.zonePReference[i] -= this.hoseidP * zone_dM[i] / zone_dMMax;
                        //System.out.print( "-"+i);
/////                  }else if( zone_dQ[i] < 0 && !isB[i]  ){
                     }else if( zone_dM[i] < 0 && !isB[i]  ){
/////                     this.zonePReference[i] -= this.hoseidP * zone_dQ[i] / zone_dQMax;
                        this.zonePReference[i] -= this.hoseidP * zone_dM[i] / zone_dMMax;
                        //System.out.print( "+"+i);
                     }else{
                     }
                  }
               }else{
                  for( int i=1; i<this.zonePReference.length; i++ ){
                     this.zonePOldReference[i] = this.zonePReference[i];
                     //zone_dQ の値の正負は逆である←→これに合わせて強制補正
/////                  if( zone_dQ[i] > 0 && !isB[i]  ){
                     if( zone_dM[i] > 0 && !isB[i]  ){
                        this.zonePReference[i] = this.zonePReference[i] * 0.8 + SPa * 0.2;
                        //System.out.print( "-"+i);
/////                  }else if( zone_dQ[i] < 0 && !isB[i]  ){
                     }else if( zone_dM[i] < 0 && !isB[i]  ){
                        this.zonePReference[i] = this.zonePReference[i] * 0.8 + SPa * 0.2;
                        //System.out.print( "+"+i);
                     }else{
```

```
                    }
                }
            }
            System.out.println( "" );

            this.matrixSum = 0.999993;
            count = 0;
         }else

         //System.out.println( "" );

         if( count == 150 ){
            count =0;
         }

      }else{

         for( int i=0; i<zone_dQm.length; i++ ){
            zone_dQm[i] = zone_dQ[i+1];
            zone_dMm[i] = zone_dM[i+1];
         }
//////       this.matrixSum = calcMatrixZonesVent( matrixSum, matrix, zone_dQm );
         this.matrixSum = calcMatrixZonesVent( matrixSum, matrix, zone_dMm );
      }

      this.matrixSumOld = this.matrixSum;
      this.matrixSum = 0;


   }//wile end



   //記録
   if( this.isRecord  && super.rm != null ){
      this.record();
   }

   for( int i=0; i<this.airOutFlowRate.length; i++ ){
      this.airOutFlowRate[i]=0;
   }
}


private double calcMatrixZonesVent( double matrixSum, double[] matrix, double[] zone_dQm ){
   double flag = matrixSum;

   try{
      GVector gvdP = this.calcMatrixVector( this.zvName.length-1, this.zvName.length-1, matrix, zone_dQm );

      //System.out.print( "gvdP size="+gvdP.getSize()+" gvdP 次の補正圧力=" + gvdP );

      double max = dP;
      double hosei = 0;

      for( int i=1; i<this.zonePReference.length; i++ ){
         this.zonePOldReference[i] = this.zonePReference[i];

         hosei = gvdP.getElement( i-1 );
         this.zonePReference[i] += hosei / 2.;

         if( max < Math.abs( gvdP.getElement( i-1 ) ) ){
            max = Math.abs( gvdP.getElement( i-1 ) );
         }
      }
      //System.out.println( "  max dp="+max);

   }
   catch( Exception e ){
      //System.out.println( "(E)圧力の補正過程でエラー発生" );

      this.matrixSum = this.matrixSumOld;
```

```
            double rm = Math.random();

            //System.out.print( "zonePReference[] =" );
            for( int i=1; i<this.zonePReference.length; i++ ){
               this.zonePOldReference[i] = this.zonePReference[i];
               this.zonePReference[i] = this.zonePReference[i] * rm + SPa * ( 1 - rm );
               //System.out.print( this.zonePReference[i] + ", ");
            }
            //System.out.println( "");

            flag = 0.999990;
         }

         return flag;
      }

      /**
       * 連立一次方程式を解く
       * @param row
       * @param col
       * @param matrix
       * @param dQ
       * @return
       */
      private GVector calcMatrixVector( int row, int col, double[] matrix, double[] dQ ){

         //System.out.print( "(calcMatrixVector) raw="+row+ " col="+col );
         //System.out.print( "(calcMatrixVector) matrix[] = ");
         for( int i=0; i<matrix.length; i++ ){
            //System.out.print( matrix[i] + ", " );
         }
         //System.out.println( " ");
         //ヤコビアン A  の定義
         GMatrix gmatrix = new GMatrix( row, col, matrix );

         //LU分解用の領域の確保
         GMatrix lu = new GMatrix( row, col );
         GVector permutation = new GVector( row );

         //ｂベクトルの定義
         GVector b = new GVector( dQ );
         //ｘベクトル（解）の領域
         GVector x = new GVector( col );

         //Ax=bを解く　まずAのLU分解を求める
         int lud = gmatrix.LUD( lu, permutation );
         //System.out.println( "*****************************LUD="+lud);
         //ｘを求める
         x.LUDBackSolve(lu, b, permutation);

//       gmatrix.invert();
//       x.mul( gmatrix, b );
         //
         return x;
      }


      /**
       * @param zonePR ゾーンの床面の圧力[Pa]
       * @param zone_dQ ゾーンの風量収支（入ってくる風量が正）[m3/s]
       * @param zone_dM ゾーンの風量収支（入ってくる風量が正）[g/s]
       */
      private void calcZone_dQ( double[] zonePR, double[] zone_dQ, double[] zone_dM ){

         //System.out.print( "(calcZone_dQ)" );
         //
         for( int i=0; i<this.airOutFlowRate.length; i++ ){
            this.airOutFlowRate[i]=0;
         }

         double[] tf_sum = new double[ this.z_airIn.length ];//温度×流量
         double[] xf_sum = new double[ this.z_airIn.length ];//絶対湿度×流量
```

```
        double[] fr_sum = new double[ this.z_airIn.length ];//流量
        double[] co2sum = new double[ this.z_airIn.length ];//CO2ppm×流量

        //換気経路の数だけ
        for( int i=0; i<this.zvSpec.length; i++ ){

            int numZone1 = this.openZoneName1[i];
            int numZone2 = this.openZoneName2[i];

            double openArea = this.openArea[i];
            if( this.isAvailableDAS[i] ){
               openArea *= this.das[i].getSchedule();
            }

            //System.out.println( "getDensity1" );
            double d1 = this.airMy[ numZone1 ].getDensity();//[kg/m3]
            //System.out.println( "getDensity2" );
            double d2 = this.airMy[ numZone2 ].getDensity();//[kg/m3]
            //System.out.println( "getDensity2 [i]="+ numZone2 +" airMy[numZone2].humi=" + this.airMy[numZone2].getHumi() +
" DB=" + this.airMy[numZone2].getTempDB() + " FlowRate=" + this.airMy[numZone2].getFlowRate() + " airMy[numZone2].humi="+
this.airMy[numZone2].getHumi() );
            //double d2 = this.airMy[ numZone2 ].getDensity();//[kg/m3]
            //double ohL1 = this.openHLabovezoneReference[i] + this.zoneFloorLevel[ numZone1 ];//Zone1床からの開口下端の高さ
[m]
            //double ohH1 = this.openHeight[i] + ohL1;//Zone1床からの開口上端の高さ[m]
            //double ohL2 = ohL1 + this.zoneFloorLevel[ numZone1 ] - this.zoneFloorLevel[ numZone2 ];//Zone2床からの開口下端
の高さ[m]
            //double ohH2 = this.openHeight[i] + ohL2;//Zone2床からの開口上端の高さ[m]
            double ohL1 = this.openHLabovezoneReference[i];//Zone1床からの開口下端の高さ[m]
            double ohH1 = this.openHHabovezoneReference[i];//Zone1床からの開口上端の高さ[m]
            double ohL2 = ohL1 + this.zoneFloorLevel[ numZone1 ] - this.zoneFloorLevel[ numZone2 ];//Zone2床からの開口下端の
高さ[m]
            double ohH2 = ohH1 + this.zoneFloorLevel[ numZone1 ] - this.zoneFloorLevel[ numZone2 ];//Zone2床からの開口上端の
高さ[m]

            double ps1L = zonePR[ numZone1 ] - d1 * g * ohL1;//開口下端Zone1側の圧力[Pa]
            double ps1H = zonePR[ numZone1 ] - d1 * g * ohH1;//開口上端Zone1側の圧力[Pa]
            double ps2L = zonePR[ numZone2 ] - d2 * g * ohL2;//開口下端Zone2側の圧力[Pa]
            double ps2H = zonePR[ numZone2 ] - d2 * g * ohH2;//開口上端Zone2側の圧力[Pa]

            if( numZone1 == 0 ){
               System.out.println( "(E)外気は換気回路の２番目のゾーンで入力すること" );
            }
            if( numZone2 == 0 ){
               //外気の時
               //double windP = this.calc_dPWind( this.openDir[i], ( ohL2 + ohH2 ) /2., this.measureHeight_V_win,
this.isChange_V_winheight, this.winIn );
               double windP = this.calc_dPWind( this.openDir[i], ( ohL2 + ohH2 ) /2. + this.zoneFloorLevel[ numZone2 ],
this.measureHeight_V_win, this.isChange_V_winheight, this.winIn );
               //System.out.println( "windP[Pa] ="+windP);
               ps2L += windP;
               ps2H += windP;

               if( this.isThisCloseNow[i] ){
                  openArea = 0;
               }

            }
            //System.out.println( " this.zonePR["+i+"] = "+zonePR[i]+" [Pa]");

            double dpsL = ps1L - ps2L;//開口下端のZone1基準の圧力差[Pa]
            double dpsH = ps1H - ps2H;//開口上端のZone1基準の圧力差[Pa]
            this.opendPsL[i] = dpsL;//開口下端のZone1基準の圧力差[Pa]
            this.opendPsH[i] = dpsH;//開口上端のZone1基準の圧力差[Pa]

            double m3s = 0;//移動風量[m3/s]

            //System.out.println( "i="+ i +" dpsL=" + dpsL + " dpsH=" + dpsH );

            double keisya = 0;//圧力差分布の傾き
            if( this.openHeight[i] != 0 ){
               keisya = ( dpsH - dpsL ) / this.openHeight[i];
```

```
            //System.out.println( "keisya="+keisya+"  openheight["+i+"]="+openHeight[i] );
        }


        //System.out.println( "keisya = "+ keisya + " dpsL="+dpsL+" dpsH="+dpsH + " openHeight="+openHeight[i] + "
openHLabovezoneReference="+openHLabovezoneReference[i] + " d1="+d1+" d2="+d2);

        if( dpsL >= 0 && dpsH >= 0 ){
            //this.message.append( "()calc_1" );
            //z numZone1→z numZone2  へ流れる
            //System.out.print( "(calcZone_dQ)case①経路["+i+"]" );
            if( keisya != 0 ){
                m3s = this.openCoef[i] * openArea * Math.pow( 2. / d1, 0.5 ) * 2. / 3. / keisya;
                //System.out.print( " m3s_1=" + m3s );
                m3s *= ( Math.pow( dpsH, 1.5 ) - Math.pow( dpsL, 1.5 ) ) / this.openHeight[i];//20160830
                //System.out.print( " m3s_2=" + m3s );
            }else{
                m3s = this.openCoef[i] * openArea * Math.pow( dpsL * 2. / d1, 0.5 );
                //System.out.println( " m3s_3=" + m3s );
            }
            //this.message.append( "()m3s="+m3s+"()keisya="+keisya+"()dpsL="+dpsL+"()dpsH="+dpsH );
            //System.out.println( " m3s_4=" + m3s );
            this.v_airOutIn[i][0].setTempDB( this.airMy[ numZone1 ].getTempDB() );
            this.v_airOutIn[i][0].setHumi( this.airMy[ numZone1 ].getHumi() );
            this.v_airOutIn[i][0].setFlowRate( m3s * d1 * 1000. );
            this.v_airOutIn[i][0].setCO2ppm( this.airMy[ numZone1 ].getCO2ppm() );//20131217

            this.v_airOutIn[i][1].setTempDB( this.airMy[ numZone2 ].getTempDB() );
            this.v_airOutIn[i][1].setHumi( this.airMy[ numZone2 ].getHumi() );
            this.v_airOutIn[i][1].setFlowRate( 0 );
            this.v_airOutIn[i][1].setCO2ppm( this.airMy[ numZone2 ].getCO2ppm() );//20131217

        }else if( dpsL <= 0 && dpsH <= 0 ){
            //this.message.append( "()calc_2" );
            //z numZone1←z numZone2  へ流れる
            //System.out.print( "(calcZone_dQ)case②経路["+i+"]" );
            if( keisya != 0 ){
                m3s = this.openCoef[i] * openArea * Math.pow( 2. / d2, 0.5 ) * 2. / 3. / Math.abs( keisya );
                //System.out.print( " m3s_1=" + m3s );
                m3s *= Math.abs( Math.pow( -dpsH, 1.5) - Math.pow( -dpsL, 1.5 ) ) / this.openHeight[i];//20160830
                //System.out.print( " m3s_2=" + m3s );
            }else{
                m3s = this.openCoef[i] * openArea * Math.pow( -dpsL * 2. / d2, 0.5 );
                //System.out.print( " m3s_3=" + m3s );
            }
            //System.out.println( " m3s_4=" + m3s );

            this.v_airOutIn[i][0].setTempDB( this.airMy[ numZone1 ].getTempDB() );
            this.v_airOutIn[i][0].setHumi( this.airMy[ numZone1 ].getHumi() );
            this.v_airOutIn[i][0].setFlowRate( 0 );
            this.v_airOutIn[i][0].setCO2ppm( this.airMy[ numZone1 ].getCO2ppm() );//20131217

            this.v_airOutIn[i][1].setTempDB( this.airMy[ numZone2 ].getTempDB() );
            this.v_airOutIn[i][1].setHumi( this.airMy[ numZone2 ].getHumi() );
            this.v_airOutIn[i][1].setFlowRate( m3s * d2 * 1000. );
            this.v_airOutIn[i][1].setCO2ppm( this.airMy[ numZone2 ].getCO2ppm() );//20131217

        }else if( dpsL >= 0 && dpsH < 0 ){
            //this.message.append( "()calc_3" );
            //開口の下部では  //z numZone1→z numZone2  へ流れる
            //      上部では  //z numZone1←z numZone2  へ流れる
            //System.out.print( "(calcZone_dQ)case③経路["+i+"]" );
            if( keisya != 0 ){
                m3s = -this.openCoef[i] * openArea * Math.pow( 2. / d1, 0.5 ) * 2. / 3. / keisya;//正に補正
                //System.out.print( " m3s_1=" + m3s );
                m3s *= Math.pow( dpsL, 1.5 ) / ( this.openHeight[i] * dpsL / ( dpsL - dpsH ) );//20160830
                //System.out.print( " m3s_2=" + m3s );
            }else{
                m3s = 0;
                //System.out.print( " m3s_3=" + m3s );
            }
            m3s *= dpsL / ( dpsL - dpsH );//開口の下部の割合
```

```
        //System.out.print( " m3s_4=" + m3s );
        this.v_airOutIn[i][0].setTempDB( this.airMy[ numZone1 ].getTempDB() );
        this.v_airOutIn[i][0].setHumi( this.airMy[ numZone1 ].getHumi() );
        this.v_airOutIn[i][0].setFlowRate( m3s * d1 * 1000. );//[m3/s][kg/m3][g/kg]=[g/s]
        this.v_airOutIn[i][0].setCO2ppm( this.airMy[ numZone1 ].getCO2ppm() );//20131217
        //上部
        if( keisya != 0 ){
           m3s = -this.openCoef[i] * openArea * Math.pow( 2. / d2, 0.5 ) * 2. / 3. / keisya;//正に補正
           //System.out.print( " m3s_5=" + m3s );
           m3s *= Math.pow( -dpsH, 1.5 ) / ( this.openHeight[i] * (-dpsH / ( dpsL - dpsH )) );//20160830
           //System.out.print( " m3s_6=" + m3s );
        }else{
           m3s = 0;
           //System.out.print( " m3s_7=" + m3s );
        }
        m3s *= ( -dpsH / ( dpsL - dpsH ));//開口の上部の割合
        //System.out.println( " m3s_8=" + m3s );
        this.v_airOutIn[i][1].setTempDB( this.airMy[ numZone2 ].getTempDB() );
        this.v_airOutIn[i][1].setHumi( this.airMy[ numZone2 ].getHumi() );
        this.v_airOutIn[i][1].setFlowRate( m3s * d2 * 1000. );
        this.v_airOutIn[i][1].setCO2ppm( this.airMy[ numZone2 ].getCO2ppm() );//20131217

     }else if( dpsL <= 0 && dpsH > 0 ){
        //this.message.append( "()calc_4" );

        //開口の下部では  //z numZone1←z numZone2  へ流れる
        //      上部では  //z numZone1→z numZone2  へ流れる
        //System.out.print( "(calcZone_dQ)case④経路["+i+"]" );
        if( keisya != 0 ){
           m3s = this.openCoef[i] * openArea * Math.pow( 2. / d2, 0.5 ) * 2. / 3. / keisya;
           //System.out.print( " m3s_1=" + m3s );
           m3s *= Math.pow( -dpsL, 1.5 ) / ( this.openHeight[i] * ( -dpsL / ( dpsH - dpsL )) );//20160830
           //System.out.print( " m3s_2=" + m3s );
        }else{
           m3s = 0;
           //System.out.print( " m3s_3=" + m3s );
        }
        m3s *= ( -dpsL / ( dpsH - dpsL ));//開口の下部の割合
        //System.out.print( " m3s_4=" + m3s );
        this.v_airOutIn[i][1].setTempDB( this.airMy[ numZone2 ].getTempDB() );
        this.v_airOutIn[i][1].setHumi( this.airMy[ numZone2 ].getHumi() );
        this.v_airOutIn[i][1].setFlowRate( m3s * d2 * 1000. );
        this.v_airOutIn[i][1].setCO2ppm( this.airMy[ numZone2 ].getCO2ppm() );//20131217
        //上部
        if( keisya != 0 ){
           m3s = this.openCoef[i] * openArea * Math.pow( 2 / d1, 0.5 ) * 2. / 3. / keisya;
           //System.out.print( " m3s_5=" + m3s );
           m3s *= Math.pow( dpsH, 1.5 ) / ( this.openHeight[i] * ( dpsH / ( dpsH - dpsL )) );//20160830
           //System.out.print( " m3s_6=" + m3s );
        }else{
           m3s = 0;
           //System.out.print( " m3s_7=" + m3s );
        }
        m3s *= ( dpsH / ( dpsH - dpsL ));//開口の上部の割合
        //System.out.println( " m3s_8=" + m3s );
        this.v_airOutIn[i][0].setTempDB( this.airMy[ numZone1 ].getTempDB() );
        this.v_airOutIn[i][0].setHumi( this.airMy[ numZone1 ].getHumi() );
        this.v_airOutIn[i][0].setFlowRate( m3s * d1 * 1000. );
        this.v_airOutIn[i][0].setCO2ppm( this.airMy[ numZone1 ].getCO2ppm() );//20131217

     }else{
        //this.message.append( "()calc_5******" );
     }
  // System.out.println( "m3s = "+ m3s );

     fr_sum[ numZone1 ] += this.v_airOutIn[i][1].getFlowRate();
     tf_sum[ numZone1 ] +=(this.v_airOutIn[i][1].getTempDB() * this.v_airOutIn[i][1].getFlowRate());
     xf_sum[ numZone1 ] +=(this.v_airOutIn[i][1].getHumi()    * this.v_airOutIn[i][1].getFlowRate());
     co2sum[ numZone1 ] +=(this.v_airOutIn[i][1].getCO2ppm() * this.v_airOutIn[i][1].getFlowRate());

     fr_sum[ numZone2 ] += this.v_airOutIn[i][0].getFlowRate();
     tf_sum[ numZone2 ] +=(this.v_airOutIn[i][0].getTempDB() * this.v_airOutIn[i][0].getFlowRate());
```

```
            xf_sum[ numZone2 ] +=(this.v_airOutIn[i][0].getHumi()    * this.v_airOutIn[i][0].getFlowRate());
            co2sum[ numZone2 ] +=(this.v_airOutIn[i][0].getCO2ppm() * this.v_airOutIn[i][0].getFlowRate());

            this.airOutFlowRate[ numZone1 ] += this.v_airOutIn[i][0].getFlowRate();
            this.airOutFlowRate[ numZone2 ] += this.v_airOutIn[i][1].getFlowRate();


            //System.out.println( "(airOutIn) O1="+v_airOutIn[0][1].getFlowRate() +"   OO="+
v_airOutIn[0][0].getFlowRate() );
            //System.out.println( "(airOutFR) O1="+this.airOutFlowRate[1] +"   OO="+ this.airOutFlowRate[0] );

        }


        for( int i=0; i<this.z_airIn.length; i++ ){
            if( xf_sum[i] < 0 ){
                //System.out.println( "(E) xf ==> =0; ["+xf_sum[i]+"]");
                xf_sum[i] = 0;
            }

            if( fr_sum[i] < 0 ){
                //System.out.println( "fr_swm[]="+AirFormat.df_2( fr_sum[i])+"<0 i="+i );
            }
            this.z_airIn[i].setFlowRate( fr_sum[i] );
            if( fr_sum[i]==0 ){
                this.z_airIn[i].setTempDB( 0 );//20131217
                this.z_airIn[i].setHumi( 0 );//20131217
                this.z_airIn[i].setCO2ppm( 0 );//20131217
            }else{
                this.z_airIn[i].setTempDB( tf_sum[i] / fr_sum[i] );
                this.z_airIn[i].setHumi( xf_sum[i] / fr_sum[i] );
                this.z_airIn[i].setCO2ppm( co2sum[i] / fr_sum[i] );
            }
        }

        //System.out.print( "zone_dQ = ");
        //System.out.print( "zone_dM = ");
        for( int i=0; i<this.z_airIn.length; i++ ){

            //if( i==0 ){
            //System.out.println( "getDensity3 [i]="+i+" airIn[i].humi=" + this.z_airIn[i].getHumi() + " DB=" +
this.z_airIn[i].getTempDB() + " FlowRate=" + this.z_airIn[i].getFlowRate() + " airMy[i].humi="+
this.airMy[i].getHumi() );
            //}
            zone_dQ[i] =  ( this.z_airIn[i].getFlowRate() / this.z_airIn[i].getDensity() - this.airOutFlowRate[i] /
this.airMy[i].getDensity() ) / 1000.;
            zone_dM[i] = this.z_airIn[i].getFlowRate() - this.airOutFlowRate[i];
            //System.out.print( zone_dQ[i] + ", ");
            //System.out.print( zone_dM[i] +"( Humi="+this.z_airIn[i].getHumi()+", fr_sum="+fr_sum[i]+"), ");
        }
        //System.out.println( "");
        //
        //System.exit(0);

    }

    /**
     * @param openDir 開口の方位[-]
     * @param openLevel 開口の中心高さ[m]
     * @param hs 風の観測高さ[m]
     * @param isChange_V_H 風速を高さで変える=true[boolean]
     * @param winIn 風情報[BestWind]
     * @return 風による圧力[Pa]
     */
    private double calc_dPWind( double openDir, double openLevel, double hs, boolean isChange_V_H, BestWind winIn ){
        double dPWind = 0;

        //以下　建築Infiltationクラスの getDeltaPWind メソッドを参考とした

        double deltaW;    //壁面に対する風向(正面から吹く場合は0,真後から吹く場合は180)
        double cw;        //風圧係数      [-]
        //double hs = 15.0;  //風速測定高       [m] 不明の場合15.0[m]とのこと
```

```
    double windDir = winIn.getDir();  //風向[-](北北東:1,北:16)
    //System.out.print( "windDir="+windDir);

    double windVel = winIn.getVelocity();  //風速[m/s]
    if( windVel==0. || windDir==0. ){
        return 0.;
    }

    double wallDir = openDir + 180.0;       //方位角を気象データに合わせる
    if( wallDir > 360 ){
        wallDir -= 360.0;
    }

    //壁面に対する風向の計算
    deltaW = wallDir - windDir * 22.5;
    //角度の絶対値をとる
    if( deltaW < 0. ){
        deltaW *= -1.;
    }
    //角度を0～180の値にする
    if( deltaW > 180. ){
        deltaW = 360.0 - deltaW;
    }

    //風圧係数の計算(空気調和衛生工学会による)
    if( deltaW <= 30.0 ){
        cw = 0.75;
    }else if( deltaW <= 75.0 ){
        cw = 0.75 - 0.0166 * ( deltaW - 30.0 );
    }else if( deltaW <= 90.0 ){
        cw = -0.0267 * ( deltaW - 75.0 );
    }else{
        cw = -0.4;
    }

    if( isChange_V_H ){
        //室の地上高における風速補正係数の計算
        double vRatio=1.;
        //double hs = 15.0;  //風速測定高[m] 不明の場合15.0[m]とのこと
        double hRatio = openLevel / hs;//測定高と室高さ比
        if( openLevel > hs ){
            vRatio = Math.pow( hRatio, 0.25 );//測定高と室高さ比を1/4乗
        }
        windVel = windVel * vRatio;//室高さでの風速
    }else{

    }

    //風力による圧力の計算
    dPWind = cw * winIn.getDensity() * Math.pow( windVel, 2 ) / 2.;       //外部風分

    //System.out.println( "calc_windP() dpWind="+dPWind+"[Pa]" );
    return dPWind;
}


/**
 * 記録
 */
private void record(){

    if( CheckPrintModule.isPrintMessage ){
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                this.RECORD_message, this.name0, this.message.toString());
    }

    if( CheckPrintModule.isPrintStateOut ){
        if( this.isZoneRecord ){
        for( int i=1; i<this.zvName.length; i++ ){
            String num0 = this.get0num_3(i);
```

```
            // super.rm.setRecord(super.getConnectionNode(this.R_NODE),
        //    i+this.RECORD_T_aOut, this.name[i], airOut[i][i].getTempDB());
        // super.rm.setRecord(super.getConnectionNode(this.R_NODE),
        //    i+this.RECORD_X_aOut, this.name[i], airOut[i][i].getHumi());
        // super.rm.setRecord(super.getConnectionNode(this.R_NODE),
        //    i+this.RECORD_D_aOut, this.name[i], airOut[i][i].getDensity());
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_M_aOut+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_3( this.airOutFlowRate[i]));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_V_aOut+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_3( this.airOutFlowRate[i]/this.airMy[i].getDensity() * 3.6 ));
        }
        }
    }

    if( CheckPrintModule.isPrintStateMy ){
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
              this.RECORD_Dir_win, this.name0+"zav外気", this.winIn.getDir());
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
              this.RECORD_Vel_win, this.name0+"zav外気", this.winIn.getVelocity());
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
              this.RECORD_NV_all, this.name0+"zav外気",
              ( this.zoneVolumeSum==0 ? 0 :
              AirFormat.df_3( this.airOutFlowRate[0] / this.airInOA.getDensity() * 3.6 / this.zoneVolumeSum )));
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
              this.RECORD_MMrain, this.name0+"zav外雨", this.watInRain.getFlowRate() );

        if( this.isZoneRecord ){
        for( int i=0; i<this.airMy.length; i++ ){
            String num0 = this.getOnum_3(i);

            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_T_aMy+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_3( this.airMy[i].getTempDB()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_X_aMy+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_5( this.airMy[i].getHumi()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_CO2aMy+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_3( this.airMy[i].getCO2ppm()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_D_aMy+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_5( this.airMy[i].getDensity()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_PRef+"_"+this.name[i], this.name0+"zav節点"+num0, this.zonePReference[i] );
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_NV_aIn+"_"+this.name[i], this.name0+"zav節点"+num0,
               ( this.zoneVolume[i]==0 ? 0 :
               AirFormat.df_3( this.z_airIn[i].getFlowRate()/this.z_airIn[i].getDensity() * 3.6 /
this.zoneVolume[i] ) ));
        }
        }
        if( this.isVentRecord ){
        for( int i=0; i<this.v_airOutIn.length; i++ ){
            String num0 = this.getOnum_3(i);

            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  " → DB#℃#温度"+this.zvSpec[i], this.name0+"zav回路"+num0,
                  AirFormat.df_3( this.v_airOutIn[i][0].getTempDB()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  " → 流量#g/s#流量"+this.zvSpec[i], this.name0+"zav回路"+num0,
                  AirFormat.df_3( this.v_airOutIn[i][0].getFlowRate()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  " → 風量#m3/h#風量"+this.zvSpec[i], this.name0+"zav回路"+num0,
                  AirFormat.df_3( this.v_airOutIn[i][0].getFlowRate()/this.v_airOutIn[i][0].getDensity() * 3.6 ));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  " ← DB#℃#温度"+this.zvSpec[i], this.name0+"zav回路"+num0,
                  AirFormat.df_3( this.v_airOutIn[i][1].getTempDB()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  " ← 流量#g/s#流量"+this.zvSpec[i], this.name0+"zav回路"+num0,
                  AirFormat.df_3( this.v_airOutIn[i][1].getFlowRate()));
```

```
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  " ← 風量#m3/h#風量"+this.zvSpec[i], this.name0+"zav回路"+num0,
                  AirFormat.df_3( this.v_airOutIn[i][1].getFlowRate()/this.v_airOutIn[i][1].getDensity() * 3.6 ));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  " ― 下端圧力差#Pa#圧力"+this.zvSpec[i], this.name0+"zav回路"+num0,
                  AirFormat.df_3( this.opendPsL[i] ));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  " ― 上端圧力差#Pa#圧力"+this.zvSpec[i], this.name0+"zav回路"+num0,
                  AirFormat.df_3( this.opendPsH[i] ));
         }
         }
      }

      if( CheckPrintModule.isPrintStateIn ){
         if( this.isZoneRecord ){
         for( int i=0; i<this.zvName.length; i++ ){
            String num0 = this.getOnum_3(i);

            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_T_aIn+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_3( this.z_airIn[i].getTempDB()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_X_aIn+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_5( this.z_airIn[i].getHumi()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_CO2aIn+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_3( this.z_airIn[i].getCO2ppm()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_D_aIn+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_5( this.z_airIn[i].getDensity()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_M_aIn+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_3( this.z_airIn[i].getFlowRate()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_V_aIn+"_"+this.name[i], this.name0+"zav節点"+num0,
               AirFormat.df_3( this.z_airIn[i].getFlowRate()/this.z_airIn[i].getDensity() * 3.6 ));
         }
         }
      }

      this.message.setLength( 0 );
   }


   //20131217
   public void update(){
      //if( super.sm == null || super.cm == null) return;
      for( int ico = 1; ico< this.sysSpace.length; ico++ ){

         this.zoneAirSysCalc.countUp();
      }
   }

   //20131217
/* public void update() {
      if( super.sm == null || super.cm == null) return;

      double sec = BestTimeManager.getCurrentInterval();

      for( int ico = 1; ico< this.coreSpace.length; ico++ ){
         //if( this.z_airIn[ico].getFlowRate() == 0.) return;
         if( this.z_airIn[ico].getFlowRate() <= 0.) continue;

         //空調時のみ建物側へ計算値を受け渡す

         //室熱平衡式係数計算
         double[] coeffT = new double[4];
         double[] coeffX = new double[3];
         double cpg = CP * this.z_airIn[ico].getFlowRate();
         coeffT[0] =-cpg;
         coeffT[2] = cpg * this.z_airIn[ico].getTempDB();
         coeffT[3] = this.z_airIn[ico].getFlowRate();
```

```
        double rg = R * this.z_airIn[ico].getFlowRate();
        coeffX[0] =-rg;
        coeffX[2] = rg * this.z_airIn[ico].getHumi();

        double ventN = this.z_airIn[ico].getFlowRate() * sec / this.z_airIn[ico].getDensity() /1000. /
this.zoneVolume[ico];
        //System.out.print( "ico ="+ico+" ventN = " + ventN );
        if( ventN > 1 ){
           //計算ステップで室容積分以上入れ替わるとき
           // spLoad[0] = -( coeffT[2] + ( -coeffT[2] * ( ventN - 1 ) + coeffT[0] * tx[0] ) / ventN );  //室顕熱負荷（冷
房：正）
           //            = -( (1-(ventN-1)/ventN ) *coeffT[2] + 1 / ventN * coeffT[0] * tx[0] );
           coeffT[0] *= 1 / ventN;
           coeffT[2] *= 1-(ventN-1)/ventN;
           //coeffT[3] = this.z_airIn[ico].getFlowRate();
           // spLoad[1] = -( coeffX[2] + ( -coeffX[2] * ( ventN - 1 ) + coeffX[0] * tx[1] ) / ventN );  //室潜熱負荷（除
湿：正）
           coeffX[0] *= 1 / ventN;
           coeffX[2] *= 1-(ventN-1)/ventN;
        }

        //建物側へ熱平衡式係数などを設定
        coreSpace[ico].setValues( ISpace.ATOMCOEFFT, coeffT );
        coreSpace[ico].setValues( ISpace.ATOMCOEFFX, coeffX );

        //室負荷計算
        double[] spLoad = new double[3];
        double[] tx = coreSpace[ico].getValues( ISpace.SPACETX );

        if( tx[1] < 0 ){
           //System.out.println( "update() tx[1] Humi "+tx[1]+"< 0 ***** --> 0   tx[0]="+tx[0] );
           tx[1] = 0;
        }

        //double ventN = this.z_airIn[ico].getFlowRate() * sec / this.z_airIn[ico].getDensity() /1000. /
this.zoneVolume[ico];
        //System.out.print( "ico ="+ico+" ventN = " + ventN );
        //if( ventN > 1 ){
        // //計算ステップで室容積分以上入れ替わるとき
        // spLoad[0] = -( coeffT[2] + ( -coeffT[2] * ( ventN - 1 ) + coeffT[0] * tx[0] ) / ventN );  //室顕熱負荷（冷
房：正）
        // spLoad[1] = -( coeffX[2] + ( -coeffX[2] * ( ventN - 1 ) + coeffX[0] * tx[1] ) / ventN );  //室潜熱負荷（除
湿：正）
        //}else{
        // spLoad[0] = -( coeffT[2] + coeffT[0] * tx[0] ); //室顕熱負荷（冷房：正）
        // spLoad[1] = -( coeffX[2] + coeffX[0] * tx[1] ); //室潜熱負荷（除湿：正）
        //}
        spLoad[0] = -( coeffT[2] + coeffT[0] * tx[0] ); //室顕熱負荷（冷房：正）
        spLoad[1] = -( coeffX[2] + coeffX[0] * tx[1] ); //室潜熱負荷（除湿：正）
        spLoad[2] = spLoad[0] + spLoad[1];        //室全熱負荷
        //System.out.println( " spLoaad[0]="+ spLoad[0] + " spLoaad[1]="+ spLoad[1]);
        //建物側へ室負荷値を設定
        coreSpace[ico].setValues( ISpace.SPACELOAD, spLoad );
     }
   }
*/
   public Object viewInternal(TestCommand cmd) {
      List<Object> result = new ArrayList<Object>();
      result.add(super.cm);
      result.add(super.rm);
      result.add(super.sm);

      result.add( this.modIn );
      result.add( this.sunIn.getSunDir() );
      return result;
   }


   /**
    * "0"で埋めた３桁の文字列整数を返す
    * @param i
    * @return
```

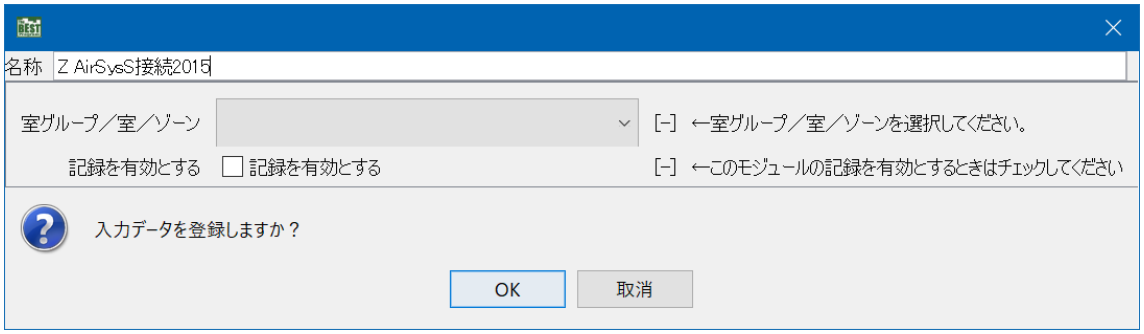
```
     */
    private String getOnum_3( int i ){
        if( i<0 ){
            return "000";
        }else if( i<10 ){
            return "00"+i;
        }else if( i<100 ){
            return "0"+i;
        }else if( i<1000 ){
            return ""+i;
        }else{
            return ""+i;
        }
    }
}
```

「Z AirSysS 接続 201502」（場所：設備 2015／Z ゾーン 2015／）

モジュール名	Z AirSysS 接続 201502
クラス	ZoneAirforSystemSModule201502*

（1）入力画面

・スペック



（2）モジュールの概要

指定したゾーンへ一組の吸排気の接続と環境情報の取り出しができるモジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

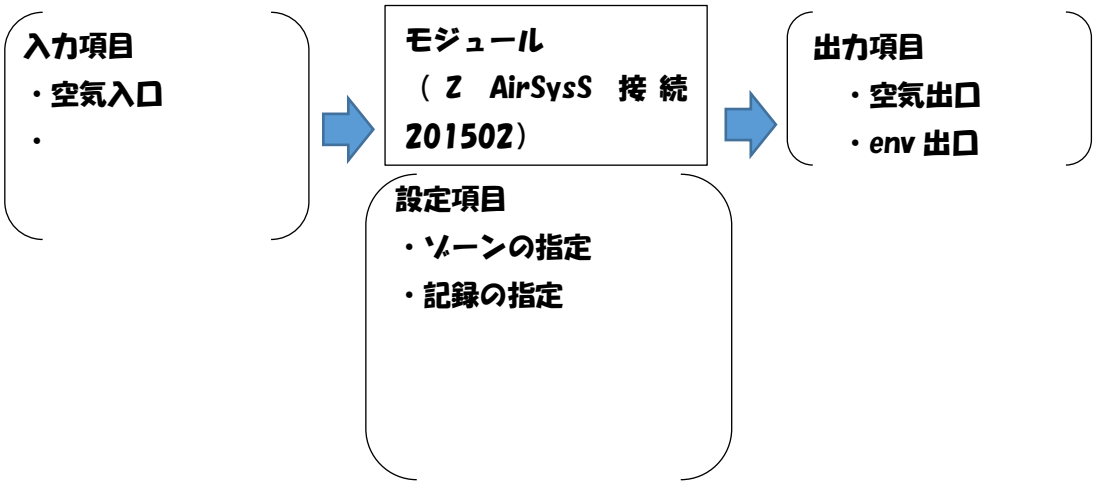


図1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	室グループ／室／ゾーン	String	coreSpace sysSpace	#RoomGroup	[-]	ー	ー		←室グループ／室／ゾーンを選択してください。
2	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	空気入口	L0_airIn	airIn	-	状態	空気	入口	
3	空気出口	L0_airOut	airOut	-	状態	空気	出口	
4	Env 出口	L0_envOut	envOut	-	状態	室環境	出口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	ZAirSys_Message#-#-	メッセージ	—	メッセージ
2	ZAirSys_室入口乾球温度#℃#温度	ZAirSys_室入口乾球温度	℃	入口
3	ZAirSys_室入口質量流量#g/s#質量流量	ZAirSys_室入口質量流量	g/s	入口
4	ZAirSys_室入口絶対湿度#g/g'#絶対湿度	ZAirSys_室入口絶対湿度	g/g	入口
5	ZAirSys_室入口 CO2 濃度#ppm#濃度	ZAirSys_室入口 CO2 濃度	ppm	入口
6	ZAirSys_室出口乾球温度#℃#温度	ZAirSys_室出口乾球温度	℃	出口
7	ZAirSys_室出口質量流量#g/s#質量流量	ZAirSys_室出口質量流量	g/s	出口
8	ZAirSys_室出口絶対湿度#g/g'#絶対湿度	ZAirSys_室出口絶対湿度	g/g	出口
9	ZAirSys_室出口 CO2 濃度#ppm#濃度	ZAirSys_室出口 CO2 濃度	ppm	出口
10	ZAirSys_室 PMV#-#PMV	ZAirSys_室 PMV	—	My
11	ZAirSys_室作用温度#℃#作用温度	ZAirSys_室作用温度	℃	My

（7）計算フロー・計算内容

・接続された L0_airIn（空調機器からのゾーンへの送風）を指定したゾーンへ伝達し建築側のゾーンの計算に反映させます。

・指定したゾーンの空気を L0_airOut（空調機器への還気）にセットします。

・指定したゾーンの環境情報を L0_envOut にセットします。

（8）データ範囲と範囲外の取扱い

特になし。

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.List;
import java.util.ArrayList;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;

import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.building.spaces.ISpace;
import jp.or.ibec.best.domain.building.spaces.ZoneEnv;


/**
 * 空調吹出空気情報を建物側に渡し、リターン空気を出力するモジュール
 * @author  kohri HIROSHI NINOMIYA 20080429 /20080501 /20080909 suganaga/2009/10/10
 *           NINOMIYA20150205
 *
 * 20150205/airIn,airOut  のみ対応のシンプルなZoneAirforSystemS  モジュール
 *   ZoneAirforSystemModule201310  を基に作成
 *
 */
public class ZoneAirforSystemSModule201502 extends AbstractBestModule implements
      IBestMetaModule, jp.or.ibec.best.dk.test.IProbeForUT {

   //
   private final String moduleName = "(ZoneAirforSystemSModule201502) ";

   //接続ノード
   private final String S_NODE_airOut = "L0_airOut";
   private final String S_NODE_airIn  = "L0_airIn";

   private final String S_NODE_envOut = "L0_envOut";

   //private final String C_NODE_swcIn  = "L1_swcIn";
   //private final String C_NODE_modeIn = "L1_modeIn";

   private final String R_NODE = "L2_recOut";

   //外部定義項目
   //仕様
   private final String SPEC_name     = "名称";
   private final String SPEC_grzName  = "室グループ／室／ゾーン";

   private final String SPEC_isRecord = "記録を有効とする";

   //記録
   private final String RECORD_message    = "ZAirSys_Message#-#-";
   private final String RECORD_T_airIn    = "ZAirSys_室入口乾球温度#℃#温度";
   private final String RECORD_M_airIn    = "ZAirSys_室入口質量流量#g/s#質量流量";
   private final String RECORD_X_airIn    = "ZAirSys_室入口絶対湿度#g/g'#絶対湿度";
   private final String RECORD_CO2_airIn  = "ZAirSys_室入口CO2濃度#ppm#濃度";
   private final String RECORD_T_airOut   = "ZAirSys_室出口乾球温度#℃#温度";
   private final String RECORD_M_airOut   = "ZAirSys_室出口質量流量#g/s#質量流量";
   private final String RECORD_X_airOut   = "ZAirSys_室出口絶対湿度#g/g'#絶対湿度";
   private final String RECORD_CO2_airOut = "ZAirSys_室出口CO2濃度#ppm#濃度";

   private final String RECORD_P_envOut   = "ZAirSys_室PMV#-#PMV";
   private final String RECORD_O_envOut   = "ZAirSys_室作用温度#℃#作用温度";

   //接続熱媒など
```

```
    private BestAir   airOut = null;
    private BestAir   airIn = null;
    private BestAir   airInMix  = null;

    private ZoneEnv   envOut = null;

    //制御信号など
    //private int swcIn;
    //private int modeIn;

    //仕様など
    private String name;          //名称

    private boolean isRecord = false;

    private StringBuffer message = new StringBuffer();


    private ISpace coreSpace   ;//建物側インスタンス
    private SystemISpace sysSpace = null;

    private ZoneAirSysCalc201310 zoneAirSysCalc = ZoneAirSysCalc201310.getInstance();

    private double zoneVolume = 0;
    private double zoneCO2ppm = 500.;


    @Override
    public void setProfile(BestSpecs spec) {
        if(spec == null) {
            return;
        }

        Map<String, String> map = spec.getSpec();
        if(map == null) {
            return;
        }

        //名称を取得
        this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

        //室グループ、ゾーン名の取得
        String grzName = null;
        grzName = spec.getSpecValue( this.SPEC_grzName, "" );

        this.coreSpace = ZoneGRZName.getCoreSpace( grzName, this.name );

        this.sysSpace = this.zoneAirSysCalc.getSystemISpace( grzName, this.coreSpace );

        //該当するゾーンの電力情報インスタンスを取得
        if( this.sysSpace!=null){
            this.zoneVolume = this.sysSpace.getISpace().getValue( ISpace.FLOORAREA ) *
this.sysSpace.getISpace().getValue( ISpace.CEILINGHEIGHT );
        }

        this.sysSpace.setVolume( this.zoneVolume );
        this.sysSpace.setCO2ppm( this.zoneCO2ppm );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

    }


    public void initialize(IBestStateMessage stateNodes,
            IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
            IBestDomainSchedule schedule) {
        //状態ノードを取得
        super.sm = stateNodes;

        //制御ノードを取得
        super.cm = commandNodes;
```

```
    //記録ノードを取得
    super.rm = recordNodes;

    //接続ノード　出口
    //airOut
    this.airOut = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOut );

    //envOut
    if( super.sm.getState( super.getConnectionNode( this.S_NODE_envOut)) != null ){
       this.envOut
       = (ZoneEnv) super.sm.getState( super.getConnectionNode( this.S_NODE_envOut));
    }else{
       this.envOut = new ZoneEnv();
       super.sm.setState( super.getConnectionNode( this.S_NODE_envOut ), this.envOut );
    }

    //入口
    //airIn
    this.airIn = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airIn );

    //airInMix
    this.airInMix = new BestAir();

    this.zoneAirSysCalc.setZoneIDandAir( this.sysSpace, this.airInMix );

  }


  public void outputs() {

    //
    //this.swcIn  = super.cm.getCommand(super.getConnectionNode( this.C_NODE_swcIn ));
    //this.modeIn = super.cm.getCommand(super.getConnectionNode( this.C_NODE_modeIn ));

    //接続　入口状態値の取得
    this.airIn
    =(BestAir)super.sm.getState( super.getConnectionNode( this.S_NODE_airIn ));

    BestAir.checkOpeData(this.airIn, "airIn", this.moduleName, "outputs()", this.name, this.isRecord, this.message);

    //airIn＊＊　の混合空気
    this.airInMix.copyAllvalState( this.airIn );

    //室状態値の取得
    double[] tx = this.sysSpace.getISpace().getValues(ISpace.SPACETX);
    if( tx == null ) {
       System.out.println("SpaceModule >>Error<< spaceTX data are not defined !!");
       return;
    }
    this.airOut.setTempDB(tx[0]);
    this.airOut.setHumi(tx[1]);
    this.airOut.setFlowRate( this.airIn.getFlowRate());
    this.airOut.setCO2ppm( this.sysSpace.getCO2ppm());

    //ノードに設定
    super.sm.setState(super.getConnectionNode( this.S_NODE_airOut ),
          this.airOut );

    //ZoneEnv
    this.envOut.setEnv( this.sysSpace.getISpace().getZoneEnv().getTa(),
          this.sysSpace.getISpace().getZoneEnv().getXa(),
          this.sysSpace.getISpace().getZoneEnv().getPMV(),
          this.sysSpace.getISpace().getZoneEnv().getOT() );

    //ノードに設定
    super.sm.setState( super.getConnectionNode( this.S_NODE_envOut), this.envOut );


    //記録
    if( this.isRecord  && super.rm != null ){
       this.record();
```

```
    }

    message.setLength(0);
}


/**
 * 記録
 */
private void record(){
    if( CheckPrintModule.isPrintMessage ){
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_message, this.name, this.message.toString() );
    }

    //if( CheckPrintModule.isPrintEnergy ){
    //}

    //if( CheckPrintModule.isPrintLoad ){
    //}

    if( CheckPrintModule.isPrintStateOut ){
        //出口
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_T_airOut, this.name,
                AirFormat.df_2( this.airOut.getTempDB() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_X_airOut, this.name,
                AirFormat.df_5( this.airOut.getHumi() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_M_airOut, this.name,
                AirFormat.df_2( this.airOut.getFlowRate() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_CO2_airOut, this.name,
                AirFormat.df_0( this.airOut.getCO2ppm() ));
    }

    if( CheckPrintModule.isPrintStateMy ){
        //ZoneEnv
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_P_envOut,this.name,
                AirFormat.df_2( this.envOut.getPMV() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_O_envOut,this.name,
                AirFormat.df_2( this.envOut.getOT() ));
    }

    if( CheckPrintModule.isPrintStateIn ){
        //入口
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_T_airIn, this.name,
                AirFormat.df_2( this.airIn.getTempDB() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_X_airIn, this.name,
                AirFormat.df_5( this.airIn.getHumi() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_M_airIn, this.name,
                AirFormat.df_2( this.airIn.getFlowRate() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_CO2_airIn, this.name,
                AirFormat.df_0( this.airIn.getCO2ppm() ));
    }
}


public void update() {
    //if( super.sm == null || S_NODE_aIn == null) return;

    this.zoneAirSysCalc.countUp();

}
```

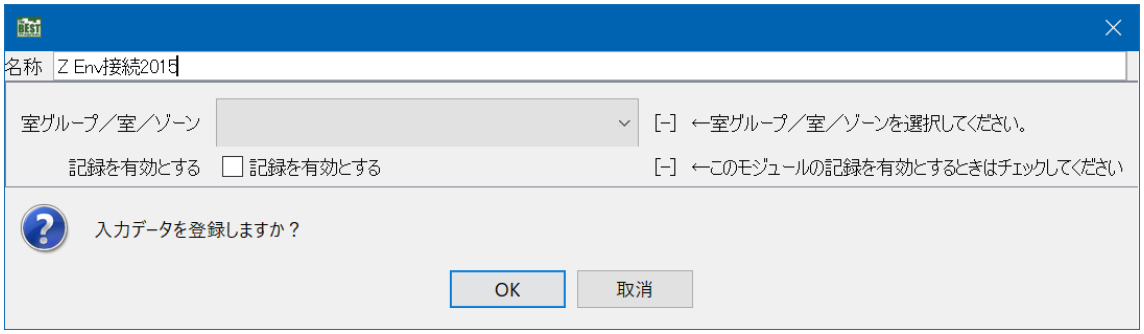
```
    public Object viewInternal(TestCommand cmd) {
        List<Object> result = new ArrayList<Object>();
        result.add(super.cm);
        result.add(super.rm);
        result.add(super.sm);

        //result.add( this.swcIn );
        return result;
    }
}
```

「Z Env 接続 2015」（場所：設備 2015／Z ゾーン 2015／）

モジュール名	Z Env 接続 2015
クラス	ZoneEnvModule201505

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールでは、負荷計算のゾーンを指定しそのゾーンの環境情報（ZoneEnv）を L0_envOut ノードから取り出すことができる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

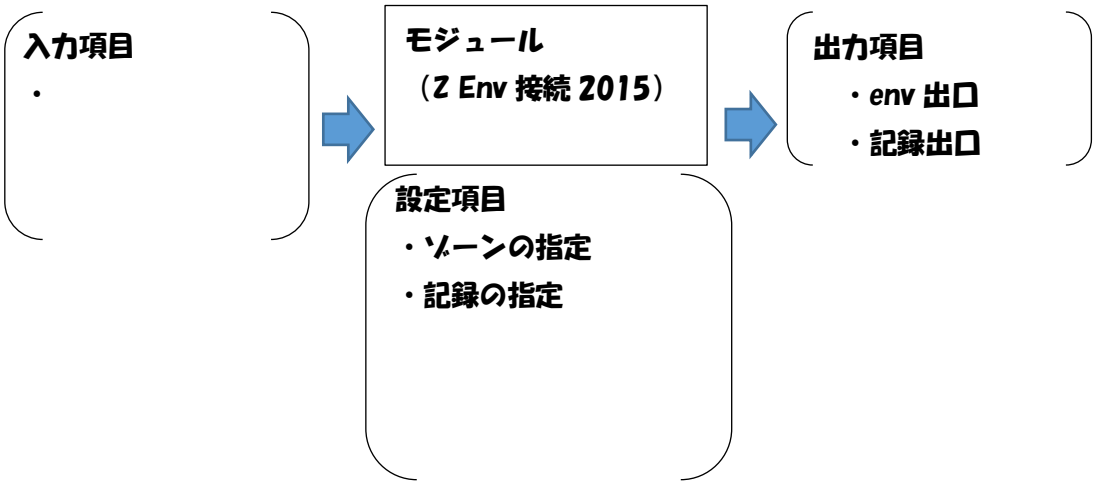


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	室グループ／室／ゾーン	String	coreSpace	#RoomGroup	[-]	ー	ー		←室グループ／室／ゾーンを選択してください。
2	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	Env 出口	L0_envOut	envOut	-	状態	室環境	出口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	ZEnv_Message#-#-	メッセージ	—	メッセージ
2	ZEnv_室乾球温度#℃#温度	室乾球温度	℃	出口
3	ZEnv_室絶対湿度#g/g'#絶対湿度	室絶対湿度	g/g	出口
4	ZEnv_室 PMV#-#PMV	室 PMV	—	出口
5	ZEnv_室作用温度#℃#作用温度	室作用温度	℃	出口

（7）計算フロー・計算内容

・指定されたゾーンから、環境情報（ZoneEnv）を取得する。

（8）データ範囲と範囲外の取扱い

特になし。

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.List;
import java.util.ArrayList;
import java.util.Map;

import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.building.spaces.ISpace;
import jp.or.ibec.best.domain.building.spaces.ZoneEnv;

/**
 * ゾーン熱環境を出力するモジュール
 * @author  kohri
 * 201505 NINOMIYA ゾーン
 */
public class ZoneEnvModule201505 extends AbstractBestModule implements
      IBestMetaModule, jp.or.ibec.best.dk.test.IProbeForUT {

   //
   private final String moduleName = "(ZoneEnvModule201505) ";

   //外部定義項目
   private final String SPEC_name     = "名称";
   private final String SPEC_grzName  = "室グループ／室／ゾーン";

   private final String SPEC_isRecord = "記録を有効とする";

   private final String S_NODE_envOut = "L0_envOut";

   private final String R_NODE = "L2_recOut";

   //記録
   private final String RECORD_message    = "ZEnv_Message#-#-";
   private final String RECORD_T_envOut   = "ZEnv_室乾球温度#℃#温度";
   private final String RECORD_X_envOut   = "ZEnv_室絶対湿度#g/g'#絶対湿度";
   private final String RECORD_P_envOut   = "ZEnv_室PMV#-#PMV";
   private final String RECORD_O_envOut   = "ZEnv_室作用温度#℃#作用温度";

   //仕様など
   private String name;          //名称

   private boolean isRecord = false;

   private StringBuffer message = new StringBuffer();

   private ISpace coreSpace;  //建物側室インスタンス

   private ZoneEnv envOut; //室環境インスタンス
   //ZoneEnvは、室温、絶対湿度、相対湿度、PMV、作用温度を保持する室環境クラス


   @Override
   public void setProfile(BestSpecs spec) {
      if(spec==null) return;
      Map<String, String> map=spec.getSpec();
      if(map==null) {
         System.out.println("(E) ZoneEnvModule : null spec map !!");
         return;
      }
```

```
        //名称を取得
        this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

        //室グループ、ゾーン名の取得
        String grzName = null;
        grzName = spec.getSpecValue( this.SPEC_grzName, "" );

        this.coreSpace = ZoneGRZName.getCoreSpace( grzName, this.name );

//      this.sysSpace = this.zoneAirSysCalc.getSystemISpace( grzName, this.coreSpace );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

    }


    public void initialize(IBestStateMessage stateNodes,
          IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
          IBestDomainSchedule schedule) {
        super.sm = stateNodes   ;     //状態ノードを取得
        super.cm = commandNodes ;     //制御ノードを取得
        super.rm = recordNodes  ;     //記録ノードを取得

        if(super.sm == null) return;

        //envOut
        if( super.sm.getState( super.getConnectionNode( this.S_NODE_envOut)) != null ){
            this.envOut
            = (ZoneEnv) super.sm.getState( super.getConnectionNode( this.S_NODE_envOut));
        }else{
            this.envOut = new ZoneEnv();
            super.sm.setState( super.getConnectionNode( this.S_NODE_envOut ), this.envOut );
        }

    }


    public void outputs() {
        if( super.sm == null) return;
        //室内環境状態値の取得と出力用インスタンスへの設定
        ZoneEnv env = coreSpace.getZoneEnv();
        this.envOut.setEnv( env.getTa(), env.getXa(), env.getPMV(), env.getOT() );

        //記録
        if( this.isRecord  && super.rm != null ){
            this.record();
        }

        message.setLength(0);

    }

    /**
     * 記録
     */
    private void record(){
        if( CheckPrintModule.isPrintMessage ){
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                  this.RECORD_message, this.name, this.message.toString() );
        }

        //if( CheckPrintModule.isPrintEnergy ){
        //}

        //if( CheckPrintModule.isPrintLoad ){
        //}

        if( CheckPrintModule.isPrintStateOut ){
            //ZoneEnv
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                  this.RECORD_T_envOut,this.name,
```

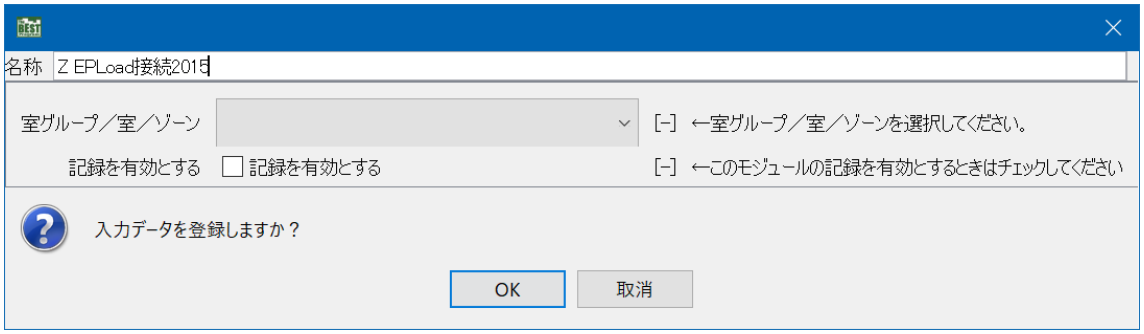
```
                AirFormat.df_2( this.envOut.getTa() ));
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_X_envOut,this.name,
                AirFormat.df_2( this.envOut.getXa() ));
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_P_envOut,this.name,
                AirFormat.df_2( this.envOut.getPMV() ));
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_O_envOut,this.name,
                AirFormat.df_2( this.envOut.getOT() ));
        }

        //if( CheckPrintModule.isPrintStateMy ){
        //}

        //if( CheckPrintModule.isPrintStateIn ){
        //}
    }


    public void update() {
    }

    public Object viewInternal(TestCommand cmd) {
        List<Object> result = new ArrayList<Object>();
        result.add(super.cm);
        result.add(super.rm);
        result.add(super.sm);
        return result;
    }
}
```

「Z EPLoad 接続 2015」（場所：設備 2015／Z ゾーン 2015／）

モジュール名	Z EPLoad 接続 2015
クラス	ZoneEPLoadModule201505

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールでは、負荷計算のゾーンを指定しそのゾーンの照明消費電力とコンセント消費電力を BestElectricity 媒体で L0_eleInLighting、L0_eleInConcent ノードから取り出すことができる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	室グループ／室／ゾーン	String	coreSpace	#RoomGroup	[-]	ー	ー		←室グループ／室／ゾーンを選択してください。
2	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	照明消費電力	L0_eleInLighting	eleInLighting	-	状態	電気	入口	
3	コンセント消費電力	L0_eleInConcent	eleInConcent	-	状態	電気	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	ZSys_Message#-#-	メッセージ	—	メッセージ
2	ZSys_室照明消費電力#W#電力 照明・コンセント 照明	室照明消費電力	W	エネルギー消費
3	ZSys_室コンセント消費電力#W#電力 照明･コンセント コンセント	室コンセント消費電力	W	エネルギー消費
4				

（7）計算フロー・計算内容

・指定されたゾーンから、照明消費電力とコンセント消費電力を取得する。

（8）データ範囲と範囲外の取扱い

とくになし。

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.List;
import java.util.ArrayList;
import java.util.Map;

import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.building.spaces.ISpace;
import jp.or.ibec.best.domain.building.spaces.ZoneElectric;

/**
 * ゾーン電力消費量をシステム側に渡すモジュール
 * @author  kohri
 * 201505 NINOIMYA
 * private final String S_NODE_out = "L0_eleOut"; ->"L0_eleIn"
 */
public class ZoneEPLoadModule201505 extends AbstractBestModule implements
      IBestMetaModule, jp.or.ibec.best.dk.test.IProbeForUT {

   //
   private final String moduleName = "(ZoneEnvModule201505) ";

   //外部定義項目
   private final String SPEC_name     = "名称";
   private final String SPEC_grzName  = "室グループ／室／ゾーン";

   private final String SPEC_isRecord = "記録を有効とする";

   private final String R_NODE = "L2_recOut";

   private final String S_NODE_eleInLight   = "L0_eleInLighting";
   private final String S_NODE_eleInConcent = "L0_eleInConcent";

   private final String RECORD_message    = "ZSys_Message#-#-";
   private final String RECORD_PPE_eleInLighting = "ZSys_室照明消費電力#W#電力 照明・コンセント 照明";
   private final String RECORD_PPE_eleInConcent  = "ZSys_室コンセント消費電力#W#電力 照明・コンセント コンセント";


   private String name = null;   //名称

   private boolean isRecord = false;//記録を有効とする=true


   private StringBuffer message = new StringBuffer();

   private ISpace coreSpace;  //建物側室インスタンス

   private ArrayList<ZoneElectric> zonesElectric;  //建物側電力情報インスタンス

   private BestElectricity eleInLight= null;
   private BestElectricity eleInConcent= null;


   @Override
   public void setProfile(BestSpecs spec) {
      if(spec==null) return;
      Map<String, String> map=spec.getSpec();
      if(map==null) {
         System.out.println("(E) ZoneEPLoadModule : null spec map !!");
         return;
```

```
        }


        //名称を取得
        this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

        //室グループ、ゾーン名の取得
        String grzName = null;
        grzName = spec.getSpecValue( this.SPEC_grzName, "" );

        this.coreSpace = ZoneGRZName.getCoreSpace( grzName, this.name );

        if( this.coreSpace != null ){
           ZoneElectric zoneElectric = this.coreSpace.getZoneElectric();
           if( zoneElectric != null ){
              zonesElectric = new ArrayList<ZoneElectric>();
              zonesElectric.add(zoneElectric);
           }
        }

//      this.sysSpace = this.zoneAirSysCalc.getSystemISpace( grzName, this.coreSpace );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

     }



     public void initialize(IBestStateMessage stateNodes,
           IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
           IBestDomainSchedule schedule) {
        super.sm = stateNodes   ;     //状態ノードを取得
        super.cm = commandNodes ;     //制御ノードを取得
        super.rm = recordNodes  ;     //記録ノードを取得

        if(super.sm == null) return;


        //eleInLighting
        this.eleInLight   = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleInLight );

        //eleInConcent
        this.eleInConcent = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleInConcent );

     }

     public void outputs() {

        if( super.sm == null) return;

        //Lighting Concent
        //システム側時間間隔が建物側の時間間隔と同じか大きいとき
        //ゾーン別電力量を集計して、出力側インスタンスに設定
        if( this.coreSpace == null ){
           this.eleInLight.setActivePower( 0 );
           this.eleInLight.setReactivePower( 0 );
           this.eleInConcent.setActivePower( 0 );
           this.eleInConcent.setReactivePower( 0 );

        }else
        if( this.zonesElectric.get(0).getIntegTimeStep() > 0. ){
           double[] totalEpl=new double[2];
           for( ZoneElectric zoneElectric : this.zonesElectric ){
              double[] epl0=zoneElectric.getElectricPowerLoad();
              for(int i=0; i<2; i++) totalEpl[i]+=epl0[i];
           }
           this.eleInLight.setActivePower( totalEpl[0] );
           this.eleInLight.setReactivePower( totalEpl[0] );
           this.eleInConcent.setActivePower( totalEpl[1] );
           this.eleInConcent.setReactivePower( totalEpl[1] );
```

```
            //System.out.println( "Lighting, Concent = " + totalEpl[0] + "," + totalEpl[1] );
        }

        //20081217
        super.sm.setState(super.getConnectionNode( this.S_NODE_eleInLight ), this.eleInLight );
        super.sm.setState(super.getConnectionNode( this.S_NODE_eleInConcent ), this.eleInConcent );

        if( this.isRecord ){
            this.record( );
        }
    }

    /**
     * 記録
     */
    private void record( ){

        if( CheckPrintModule.isPrintMessage ){
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                    this.RECORD_message, this.name, this.message.toString() );
        }

        if( CheckPrintModule.isPrintEnergy ){
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                    this.RECORD_PPE_eleInLighting, this.name,
                    AirFormat.df_2( this.eleInLight.getActivePower() )); //"ZoneforSystem室照明消費電力#W#電力 照明・コンセント 照明";
            super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                    this.RECORD_PPE_eleInConcent, this.name,
                    AirFormat.df_2( this.eleInConcent.getActivePower() )); //"ZoneforSystem室コンセント消費電力#W#電力 照明・コンセント コンセント";
        }
    }

    public void update() {
    }

    public Object viewInternal(TestCommand cmd) {
        List<Object> result = new ArrayList<Object>();
        result.add(super.cm);
        result.add(super.rm);
        result.add(super.sm);
        return result;
    }
}
```

「BMi 室内機 2015」（場所：設備 2015／個別分散 2015／）

モジュール名	BMi 室内機 2015
クラス	BMin_SModule2015（本体） BMinSpec2015（仕様） BMinCalc2015（計算）

（1）入力画面

・スペック

（2）モジュールの概要

ビル用マルチの室内機モジュールである。EHP 室外機、GHP 室外機に接続して使用する。

運転能力は、外部 PID 制御モジュールからの操作量により決定する。

外気導入、加湿、全熱交換器を計算できる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	ｎame	ー	[-]	ー	ー		
1	室グループ／室／ゾーン	String	zoneAirforSSinside	#RoomGroup	[-]	ー	ー		←室グループ／室／ゾーンを選択してください。
2	機器番号	String	m_no		[-]	ー	ー		
3	機器種別	String	m_ty	0_201303_基本タイプ、1_201303_外気処理エアコン全熱交なし、0_基本タイプ、1_外気処理エアコン全熱交なし201002	[-]	ー	ー		←チェックボックスから選択してください
4	機器型式	String	m_kt	#File 、BMList¥¥BMEHPinListA.csv	[-]	ー	ー		
5	台数	double	in_dai	1	[台]	ー	1		←ゾーンに異なる型番の機器がある場合には、入力型番に換算した台数を入力
6	■定格能力など■					ー	ー		←以下は１台当たりの仕様を入力してください
7	定格冷房能力	double	in_Qc_S	3.6	[kW]	ー	0		←１台あたりの仕様を入力。以下同様
8	定格暖房能力	double	in_Qh_S	4	[kW]	ー	0		
9	定格暖房能力（潜熱）	double	in_QhL_S	1	[kW]	ー	0		←機器種別で「1_外気処理エアコン～」を指定した場合に有効
10	定格風量	double	in_Va_S	870	[m3/h(a)]	ー	0		
11	定格消費電力　冷房時	double	in_PPEac_S	31	[W]	ー	0		
12	定格消費電力　暖房時	double	in_PPEah_S	27	[W]	ー	0		
13	機器起動停止負荷率	double	in_run_stop	10	[%]	100	0		←部分負荷率を入力してください
14	冷房能力補正係数	double		1	[-]	ー	0		←定格冷房能力を補正します

15	冷房入力補正係数	double		1	[-]	―	0		←定格冷房消費電力を補正します
16	暖房能力補正係数	double		1	[-]	―	0		←定格暖房能力を補正します
17	暖房入力補正係数	double		1	[-]	―	0		←定格暖房消費電力を補正します
18	冷媒管長	double	in_Lp	20	[m]	―	0		
19	冷媒管高低差	double	in_Lh	10	[m]	―	―		←室内機が下にある場合はマイナスで入力
20	■ファンの運転■					―	―		
21	室内機ファンと圧縮機を連動する	boolean	isFanOnOff	FALSE	[-]	―	―		←室内機ファンと圧縮機を連動運転するときはチェックしてください
22	■吹出し状態の算定法■					―	―		BESTEST ではバイパスファクター法（参考 7%）とする
23	吹出し状態算定法	String	typeCalcSA	0_相対湿度を設定、1_バイパスファクターを設定	[-]	―	―		←吹出し状態の算定方法を選択してください
24	設定値	double	valCalcSA	90	[%]	―	0		←参考（相対湿度 90%、バイパスファクター7%）
25	■外気・加湿■					―	―		
26	定格加湿能力	double	in_HUM_mx	1	[kg/h]	―	0		←加湿を行わない場合は　0 入力
27	加湿効率	double	in_HUM_eff	95	[%]	―	0		←参考（浸透膜式 95%、自然蒸発式 22%）
28	加湿飽和効率	double	in_HUM_rt	70	[%]	―	0		←吹出空気の相対湿度設定
29	加湿 On・Off 設定値	double	in_HUM_on	40	[%]	―	0		←吸込空気の相対湿度設定
30	取入外気量	double	in_OA_S	200	[m3/h(a)]	―	0		
31	外気量を外部 valInOARate で制御する	boolean		FALSE	[-]	―	―		←外気量を valInOARate×基準外気量で制御するときはチェックしてください
32	基準外気量	double		200	[m3/h(a)]	―	0		
33	全熱交換器効率	double	in_EX_S	60	[%]	100	0		←全熱交が無い場合は　0 入力
34	全熱交換器バイパスあり	boolean	isHexbypass	FALSE	[-]	―	―		←全熱交換器にバイパスがあるときはチェック

									してください
35	全熱交換器消費電力	double	in_EX_PE	0	[W]	―	0		
36	全熱交換器効率補正係数	double		1	[-]	―	0		←全熱交の効率を補正します
37	全熱交換器消費電力補正形数	double		1	[-]	―	0		←全熱交の消費電力を補正します
38	■電気■					―	―		
39	相数	int	phase	3	[相]	―	1		
40	電圧	double	voltage	200	[V]	―	100		
41	周波数	double	frequency	50	[Hz]	60	50		
42	力率	double	powerFactor	0.8	[-]	1	0		
43	■記録・グラフ表示■					―	―		
44	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
45	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください
46	■仮設調整■					―	―		
47	台数を調整する	boolean	isAdjust2012	FALSE	[-]	―	―		←台数を仮設調整するときはチェックしてください
48	調整の計算ステップ数	int	numAdjustSteps	18	[-]	―	6		←仮設調整する計算ステップ数を入力してください
49	調整にリストを使用する	boolean	isUseAdjustList	FALSE	[-]	―	―		←自動調整にリストを使用する場合はチェックしてください。

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	外気導入信号入口	L1_swcInOA	swcInOA	-	制御	OnOff 信号	入口	
5	PID モード入口	L1_modInPID	modInPID	-	制御	制御モード	入口	
6	外気入口	L0_airInOA	airInOA	-	状態	空気	入口	
7	還気入口	L0_airInRM	airInRM	-	状態	空気	入口	
8	給気出口	L0_airOutRM	airOutRM	-	状態	空気	出口	
9	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	
10	ドレン出口	L0_watOutD	watOutD	-	状態	水	出口	
11	冷媒出口	L0_valOutLine	valOutLine	-	値	値	出口	
12	操作量入口	L0_valInCtrl	valInCtrl	-	値	値	入口	
13	給水入口	L0_watInCW	watInCW	-	状態	水	入口	
14	外気導入率入口	L0_valInOARate	valInOARate	-	値	値	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	室内機_Message#-#-	メッセージ	—	メッセージ
2	室内機_処理顕熱量#W#熱量	処理顕熱量	W	負荷
3	室内機_処理冷却顕熱量#W#熱量	処理冷却顕熱量	W	負荷
4	室内機_処理加熱顕熱量#W#熱量	処理加熱顕熱量	W	負荷
5	室内機_処理全熱量#W#熱量	処理全熱量	W	負荷
6	室内機_処理冷却全熱量#W#熱量	処理冷却全熱量	W	負荷
7	室内機_処理加熱全熱量#W#熱量	処理加熱全熱量	W	負荷
8	室内機_消費電力#W#消費電力	消費電力	W	エネルギー消費量
9	室内機_吹出乾球温度#℃#温度	吹出乾球温度	℃	出口
10	室内機_吹出絶対湿度#g/g#湿度	吹出絶対湿度	g/g	出口
11	室内機_吹出風量#g/s#質量流量	吹出風量	g/s	出口
12	室内機_吹出 CO2 濃度#ppm#濃度	吹出 CO2 濃度	ppm	出口
13	室内機_ドレン量#g/s#質量流量	ドレン量	g/s	出口
14	室内機_加湿給水量#g/s#質量流量	加湿給水量	g/s	入口
15	室内機_還気乾球温度#℃#温度	還気乾球温度	℃	入口
16	室内機_還気湿球温度#℃#温度	還気湿球温度	℃	入口
17	室内機_還気絶対湿度#g/g#湿度	還気絶対湿度	g/g	入口
18	室内機_還気 CO2 濃度#ppm#濃度	還気 CO2 濃度	ppm	入口
19	室内機_入口乾球温度#℃#温度	入口乾球温度	℃	入口
20	室内機_入口絶対湿度#g/g#湿度	入口絶対湿度	g/g	入口
21	室内機_入口湿球温度#℃#温度	入口湿球温度	℃	入口
22	室内機_外気乾球温度#℃#温度	外気乾球温度	℃	入口
23	室内機_外気絶対湿度#g/g#湿度	外気絶対湿度	g/g	入口
24	室内機_外気質量流量#g/s#質量流量	外気質量流量	g/s	入口
25	室内機_外気 CO2 濃度#ppm#濃度	外気 CO2 濃度	ppm	入口
26	室内機_調整冷却能力#W#熱量	調整冷却能力	W	調整
27	室内機_調整加熱能力#W#熱量	調整加熱能力	W	調整
28	室内機_調整台数#-#台数	調整台数	—	調整
29	室内機_調整ステップ平均台数#-#台数	調整ステップ平均台数	—	調整
30	室内機_調整仕様#-#仕様	調整仕様	—	調整
31	室内機_全熱交処理顕熱量#W#熱量	全熱交処理顕熱量	W	My

32	室内機_全熱交処理潜熱量#W#熱量	全熱交処理潜熱量	W	My
33	室内機_全熱交処理全熱量#W#熱量	全熱交処理全熱量	W	My

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

■BMin_SModule2015

```
package jp.or.ibec.best.domain.sample.air;



import java.util.Map;

import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;

import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * @author SUGANAGA 2008/04/07 /20080428 /NINOMIYA/20101212
 * ビルマルチ：室内機クラス
 *
 * 冷暖同時追加/20101212nino
 */
public class BMin_SModule2015 extends AbstractBestModule implements
IBestMetaModule  {

   /**
    * 論理ノード
    */
   private final String moduleName="(BMin_SModule2015)";


   private IBestStateMessage stateNodes = null;
   private IBestControlMessage commandNodes = null;
   private IBestRecordMessage recordNodes = null;

   private BMinSpec2015 bminSpec = null;


   @Override
   public void setProfile(BestSpecs spec) {

      // 外部定義項目取得
      if(spec == null) {
         return;
      }
      Map<String,String> map=spec.getSpec();
      if(map == null) {
         return;
      }

      this.bminSpec = new BMinSpec2015();

      BMinCalc2015.setProfile( spec, this.bminSpec );

   }


   @Override
   public void initialize(IBestStateMessage stateNodes,
         IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
         IBestDomainSchedule schedule) {


      this.stateNodes = stateNodes;
      this.commandNodes = commandNodes;
      this.recordNodes = recordNodes;
```

```
        super.sm = stateNodes   ;     //状態ノードを取得
        super.cm = commandNodes ;     //制御ノードを取得
        super.rm = recordNodes  ;     //記録ノードを取得

        BMinCalc2015.initialize( super.transferMapState, this, this.stateNodes, this.commandNodes, this.recordNodes,
schedule, this.bminSpec );


    }

    /**
     * ビルマルチ計算及び結果出力/室外機が先に動いています。
     */

    @Override
    public void outputs() {

        if( !this.bminSpec.isCalc ){
            return;
        }

        if (null == super.cm || null == super.sm )
            return;

        BMinCalc2015.outputs_calc( this, this.stateNodes, this.commandNodes, this.recordNodes, this.bminSpec );
    }


    public void update(){

        BMinCalc2015.update( this, this.stateNodes, this.commandNodes, this.recordNodes, this.bminSpec );
    }

}
```

■BMinSpec2015

```
package jp.or.ibec.best.domain.sample.air;


import java.util.LinkedList;
import java.util.Map;

import jp.or.ibec.best.DO.BM_EHPdata;
import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.DO.BestWater;

/**
 * @author SUGANAGA 2008/04/07 /20080428 /NINOMIYA/20101212
 * ビルマルチ：室内機クラス
 *
 * 冷暖同時追加/20101212nino
 */
public class BMinSpec2015 {

    /**
     * 論理ノード
     */
    protected final String moduleName="(BMinSpec2015)";

    protected Map<String, BM_EHPdata> rmlinemap=null;

    /**
     * 外部定義
     */



    /**
     * 変数（外部定義に対応）
     */

    protected String name            ;//= "機器名称";

    protected boolean isCalc = true;//このモジュールを計算する

    protected String m_no         ;//= "機器番号";
    protected String m_ty            ;//= "機器種別";
    protected String m_kt         ;//= "機器型式";
    protected double in_dai       ;//= "台数";
    protected double in_Qc_S      ;//= "定格冷房能力";
    protected double in_Qh_S         ;//= "定格暖房能力";
    protected double in_QhL_S     ;//= "定格暖房能力（潜熱）";
    protected double in_Va_S         ;//= "定格風量";
    protected double in_PPEac_S      ;//= "定格消費電力冷房時";
    protected double in_PPEah_S      ;//= "定格消費電力暖房時";
    protected double in_TKEa_R=0.0   ;//= "定格待機電力(運転時)";
    protected double in_TKEa_S=0.0   ;//= "定格待機電力(停止時)";
    protected double in_run_stop     ;//= "機器起動停止負荷率";
    protected double in_Lp        ;//= "冷媒管長";
    protected double in_Lh        ;//= "冷媒管高低差";
    protected double in_HUM_mx    ;//= "定格加湿能力";
    protected double in_HUM_eff   ;//= "加湿効率";
    protected double in_HUM_rt    ;//= "加湿飽和効率";
    protected double in_HUM_on    ;//= "加湿On・Off設定値";
    protected double in_OA_S         ;//= "取入外気量";
    protected double in_OA_100    ;//= "基準外気量";
    protected boolean isScheduleOA = false;//
    protected double in_EX_S         ;//= "全熱交換器効率";
    protected boolean isHexbypass = false;//熱交バイパスあり=true
    protected double in_EX_PE     ;//= "全熱交換器消費電力[W]";//20130717
```

```
    protected int phase        ;//= "相数";
    protected double voltage      ;//= "電圧";
    protected double frequency   ;// = "周波数";
    protected double powerFactor  ;//= "力率";

    protected boolean isFanOnOff = false;

    protected String typeCalcSA = null;//"吹出し状態算定法";
    protected int numCalcSA;//
    protected double valCalcSA;//"吹出し状態算定法設定値";

    protected boolean isGVisible = false;
    protected boolean isRecord   = false;

    protected StringBuffer message= new StringBuffer();

    protected BestAir airInOA   = null;
    protected BestAir airInRM   = null;
    protected BestAir airOutRM  = null;       //給気
    protected BestWater watOutD = null;
    protected BestWater watInCW = null;
    protected BestElectricity eleIn = null;   //電力
    protected BM_EHPdata RMdata = new BM_EHPdata(); //電力
    protected BestValue PIDrate = null;
    protected BestValue valInOARate = null;//"LO_valInOARate";//外気流量率
// private Double CAPrate  = null;
    protected Double CAPrateC = null;
    protected Double CAPrateH = null;
    protected Double LPrate   = 1.;//20120823nino

    protected PACDBManager pacDB=null;

    protected double maxTwatIn_C_FACKctw;// = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKctw() );
    protected double minTwatIn_C_FACKctw;// = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKctw() ) ;
    protected double maxTdbOA_C_FACKctai;// = bminspec.pacDB.getformulaRangeMaxRaw( bminspec.pacDB.getFACKctai() );
    protected double minTdbOA_C_FACKctai;// = bminspec.pacDB.getformulaRangeMinRaw( bminspec.pacDB.getFACKctai() );
    protected double maxTdbOA_C_FACKcta;// = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKcta() );
    protected double minTdbOA_C_FACKcta;// = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKcta() );
    protected double maxTwbIn_C_FACKctai;// = bminspec.pacDB.getformulaRangeMaxCol( bminspec.pacDB.getFACKctai() );
    protected double minTwbIn_C_FACKctai;// = bminspec.pacDB.getformulaRangeMinCol( bminspec.pacDB.getFACKctai() );
    protected double maxTwbIn_C_FACKcti;// = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKcti() );
    protected double minTwbIn_C_FACKcti;// = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKcti() );
    protected double maxTdbOA_C_FACKctdb;// = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKctdb() );
    protected double minTdbOA_C_FACKctdb;// = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKctdb() );
    protected double maxTwbOA_C_FACKctwb;// = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKctwb() );
    protected double minTwbOA_C_FACKctwb;// = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKctwb() );

    protected double maxTwatIn_H_FACKhtw;// = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKhtw() );
    protected double minTwatIn_H_FACKhtw;// = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKhtw() ) ;
    protected double maxTwbOA_H_FACKhtai;// = bminspec.pacDB.getformulaRangeMaxRaw( bminspec.pacDB.getFACKhtai() );
    protected double minTwbOA_H_FACKhtai;// = bminspec.pacDB.getformulaRangeMinRaw( bminspec.pacDB.getFACKhtai() );
    protected double maxTwbOA_H_FACKhta;// = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKhta() );
    protected double minTwbOA_H_FACKhta;// = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKhta() );
    protected double maxTdbIn_H_FACKhtai;// = bminspec.pacDB.getformulaRangeMaxCol( bminspec.pacDB.getFACKhtai() );
    protected double minTdbIn_H_FACKhtai;// = bminspec.pacDB.getformulaRangeMinCol( bminspec.pacDB.getFACKhtai() );
    protected double maxTdbIn_H_FACKhti;// = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKhti() );
    protected double minTdbIn_H_FACKhti;// = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKhti() );
    protected double maxTdbOA_H_FACKhtdb;// = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKhtdb() );
    protected double minTdbOA_H_FACKhtdb;// = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKhtdb() );
    protected double maxTwbOA_H_FACKhtwb;// = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKhtwb() );
    protected double minTwbOA_H_FACKhtwb;// = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKhtwb() );


    protected int swcIn ;
    protected int swcInOA ;
    protected int modIn ;
    protected int modInPID;
    protected int mState;   //停止、冷房、暖房フラグ

    protected double in_Va_opeS         ;//= "定格風量";

    /**
```

```
     * 出力（建物）
     */

    protected double DBin;          //室内吸込み乾球温度[℃]20130109
    protected double WBin;          //室内吸込み湿球温度[℃]20130109
    protected double XGin;
    protected double IAin;

    protected double DBoa;          //外気乾球温度[℃]
    protected double WBoa;          //外気湿球温度[℃]
    protected double XGoa;
    protected double IAoa;
    protected double Moa;

    protected double DBra;          //室内乾球温度[℃]20130109
    protected double WBra;          //室内湿球温度[℃]20130109
    protected double XGra;          //
    protected double IAra;          //20130109
    protected double RHra;          //20130109

    protected double DBout;         //乾球温度[℃]
    protected double XGout;         //絶対湿度[kg/kg.DA]

    protected double GW;            //風量(g/s)
    protected double D_Wrate;       //ドレン水量(g/s)
    protected double CW_Wrate;      //加湿水量(g/s)

    protected double PPE;
    protected double E_PPE;
    protected double S_Load;
    protected double T_Load;
    protected double Sc_Load;
    protected double Tc_Load;
    protected double Sh_Load;
    protected double Th_Load;
    protected double S_Load_RM;//室内機処理熱量（顕熱）[W]
    protected double T_Load_RM;//室内機処理熱量（全熱）[W]
    protected double Sc_Load_RM;//室内機処理熱量（顕熱）[W]
    protected double Tc_Load_RM;//室内機処理熱量（全熱）[W]
    protected double Sh_Load_RM;//室内機処理熱量（顕熱）[W]
    protected double Th_Load_RM;//室内機処理熱量（全熱）[W]

    protected double qSTHEX;//全熱交顕熱量
    protected double qLTHEX;//全熱交潜熱量
    protected double qTTHEX;//全熱交全熱量

    //******************090725-s
    protected double saijohatsu;
    protected double T_C_heat;
// private int u40=0;

//******************090725-e
    protected int dbflg=0;
    protected String path=null;//"EHP";
    protected String[] filenames= new String[2];//20110113nino
    protected String equipmentName;
    //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    protected int HEXselect=1;//1=エンタルピ基準　2=温度基準
    //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    protected boolean isOAProcessingUnit = false;//外気処理専用の場合=true

    protected boolean isSelectSPECList = false;//機器SPECListの機器を指定している=true

    protected boolean isNeedMoreData = false;//計算のためのデータが不足している＝true

    //グラフ表示など
    protected GraphJFrameBuilMultiIn_S20101212 gBMIn = null;

    //仮設調整モード2012
```

```
    protected boolean isAdjust2012  = false;//true="台数を調整する";//
    //private boolean isAdjust2012 = false;//20120406nino 建築学会大会空衛学会OS論文検討用＝true

    protected boolean isUseAdjustList = false;//true="リストで調整する";
    protected int numList = 0;

    protected double[] adjustQcListkW = {2.8, 3.6, 4.5, 5.6};//
    protected double[] adjustPPEcListkW = {0.033, 0.033, 0.047, 0.052};//
    protected double[] adjustQhListkW = {3.2, 4.0, 5.0, 6.3};//
    protected double[] adjustPPEhListkW = {0.027, 0.027, 0.034, 0.038};//
    protected double[] adjustfRateListm3min = {12.5, 12.5, 14.5, 15.5};//

    protected int numAdjustSteps;//"調整の計算ステップ数";
    protected LinkedList<Double> daiList = null;//必要台数
    protected double maxAdjustdai = 1;
    protected double avedaiAdjust = 0;
    protected double in_daiAdjust;//= "調整台数";
    protected double in_run_stop_Num;//= "機器起動停止負荷率"台数補正;

    protected double in_Qc_S_1;//= "定格冷房能力";
    protected double in_Qh_S_1;//= "定格暖房能力";
    protected double in_QhL_S_1;//= "定格暖房能力（潜熱）";
    protected double in_Va_S_1;//= "定格風量";
    protected double in_HUM_mx_1  ;//= "定格加湿能力";
    protected double in_PPEac_S_1;//= "定格消費電力冷房時";
    protected double in_PPEah_S_1;//= "定格消費電力暖房時";
    protected double in_OA_S_1 ;//= "取入外気量";
    protected double in_OA_100_1  ;//= "基準外気量";

    protected boolean isALLOA = false;//全外気か？
    protected boolean is2018SinseiOA = false;//20181018誘導基準申請OA処理方法

    protected ZoneAirforSSinsideModule201502 zoneAirforSSinside = null;
    protected String grzName = null;
    protected boolean isUseZoneAir = false;

}
```

■

```
package jp.or.ibec.best.domain.sample.air;


import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;

import jp.or.ibec.best.DO.BM_EHPdata;
import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.client.bestgui.file.service.FileConstants;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.essentials.utility.Psychrometrics;
import jp.or.ibec.best.log.BestLogger;
import jp.or.ibec.best.message.BestEngineMessageHandler;
import jp.or.ibec.best.message.BestEngineMessageParam;
import jp.or.ibec.best.message.constants.SystemMessageConstants;

/**
 * @author SUGANAGA 2008/04/07 /20080428 /NINOMIYA/20101212
 * ビルマルチ：室内機クラス
 *
 * 冷暖同時追加/20101212nino
 */
public class BMinCalc2015 {

   /**
    * 論理ノード
    */
// private final String moduleName="(BMinCalc2015)";

   private static final String S_NODE_airInOA     = "L0_airInOA";//外気
   private static final String S_NODE_airInRM     = "L0_airInRM";//室内吸込口
   private static final String S_NODE_airOutRM    = "L0_airOutRM";//室内吸込口
   private static final String S_NODE_eleIn       = "L0_eleIn";//電力
   private static final String S_NODE_watOutD     = "L0_watOutD";//コイル出口ドレン

   private static final String S_NODE_valInPID    = "L0_valInCtrl";//供給熱量 from PID          : in

   private static final String S_NODE_valInOARate = "L0_valInOARate";//外気流量率

   private static final String S_NODE_valOutLine  = "L0_valOutLine";//冷媒配管
   private static final String S_NODE_watInCW     = "L0_watInCW";//watInCW---090407追加**************
// private static Map<String, BM_EHPdata> rmlinemap=null;
   //
   private static final String C_NODE_swcIn    = "L1_swcIn";//運転状態：off/on
   private static final String C_NODE_swcInOA  = "L1_swcInOA";//外気取入状態：off/on
   private static final String C_NODE_modIn    = "L1_modIn";//空調モード：停止/冷却/加熱
   private static final String C_NODE_modInPID = "L1_modInPID";//PIDモジュールからのモード：/冷却/加熱


   private static final String R_NODE = "L2_recOut";


   /**
    * 外部定義
    */

   private static final String SPEC_name       = "名称";

   private static final String SPEC_isCalc = "このモジュールを計算する";//行政ツール用
```

```
private static final String SPEC_grzName  = "室グループ／室／ゾーン";

private static final String SPEC_m_no       = "機器番号";
private static final String SPEC_m_ty          = "機器種別";
private static final String SPEC_m_kt       = "機器型式";
private static final String SPEC_in_dai        = "台数[-]";
private static final String SPEC_in_Qc_S       = "定格冷房能力[W]";
private static final String SPEC_in_Qh_S       = "定格暖房能力[W]";
private static final String SPEC_in_QhL_S   = "定格暖房能力（潜熱）[W]";
private static final String SPEC_in_Va_S       = "定格風量[g/s]";
private static final String SPEC_in_PPEac_S    = "定格消費電力冷房時[W]";
private static final String SPEC_in_PPEah_S    = "定格消費電力暖房時[W]";
// private static final String SPEC_in_TKEa_R       = "定格待機電力(運転時)[W]";
// private static final String SPEC_in_THEa_S       = "定格待機電力(停止時)[W]";
private static final String SPEC_in_run_stop   = "機器起動停止負荷率[-]";
private static final String SPEC_in_Lp         = "冷媒管長[m]";
private static final String SPEC_in_Lh         = "冷媒管高低差[m]";
private static final String SPEC_in_HUM_mx     = "定格加湿能力[g/s]";
private static final String SPEC_in_HUM_eff    = "加湿効率[-]";
private static final String SPEC_in_HUM_rt     = "加湿飽和効率[-]";
private static final String SPEC_in_HUM_on     = "加湿On・Off設定値[-]";
private static final String SPEC_in_OA_S       = "取入外気量[g/s]";
private static final String SPEC_isScheduleOA  = "isScheduleOA";
private static final String SPEC_in_OA_100     = "基準外気量[g/s]";
private static final String SPEC_in_EX_S       = "全熱交換器効率[-]";
private static final String SPEC_isHexbypass   = "全熱交換器バイパスあり[-]";
private static final String SPEC_in_EX_PE   = "全熱交換器消費電力[W]";//20130717

private static final String SPEC_Phase         = "相数[-]";
private static final String SPEC_Voltage       = "電圧[V]";
private static final String SPEC_Frequency     = "周波数[Hz]";
private static final String SPEC_PowerFactor   = "力率[-]";

private static final String SPEC_isFanOnOff  = "室内機ファン連動";

private static final String SPEC_typeCalcSA  = "吹出し状態算定法";
private static final String SPEC_valCalcSA   = "吹出し状態算定法設定値";

private static final String SPEC_isGVisible  = "グラフを表示する";
private static final String SPEC_isRecord       = "記録を有効とする";

private static final String SPEC_isAdjust2012   = "台数を調整する";
private static final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";
private static final String SPEC_isUseAdjustList = "is調整リストを使用する";

/**
 * 出力（建物）
 */
private static final String RECORD_message = "室内機_Message#-#-";
private static final String RECORD_ID1     = "室内機_処理顕熱量#W#熱量";
private static final String RECORD_ID1c    = "室内機_処理冷却顕熱量#W#熱量";
private static final String RECORD_ID1h    = "室内機_処理加熱顕熱量#W#熱量";
private static final String RECORD_ID2     = "室内機_処理全熱量#W#熱量";
private static final String RECORD_ID2c    = "室内機_処理冷却全熱量#W#熱量";
private static final String RECORD_ID2h    = "室内機_処理加熱全熱量#W#熱量";
private static final String RECORD_ID3     = "室内機_消費電力#W#消費電力";
private static final String RECORD_Tsa     = "室内機_吹出乾球温度#℃#温度";
private static final String RECORD_Xsa     = "室内機_吹出絶対湿度#g/g#湿度";
private static final String RECORD_Msa     = "室内機_吹出風量#g/s#質量流量";
private static final String RECORD_CO2ppmsa= "室内機_吹出CO2濃度#ppm#濃度";
private static final String RECORD_ID7     = "室内機_ドレン量#g/s#質量流量";
private static final String RECORD_ID8     = "室内機_加湿給水量#g/s#質量流量";
private static final String RECORD_DBra = "室内機_還気乾球温度#℃#温度";
private static final String RECORD_WBra = "室内機_還気湿球温度#℃#温度";
private static final String RECORD_Xra  = "室内機_還気絶対湿度#g/g#湿度";
private static final String RECORD_CO2ppmra= "室内機_還気CO2濃度#ppm#濃度";
private static final String RECORD_DBin = "室内機_入口乾球温度#℃#温度";
private static final String RECORD_Xin  = "室内機_入口絶対湿度#g/g#湿度";
private static final String RECORD_WBin = "室内機_入口湿球温度#℃#温度";
private static final String RECORD_DBoa = "室内機_外気乾球温度#℃#温度";
private static final String RECORD_Xoa  = "室内機_外気絶対湿度#g/g#湿度";
```

```
private static final String RECORD_Moa  = "室内機_外気質量流量#g/s#質量流量";
private static final String RECORD_CO2ppmoa= "室内機_外気CO2濃度#ppm#濃度";

private static final String RECORD_in_Qc_Adjust = "室内機_調整冷却能力#W#熱量";
private static final String RECORD_in_Qh_Adjust = "室内機_調整加熱能力#W#熱量";
private static final String RECORD_in_daiAdjust = "室内機_調整台数#-#台数";
private static final String RECORD_avedaiAdjust = "室内機_調整ステップ平均台数#-#台数";
private static final String RECORD_in_specAdjust = "室内機_調整仕様#-#仕様";

private static final String RECORD_qSTHEX = "室内機_全熱交処理顕熱量#W#熱量";
private static final String RECORD_qLTHEX = "室内機_全熱交処理潜熱量#W#熱量";
private static final String RECORD_qTTHEX = "室内機_全熱交処理全熱量#W#熱量";



//@Override
public static void setProfile( BestSpecs spec, BMinSpec2015 bminspec ) {

   // 外部定義項目取得
   if(spec == null) {
      return;
   }
   Map<String,String> map=spec.getSpec();
   if(map == null) {
      return;
   }

   //isCalcを取得
   bminspec.isCalc = spec.getSpecValue( SPEC_isCalc, true );

   if( !bminspec.isCalc ){
      return;
   }

   // 機器名称
   bminspec.name = spec.getSpecValue( SPEC_name, bminspec.moduleName );

   bminspec.grzName = spec.getSpecValue( SPEC_grzName, "" );
   if( !bminspec.grzName.equals( "" )){
      bminspec.isUseZoneAir = true;
   }

   if( bminspec.isUseZoneAir ){
      bminspec.zoneAirforSSinside = new ZoneAirforSSinsideModule201502();
      bminspec.zoneAirforSSinside._setProfile( bminspec.name, bminspec.grzName );
   }

   // 機器番号
   if(null!=map.get( SPEC_m_no)){
      bminspec.m_no = (String)map.get( SPEC_m_no );
      //******************090725-s
      try{
         if(bminspec.m_no.substring(0 , 1).equals("+")==true){
            bminspec.saijohatsu=Double.parseDouble( bminspec.m_no.substring(1 , 3));
         }else{
            bminspec.saijohatsu=50.0;
         }
      }
      catch(NumberFormatException e){
         bminspec.saijohatsu=50.0;
      }
      //******************090725-s
   } else {
      ///System.out.println("(W)SPEC_機器番号がありません");
      bminspec.saijohatsu=50.0;  //******************090725
   }
   //    System.out.println("加湿="+saijohatsu);


   //  機器種別  0_基本タイプ、1_外気処理エアコン全熱交なし201002
   bminspec.m_ty = spec.getSpecValue( SPEC_m_ty, "0_201303_基本タイプ" );
```

```
        // 機器型式
        bminspec.m_kt = spec.getSpecValue( SPEC_m_kt, "" );

        FileReadBMSpecModule20110221 bmSpecFR = null;

        //機器specの取り込み
        if( bminspec.m_kt == null || bminspec.m_kt.equals( "" )){
           //機器型番指定が無い時は、ダイアログ入力値を使用する
        }else{
           //機器型番指定の時は、外部SPECListのデータを使用する
           bminspec.isSelectSPECList = true;
           //BM機器SPECListA　のデータの取込み
             String filepathName = FileConstants.DIR_SYSTEM + "/BMList/BMEHPinSPECListA.csv";//**************SPECファイル名

           bmSpecFR = new FileReadBMSpecModule20110221( filepathName );
        }



        // 台数
        bminspec.in_dai = spec.getSpecValue( SPEC_in_dai, 1. );

        bminspec.in_daiAdjust = bminspec.in_dai;

        // 定格冷房能力
        bminspec.in_Qc_S = spec.getSpecValue( SPEC_in_Qc_S, 0. ) * bminspec.in_dai;
        bminspec.in_Qc_S_1 = spec.getSpecValue( SPEC_in_Qc_S, 0. );

        if( bminspec.isSelectSPECList ){
           bminspec.in_Qc_S = Double.parseDouble( bmSpecFR.getSpec( bminspec.m_kt, "NCCapacity" ) ) * 1000. *
bminspec.in_dai;//kW->W
           bminspec.in_Qc_S_1 = Double.parseDouble( bmSpecFR.getSpec( bminspec.m_kt, "NCCapacity" ) ) * 1000.;//kW->W
        }

        // 定格暖房能力
        bminspec.in_Qh_S = spec.getSpecValue( SPEC_in_Qh_S, 0. ) * bminspec.in_dai;
        bminspec.in_Qh_S_1 = spec.getSpecValue( SPEC_in_Qh_S, 0. );

        if( bminspec.isSelectSPECList ){
           bminspec.in_Qh_S = Double.parseDouble( bmSpecFR.getSpec( bminspec.m_kt, "NHCapacity" ) ) * 1000. *
bminspec.in_dai;//kW->W
           bminspec.in_Qh_S_1 = Double.parseDouble( bmSpecFR.getSpec( bminspec.m_kt, "NHCapacity" ) ) * 1000.;//kW->W
        }

        // 定格暖房能力（潜熱）
        bminspec.in_QhL_S = spec.getSpecValue( SPEC_in_QhL_S, 0. ) * bminspec.in_dai;
        bminspec.in_QhL_S_1 = spec.getSpecValue( SPEC_in_QhL_S, 0. );

        //if( this.isSelectSPECList ){
        // this.in_QhL_S = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "" ) ) * 1000.*in_dai;//kW->W
        //}

        // 定格風量
        bminspec.in_Va_S = spec.getSpecValue( SPEC_in_Va_S, 0. ) * bminspec.in_dai;
        bminspec.in_Va_S_1 = spec.getSpecValue( SPEC_in_Va_S, 0. );

        if( bminspec.isSelectSPECList ){
           bminspec.in_Va_S = Double.parseDouble( bmSpecFR.getSpec( bminspec.m_kt, "NCFRair" ) ) * 1200 / 3600. *
bminspec.in_dai;//CMH->g/s
           bminspec.in_Va_S_1 = Double.parseDouble( bmSpecFR.getSpec( bminspec.m_kt, "NCFRair" ) ) * 1200 / 3600.;//CMH-
>g/s
        }
        //    System.out.println(this.moduleName+" "+name+"  "+in_Va_S);

        // 定格消費電力冷房時[W]
        bminspec.in_PPEac_S = spec.getSpecValue( SPEC_in_PPEac_S, 0. ) * bminspec.in_dai;
        bminspec.in_PPEac_S_1 = spec.getSpecValue( SPEC_in_PPEac_S, 0. );

        if( bminspec.isSelectSPECList ){
           bminspec.in_PPEac_S = Double.parseDouble( bmSpecFR.getSpec( bminspec.m_kt, "NCEleInputMain" ) ) * 1000. *
bminspec.in_dai;//kW->W
```

```
            bminspec.in_PPEac_S_1 = Double.parseDouble( bmSpecFR.getSpec( bminspec.m_kt, "NCEleInputMain" ) ) * 1000.;//kW-
>W
        }

        // 定格消費電力暖房時[W]
        bminspec.in_PPEah_S = spec.getSpecValue( SPEC_in_PPEah_S, 0. ) * bminspec.in_dai;
        bminspec.in_PPEah_S_1 = spec.getSpecValue( SPEC_in_PPEah_S, 0. );

        if( bminspec.isSelectSPECList ){
            bminspec.in_PPEah_S = Double.parseDouble( bmSpecFR.getSpec( bminspec.m_kt, "NHEleInputMain" ) ) * 1000. *
bminspec.in_dai;//kW->W
            bminspec.in_PPEah_S_1 = Double.parseDouble( bmSpecFR.getSpec( bminspec.m_kt, "NHEleInputMain" ) ) * 1000.;//kW-
>W
        }
/*
        // 待機電力(運転時)
        if(null!=map.get(this.SPEC_in_TKEa_R)){
            this.in_TKEa_R = Double.parseDouble((String)map.get(this.SPEC_in_TKEa_R))*in_dai;
        } else {
            System.out.println("(E)SPEC_待機電力(運転時)がありません");
        }

        // 待機電力(停止時)
        if(null!=map.get(this.SPEC_in_THEa_S)){
            this.in_TKEa_S = Double.parseDouble((String)map.get(this.SPEC_in_THEa_S))*in_dai;
        } else {
            System.out.println("(E)SPEC_待機電力(停止時)がありません");
        }
*/
        // 機器起動停止負荷率
        bminspec.in_run_stop = spec.getSpecValue( SPEC_in_run_stop, 0.3 );

        bminspec.in_run_stop_Num = bminspec.in_run_stop;

        // 冷媒配管長
        bminspec.in_Lp = spec.getSpecValue( SPEC_in_Lp, 0. );

        // 冷媒配管高低差
        bminspec.in_Lh = spec.getSpecValue( SPEC_in_Lh, 0. );

        // 定格加湿能力
        bminspec.in_HUM_mx = spec.getSpecValue( SPEC_in_HUM_mx, 0. ) * bminspec.in_dai;
        bminspec.in_HUM_mx_1 = spec.getSpecValue( SPEC_in_HUM_mx, 0. );

//      System.out.println(this.moduleName+" "+name+"  "+in_HUM_mx);

        // 加湿効率
        bminspec.in_HUM_eff = spec.getSpecValue( SPEC_in_HUM_eff, 0.95 );

        // 加湿飽和効率
        bminspec.in_HUM_rt = spec.getSpecValue( SPEC_in_HUM_rt, 0.7 )*100.0;

        // 加湿器ONOFF湿度
        bminspec.in_HUM_on = spec.getSpecValue( SPEC_in_HUM_on, 0.4 )*100.0;

        // 取入外気量
        bminspec.in_OA_S = spec.getSpecValue( SPEC_in_OA_S, 0. ) * bminspec.in_dai;
        bminspec.in_OA_S_1 = spec.getSpecValue( SPEC_in_OA_S, 0. );

        bminspec.isScheduleOA = spec.getSpecValue( SPEC_isScheduleOA, false );

        // 基準外気量
        bminspec.in_OA_100 = spec.getSpecValue( SPEC_in_OA_100, 0. ) * bminspec.in_dai;
        bminspec.in_OA_100_1 = spec.getSpecValue( SPEC_in_OA_100, 0. );

        // 全熱交換器効率
        bminspec.in_EX_S = spec.getSpecValue( SPEC_in_EX_S, 0. );

        //    System.out.println(this.moduleName+" "+name+"  "+in_EX_S);

        //SPEC_isHexbypass = "熱交バイパスあり";
        bminspec.isHexbypass = spec.getSpecValue( SPEC_isHexbypass, false );
```

```
//SPEC_in_EX_PE      = "全熱交換器消費電力[W]";//20130717
bminspec.in_EX_PE = spec.getSpecValue( SPEC_in_EX_PE, 0. ) * bminspec.in_dai;

// 相数
bminspec.phase = spec.getSpecValue( SPEC_Phase, 3 );

// 電圧
bminspec.voltage = spec.getSpecValue( SPEC_Voltage, 200. );

// 周波数
bminspec.frequency = spec.getSpecValue( SPEC_Frequency, 50. );

if( bminspec.isSelectSPECList ){
   if( bmSpecFR.getSpec( bminspec.m_kt, "FrequencyEleIn" ).equals( "50 60" ) ){
      bminspec.frequency = 50;
   }else{
      bminspec.frequency = Double.parseDouble( bmSpecFR.getSpec( bminspec.m_kt, "FrequencyEleIn" ) );
   }
}

// 力率
bminspec.powerFactor = spec.getSpecValue( SPEC_PowerFactor, 0.8 );

//isFanOnOff = "室内機ファン連動";
bminspec.isFanOnOff = spec.getSpecValue( SPEC_isFanOnOff, false );

//SPEC_typeCalcSA = "吹出し状態算定法";
bminspec.typeCalcSA = spec.getSpecValue( SPEC_typeCalcSA, "0_相対湿度を設定" );
if( bminspec.typeCalcSA.equals( "0_相対湿度を設定" )){
   bminspec.numCalcSA = 0;
}else if( bminspec.typeCalcSA.equals( "1_バイパスファクターを設定" )){
   bminspec.numCalcSA = 1;
}else{
   bminspec.numCalcSA = 0;
}

//SPEC_valCalcSA  = "吹出し状態算定法設定値";
bminspec.valCalcSA = spec.getSpecValue( SPEC_valCalcSA, 0.9 );

//isGVisibleを取得
bminspec.isGVisible = spec.getSpecValue( SPEC_isGVisible, false );

//isRecordを取得
bminspec.isRecord = spec.getSpecValue( SPEC_isRecord, false );

//
bminspec.filenames[1] = "EHP_OAProcessingUnit20100226";
//
if( bminspec.m_ty.equals( "0_201303_基本タイプ2018申請" )){//20181018
   bminspec.is2018SinseiOA = true;
}
if( bminspec.m_ty.equals( "1_201303_外気処理エアコン全熱交なし" )){
   bminspec.isOAProcessingUnit = true;
   bminspec.filenames[1] = "EHP_OAProcessingUnit201303";

   bminspec.in_OA_S = bminspec.in_Va_S;//全て外気で送風とする
   bminspec.in_EX_S = 0.;//全熱交の効率=0  全熱交は無いものとする
   bminspec.in_EX_PE = 0;
}
if( bminspec.m_ty.equals( "1_201303_外気処理エアコン全熱交なし2018申請" )){//20181018
   bminspec.isOAProcessingUnit = true;
   bminspec.is2018SinseiOA = true;
   bminspec.filenames[1] = "EHP_OAProcessingUnit201303";

   bminspec.in_OA_S = bminspec.in_Va_S;//全て外気で送風とする
   bminspec.in_EX_S = 0.;//全熱交の効率=0  全熱交は無いものとする
   bminspec.in_EX_PE = 0;
}

if( bminspec.m_ty.equals( "1_外気処理エアコン全熱交なし201002" )){
   bminspec.isOAProcessingUnit = true;
```

```
        bminspec.filenames[1] = "EHP_OAProcessingUnit20100226";

        bminspec.in_OA_S = bminspec.in_Va_S;//全て外気で送風とする
        bminspec.in_EX_S = 0.;//全熱交の効率=0　全熱交は無いものとする
        bminspec.in_EX_PE = 0;
    }
    //"1_外気処理エアコン全熱交なし201002"　の追加すべき特性
    //this.filenames[1] = "EHP_OAProcessingUnit20100226";
    //this.filenames[1] = "EHP_OAProcessingUnit201303";


    //20130327
    if( bminspec.in_Va_S == 0 ){
        //計算するためにはデータ不足である場合
        bminspec.isNeedMoreData = true;
    }

    //isAdjust2012 = "台数を調整する";//
    bminspec.isAdjust2012 = spec.getSpecValue( SPEC_isAdjust2012, false );

    // 調整の計算ステップ数
    bminspec.numAdjustSteps = spec.getSpecValue( SPEC_NumAdjustSteps, 18 );

    //isUseAdjustList = "is調整リストを使用する";//
    bminspec.isUseAdjustList = spec.getSpecValue( SPEC_isUseAdjustList, false );

    bminspec.daiList = new LinkedList<Double>();
    for( int i=0; i<bminspec.numAdjustSteps; i++ ){
        bminspec.daiList.add( 0. );
    }

    if( bminspec.in_Va_S == bminspec.in_OA_S ){
        bminspec.isALLOA = true;
    }else{
        bminspec.isALLOA = false;
    }


    //自動調整
    if( bminspec.isAdjust2012 && bminspec.isUseAdjustList ){
        bminspec.in_Qc_S    = bminspec.adjustQcListkW[0] * 1000. * bminspec.in_daiAdjust;//[kW]->[W]
        bminspec.in_Qh_S    = bminspec.adjustQhListkW[0] * 1000. * bminspec.in_daiAdjust;//[kW]->[W]
        //bminspec.out_QhL_S   = 0 * this.out_daiAdjust;
        bminspec.in_PPEac_S = bminspec.adjustPPEcListkW[0] * 1000. * bminspec.in_daiAdjust;//[kW]->[W]
        bminspec.in_PPEah_S = bminspec.adjustPPEhListkW[0] * 1000. * bminspec.in_daiAdjust;//[kW]->[W]

        bminspec.in_Va_S    = bminspec.adjustfRateListm3min[0] * 60 / 3 * bminspec.in_daiAdjust;//[m3/min]->[g/s]

        bminspec.maxAdjustdai = 1.;
    }

}


//@Override
public static void initialize( Map<String, String> mapState, AbstractBestModule abs, IBestStateMessage stateNodes,
        IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
        IBestDomainSchedule schedule, BMinSpec2015 bminspec ) {

    //状態ノードを受け取る
// super.sm = stateNodes;
    //制御ノードを受け取る
// super.cm = commandNodes;
    //記録ノードを受け取る
// super.rm = recordNodes;

    //接続ノード　出口

    //airOutRM
    bminspec.airOutRM = BestAir.bindnode( mapState, stateNodes, S_NODE_airOutRM );
    bminspec.airOutRM.setMaxFlowRate( bminspec.in_Va_S );//20150423
    if( bminspec.isUseZoneAir ){
```

```
            //this.airOutRM = new BestAir();
        // this.airOutRM.setTempDB(24.0);
        // this.airOutRM.setHumi(0.01);
        // this.airOutRM.setFlowRate(0);
            bminspec.zoneAirforSSinside._initialize( );

        }else{
        }

        //watInCW
        bminspec.watInCW = BestWater.bindnode( mapState, stateNodes, S_NODE_watInCW );

        //eleIn
        bminspec.eleIn = BestElectricity.bindnode( mapState, stateNodes, S_NODE_eleIn );

        //watOutD
        bminspec.watOutD = BestWater.bindnode( mapState, stateNodes, S_NODE_watOutD );


        //接続ノード　入口
        //airInRM
        if( bminspec.isUseZoneAir ){
            bminspec.zoneAirforSSinside.checkNumberAirChange( bminspec.in_Va_S, "in_Va_S", bminspec.moduleName,
"initialize()", bminspec.name, bminspec.isRecord, bminspec.message);

            bminspec.airInRM = new BestAir();

        }else{
            bminspec.airInRM = BestAir.bindnode( mapState, stateNodes, S_NODE_airInRM );
        // this.airInRM.setTempDB(24.0);
        // this.airInRM.setHumi(0.01);
        // this.airInRM.setFlowRate(in_Va_S);

        }
        bminspec.airInRM.setMaxFlowRate( bminspec.in_Va_S );//20150423

        //S_NODE_valInPID
        bminspec.PIDrate = BestValue.bindnode( mapState, stateNodes, S_NODE_valInPID );
        bminspec.PIDrate.setValue( 1. );

        //S_NODE_valInOARate = "LO_valInOARate";//外気流量率
        bminspec.valInOARate = BestValue.bindnode( mapState, stateNodes, S_NODE_valInOARate );

//      冷媒配管を室外機に渡す
        if( stateNodes.getState( abs.getConnectionNode( S_NODE_valOutLine)) != null ){

            //System.out.println( " BM_in S_NODE_valOutLine登録済み" );

            bminspec.rmlinemap= (Map<String, BM_EHPdata>) stateNodes.getState( abs.getConnectionNode( S_NODE_valOutLine));

            if( bminspec.rmlinemap.size() == 0 ){
                //System.out.println( "BM_in  Parent 未登録" );
            }else{
                //System.out.println( "BM_in  Parent path="+ this.rmlinemap.get( "Parent" ).get_path() );
            }

        }else{

            //System.out.println( " BM_in S_NODE_valOutLine未登録" );

            //System.out.println( this.moduleName + ">>Warning<<  rmlinemap is null !!" );

            bminspec.message.append( "(W)rmlinemap接続なし→作成");
            bminspec.rmlinemap = new HashMap<String, BM_EHPdata>();
            stateNodes.setState( abs.getConnectionNode( S_NODE_valOutLine), bminspec.rmlinemap);
        }

        //個々の室内機のデータは　ここで登録する
        bminspec.RMdata = new BM_EHPdata();

        bminspec.RMdata.set_in_Qc_S( bminspec.in_Qc_S );
        bminspec.RMdata.set_in_Qh_S( bminspec.in_Qh_S );
```

```
        bminspec.RMdata.set_in_dai( (double)bminspec.in_dai );
        //bminspec.RMdata.set_kiki("");
        bminspec.RMdata.set_L_rate(1.0);
        bminspec.RMdata.set_R_wb(20.0);
        bminspec.RMdata.set_R_db(24.0);
        bminspec.RMdata.set_R_S_load( bminspec.in_Qc_S * 0.9 );
        bminspec.RMdata.set_R_T_load( bminspec.in_Qc_S);
        bminspec.RMdata.set_R_Sc_load( bminspec.in_Qc_S * 0.9 );
        bminspec.RMdata.set_R_Tc_load( bminspec.in_Qc_S);
        bminspec.RMdata.set_R_Sh_load( bminspec.in_Qh_S);
        bminspec.RMdata.set_R_Th_load( bminspec.in_Qh_S);
        bminspec.RMdata.set_CAPrate(1.0);
        bminspec.rmlinemap.put( bminspec.name, bminspec.RMdata );
        stateNodes.setState( abs.getConnectionNode( S_NODE_valOutLine),bminspec.rmlinemap);


        //接続ノード　入口
        bminspec.airInOA = BestAir.bindnode( mapState, stateNodes, S_NODE_airInOA );
        bminspec.airInOA.setMaxFlowRate( bminspec.in_OA_S );//20150423


        //グラフ表示の準備
        if( bminspec.isGVisible ){
           bminspec.gBMIn = new GraphJFrameBuilMultiIn_S20101212( bminspec.name, 300, bminspec.in_Qc_S * 1.5,
bminspec.in_Va_S * 1.5 );
        }

    }

    /**
     * ビルマルチ計算及び結果出力/室外機が先に動いています。
     */

    //@Override
    public static void outputs_calc( AbstractBestModule abs, IBestStateMessage stateNodes,
          IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
          BMinSpec2015 bminspec) {

        if( !bminspec.isCalc ){
           return;
        }

        if (null == commandNodes || null == stateNodes )
           return;

        //message.setLength(0);
        //ノード接続毎回読込み

        bminspec.rmlinemap = (Map<String, BM_EHPdata>) stateNodes.getState( abs.getConnectionNode( S_NODE_valOutLine));
        bminspec.RMdata = bminspec.rmlinemap.get( bminspec.name );
        if( bminspec.dbflg == 0 ){
           //this.filenames[0]  = this.RMdata.get_kiki();
           //this.equipmentName = this.RMdata.get_kiki();
           //this.path = this.RMdata.get_path();
           //System.out.println( " RMdata = " + this.rmlinemap.entrySet() );

           if( null == bminspec.rmlinemap.get( "Parent" ) ){//20161028
              BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                    SystemMessageConstants.CAL_ERROR2 ,
                    new String[]{ "BMin 室外機がない:" +bminspec.name+" ("+bminspec.moduleName+")",
                     AirSystemControl.getTimeStr() } ));
//            BestLogger.info( "(E)WBin 入口空気WBが異常値(DBin="+DBin+" XGin="+XGin+")"
//                  + "__"+this.name+"__("+this.moduleName+"_cooling_cal_WBin)" + AirSystemControl.getTimeStr());
//            if( this.isRecord ){
//               this.message.append( "(W)WBin 入口空気WBが異常値");
//            }
           }
           //System.out.println( " Parent path="+ bminspec.rmlinemap.get( "Parent" ).get_path() );

           bminspec.filenames[0]  = bminspec.rmlinemap.get( "Parent" ).get_kiki();
           bminspec.equipmentName = bminspec.rmlinemap.get( "Parent" ).get_kiki();
           bminspec.path = bminspec.rmlinemap.get( "Parent" ).get_path();
```

```
        //System.out.println( "filenames  "+bminspec.filenames[0]+" /equipmentName "+bminspec.equipmentName +"
/path="+bminspec.path);
        bminspec.pacDB = new PACDBManager( bminspec.path, bminspec.filenames, bminspec.RMdata.isUserTokusei() );
        bminspec.pacDB.setEquipmentName( bminspec.equipmentName );

        //COOLING
        if( bminspec.RMdata.isWaterCooled() ){
           bminspec.maxTwatIn_C_FACKctw = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKctw() );
           bminspec.minTwatIn_C_FACKctw = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKctw() ) ;
        }else if( bminspec.RMdata.isHighSH() ){
           bminspec.maxTdbOA_C_FACKctai = bminspec.pacDB.getformulaRangeMaxRaw( bminspec.pacDB.getFACKctai() );
           bminspec.minTdbOA_C_FACKctai = bminspec.pacDB.getformulaRangeMinRaw( bminspec.pacDB.getFACKctai() );
        }else{
           bminspec.maxTdbOA_C_FACKcta = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKcta() );
           bminspec.minTdbOA_C_FACKcta = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKcta() );
        }
        if( bminspec.RMdata.isHighSH() ){
           bminspec.maxTwbIn_C_FACKctai = bminspec.pacDB.getformulaRangeMaxCol( bminspec.pacDB.getFACKctai() );
           bminspec.minTwbIn_C_FACKctai = bminspec.pacDB.getformulaRangeMinCol( bminspec.pacDB.getFACKctai() );
        }else{
           bminspec.maxTwbIn_C_FACKcti = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKcti() );
           bminspec.minTwbIn_C_FACKcti = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKcti() );
        }
        if( !bminspec.isOAProcessingUnit && bminspec.RMdata.isWaterCooled() ){
        }else if( !bminspec.isOAProcessingUnit ){
        }else{
           bminspec.pacDB.setEquipmentName( bminspec.filenames[1] );
           bminspec.maxTdbOA_C_FACKctdb = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKctdb() );
           bminspec.minTdbOA_C_FACKctdb = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKctdb() );
           bminspec.maxTwbOA_C_FACKctwb = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKctwb() );
           bminspec.minTwbOA_C_FACKctwb = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKctwb() );
           bminspec.pacDB.setEquipmentName( bminspec.filenames[0] );

        }
        //HEATING
        if( bminspec.RMdata.isWaterCooled() ){
        bminspec.maxTwatIn_H_FACKhtw = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKhtw() );
        bminspec.minTwatIn_H_FACKhtw = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKhtw() ) ;
        }else if( bminspec.RMdata.isHighSH() ){
           bminspec.maxTwbOA_H_FACKhtai = bminspec.pacDB.getformulaRangeMaxRaw( bminspec.pacDB.getFACKhtai() );
           bminspec.minTwbOA_H_FACKhtai = bminspec.pacDB.getformulaRangeMinRaw( bminspec.pacDB.getFACKhtai() );
        }else{
           bminspec.maxTwbOA_H_FACKhta = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKhta() );
           bminspec.minTwbOA_H_FACKhta = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKhta() );
        }
        if( bminspec.RMdata.isHighSH() ){
           bminspec.maxTdbIn_H_FACKhtai = bminspec.pacDB.getformulaRangeMaxCol( bminspec.pacDB.getFACKhtai() );
           bminspec.minTdbIn_H_FACKhtai = bminspec.pacDB.getformulaRangeMinCol( bminspec.pacDB.getFACKhtai() );
        }else{
           bminspec.maxTdbIn_H_FACKhti = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKhti() );
           bminspec.minTdbIn_H_FACKhti = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKhti() );
        }
        if( !bminspec.isOAProcessingUnit && bminspec.RMdata.isWaterCooled() ){
        }else if( !bminspec.isOAProcessingUnit ){
        }else{
           bminspec.pacDB.setEquipmentName( bminspec.filenames[1] );
           bminspec.maxTdbOA_H_FACKhtdb = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKhtdb() );
           bminspec.minTdbOA_H_FACKhtdb = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKhtdb() );
           bminspec.maxTwbOA_H_FACKhtwb = bminspec.pacDB.getformulaRangeMax( bminspec.pacDB.getFACKhtwb() );
           bminspec.minTwbOA_H_FACKhtwb = bminspec.pacDB.getformulaRangeMin( bminspec.pacDB.getFACKhtwb() );
           bminspec.pacDB.setEquipmentName( bminspec.filenames[0] );
        }

     }
     bminspec.dbflg = 1;

     if( bminspec.isUseZoneAir ){
        bminspec.zoneAirforSSinside._outputSetRA( bminspec.airInRM );
     }else{
        bminspec.airInRM = (BestAir) stateNodes.getState( abs.getConnectionNode( S_NODE_airInRM));
     }
```

```
bminspec.airInOA = (BestAir) stateNodes.getState( abs.getConnectionNode( S_NODE_airInOA));

bminspec.swcInOA  = commandNodes.getCommand( abs.getConnectionNode( C_NODE_swcInOA) );
bminspec.swcIn    = commandNodes.getCommand( abs.getConnectionNode( C_NODE_swcIn) );
bminspec.modIn    = commandNodes.getCommand( abs.getConnectionNode( C_NODE_modIn));
bminspec.modInPID = commandNodes.getCommand( abs.getConnectionNode( C_NODE_modInPID));

bminspec.PIDrate = (BestValue) stateNodes.getState( abs.getConnectionNode( S_NODE_valInPID));

bminspec.qSTHEX = 0;
bminspec.qLTHEX = 0;
bminspec.qTTHEX = 0;


if( bminspec.isNeedMoreData ){
   //停止
   bminspec.mState = 0;
   if( bminspec.isRecord ){
      bminspec.message.append( "(C)入力情報不足停止");
   }
}else if( Airswc.isOFF( bminspec.swcIn )){
   //停止
   bminspec.mState = 0;
   if( bminspec.isRecord ){
      bminspec.message.append( "(C)停止");
   }
}else if( bminspec.isALLOA && Airswc.isOFF( bminspec.swcInOA )){//20150409
   //全外気の場合は  OAの信号でon/offする
   bminspec.mState = 0;
   if( bminspec.isRecord ){
      bminspec.message.append( "(C)停止");
   }
}else if( Airswc.isON( bminspec.swcIn )){
   //運転
   if( bminspec.isFanOnOff && bminspec.PIDrate.getValue() < bminspec.in_run_stop ){//20141008
      //停止
      bminspec.mState = 0;
      if( bminspec.isRecord ){
         bminspec.message.append( "(C)停止");
      }
   }else if( bminspec.isFanOnOff && bminspec.PIDrate.getValue() == 0 ){//20141008
      //停止
      bminspec.mState = 0;
      if( bminspec.isRecord ){
         bminspec.message.append( "(C)停止");
      }
   }else

   if( Airmod.isCOOL( bminspec.modIn ) && Airmod.isHEAT( bminspec.modIn ) ){
      //冷暖同時
      if( bminspec.RMdata.isCOOLandHEAT() ){
         if( Airmod.isCOOL( bminspec.modInPID ) ){
            bminspec.mState = 1;
            if( bminspec.isRecord ){
               bminspec.message.append( "(C)PID冷暖同時の冷房");
            }
         }else if(  Airmod.isHEAT( bminspec.modInPID ) ){
            bminspec.mState = 2;
            if( bminspec.isRecord ){
               bminspec.message.append( "(C)PID冷暖同時の暖房");
            }
         }else{
            bminspec.mState = 1;
         }
      }else{
         //冷暖同時機種で無い場合は冷房運転とする
         bminspec.mState = 1;
         if( bminspec.isRecord ){
            bminspec.message.append( "(W)冷暖同時機種ではないので冷房運転で計算" );
         }
      }
```

```
        }else if( Airmod.isCOOL( bminspec.modIn )){
           //冷房運転
           bminspec.mState = 1;
           if( bminspec.isRecord ){
              bminspec.message.append( "(C)冷房");
           }
        }else if( Airmod.isHEAT( bminspec.modIn )){
           //暖房運転
           bminspec.mState = 2;
           if( bminspec.isRecord ){
              bminspec.message.append( "(C)暖房");
           }
        }else if( Airmod.isVENTILATE( bminspec.modIn )){
           //換気モード
           bminspec.mState = 3;
           if( bminspec.isRecord ){
              bminspec.message.append( "(C)換気");
           }
        }else{
           //停止
           bminspec.mState = 0;
           if( bminspec.isRecord ){
              bminspec.message.append( "(C)停止");
           }
        }
     }

     /**
      * 入力
      */
     if( bminspec.mState > 0 ){
        //冷房、暖房、換気の時
        BestAir.checkOpeData( bminspec.airInRM, "airInRM", bminspec.moduleName, "outputs", bminspec.name,
bminspec.isRecord, bminspec.message );

     }

     //状態ノードから室内空気を設定
     //室内還気乾球温度を設定
     bminspec.DBra = bminspec.airInRM.getTempDB();//20130109
     //室内還気絶対湿度を設定
     bminspec.XGra = bminspec.airInRM.getHumi();//20130109
     //室内還気湿球温度を設定

  //20150306
  // if( Double.isInfinite( this.DBra ) || Double.isNaN( this.DBra )){
  //    if( this.isRecord ){
  //       this.message.append( "(E)DBra異常値→0℃に設定");
  //    }
  //    this.mState= -1;
  //    this.DBra = 0;
  // }
  // if( Double.isInfinite( this.XGra ) || Double.isNaN( this.XGra ) || this.XGra < 0 ){
  //    if( this.isRecord ){
  //       this.message.append( "(E)XGra異常値→0.004g/gに設定");
  //    }
  //    this.mState= -1;
  //    this.XGra = 0.000;
  // }

     bminspec.WBra = Psychrometrics.FNWbtx( bminspec.DBra, bminspec.XGra );//20130109

           //System.out.println("this name="+this.name+"DBra="+DBra+" "+XGra+"  "+WBra+"  ");


     //外気乾球温度設定
     bminspec.DBoa = bminspec.airInOA.getTempDB();
     //外気絶対湿度を設定
     bminspec.XGoa = bminspec.airInOA.getHumi();
     //外気湿球温度を設定
     bminspec.WBoa = Psychrometrics.FNWbtx( bminspec.DBoa, bminspec.XGoa );
     if( bminspec.WBoa < -9000.0 ){
```

```
            bminspec.WBoa = WB_cal( bminspec.DBoa, bminspec.XGoa );
        }

        //**************************----------------
        //**************************----------------

        // System.out.println( "BMin [ "+this.name+" ] in_daiAdjust=" + in_daiAdjust);

        //20141113
        if( bminspec.isScheduleOA ){
            bminspec.Moa = bminspec.in_OA_100 * bminspec.valInOARate.getValue();
        }else{
            bminspec.Moa = bminspec.in_OA_S;
        }

        //CO2ppm//20131029
        //外気取入時のCO2ppm
        double co2ppm = bminspec.airInRM.getCO2ppm() * ( bminspec.in_Va_S - bminspec.Moa ) + bminspec.airInOA.getCO2ppm() *
bminspec.Moa;
        if( bminspec.in_Va_S != 0 ){
            co2ppm /= bminspec.in_Va_S;
        }else{
            co2ppm = bminspec.airInRM.getCO2ppm();
        }
        bminspec.airOutRM.setCO2ppm( co2ppm );


        switch( bminspec.mState ){
        case -1:
            //異常停止
            if( bminspec.isRecord ){
                bminspec.message.append("(C)異常停止");
            }
            calc_Stop1( bminspec );
            break;
        case 0:
            //停止
            if( bminspec.isRecord ){
                bminspec.message.append("(C)停止");
            }
            calc_Stop1( bminspec );
            break;
        case 1:
            //冷房
            if( bminspec.in_Qc_S_1 > 0 ){
                if( bminspec.isRecord ){
                    bminspec.message.append("(C)冷房運転");
                }
                if( bminspec.isFanOnOff ){//20141009 BESTEST
                    //this.GW = this.in_Va_S * this.PIDrate.getValue();
                    bminspec.GW = bminspec.in_Va_S;
                }else{
                    bminspec.GW = bminspec.in_Va_S;
                }

                cooling_cal( bminspec );

                if( bminspec.isFanOnOff ){
                    bminspec.PPE = bminspec.in_PPEac_S * bminspec.PIDrate.getValue() + bminspec.in_TKEa_R +
bminspec.in_EX_PE;//20130717
                }else{
                    bminspec.PPE = bminspec.in_PPEac_S + bminspec.in_TKEa_R + bminspec.in_EX_PE;//20130717
                }
            }else{
                if( bminspec.isRecord ){
                    bminspec.message.append("(C)冷能力=0停止");//20130625
                }
                calc_Stop1( bminspec );
                bminspec.mState = 0;
            }
            break;
        case 2:
```

```
        //暖房
        if( bminspec.in_Qh_S_1 > 0 ){
            if( bminspec.isRecord ){
                bminspec.message.append("(C)暖房運転");
            }
            heating_cal( bminspec );
            bminspec.PPE = bminspec.in_PPEah_S + bminspec.in_TKEa_R + bminspec.in_EX_PE;//20130717
        }else{
            if( bminspec.isRecord ){
                bminspec.message.append("(C)暖能力=0停止");//20130625
            }
            calc_Stop1( bminspec );
            bminspec.mState = 0;
        }
        break;
    case 3:
        //換気
        if( bminspec.isRecord ){
            bminspec.message.append("(C)換気運転");
        }
        ventilation_cal( bminspec );
        bminspec.PPE = bminspec.in_PPEah_S + bminspec.in_TKEa_R + bminspec.in_EX_PE;//20130717
        break;
    default:
        if( bminspec.isRecord ){
            bminspec.message.append("(C)運転モード？停止");//20130625
        }
        calc_Stop1( bminspec );
        bminspec.mState = 0;
        //System.out.println(this.moduleName + ">>Error<< onOff*modeが範囲外");
    }

    bminspec.E_PPE = bminspec.PPE * Math.pow( 1/bminspec.powerFactor/ bminspec.powerFactor - 1, 0.5 );
    bminspec.eleIn.setActivePower( bminspec.PPE);
    bminspec.eleIn.setReactivePower( bminspec.E_PPE );
    bminspec.eleIn.setPhase( bminspec.phase );
    bminspec.eleIn.setVoltage( bminspec.voltage );
    bminspec.eleIn.setFrequency( bminspec.frequency );

    //    熱量を室外機に渡す
    bminspec.rmlinemap= (Map<String,BM_EHPdata>) stateNodes.getState( abs.getConnectionNode( S_NODE_valOutLine));

    bminspec.RMdata = bminspec.rmlinemap.get( bminspec.name );

    bminspec.RMdata.set_L_rate( bminspec.LPrate );
    //20181018誘導基準申請計算室内温度特性
    if( bminspec.is2018SinseiOA ) {
        bminspec.RMdata.set_R_wb( bminspec.WBra );//20130109
        bminspec.RMdata.set_R_db( bminspec.DBra );//20130109
    }else {
        bminspec.RMdata.set_R_wb( bminspec.WBin );//20130109
        bminspec.RMdata.set_R_db( bminspec.DBin );//20130109
    }
    bminspec.RMdata.set_R_S_load( bminspec.S_Load );
    bminspec.RMdata.set_R_T_load( bminspec.T_Load );
    bminspec.RMdata.set_R_Sc_load( bminspec.Sc_Load );
    bminspec.RMdata.set_R_Tc_load( bminspec.Tc_Load );
    bminspec.RMdata.set_R_Sh_load( bminspec.Sh_Load );
    bminspec.RMdata.set_R_Th_load( bminspec.Th_Load );
    bminspec.rmlinemap.put( bminspec.name, bminspec.RMdata );
    stateNodes.setState( abs.getConnectionNode( S_NODE_valOutLine), bminspec.rmlinemap );

    // System.out.println( "BMin this.Tc_Load"+this.Tc_Load + "BMin this.Th_Load"+this.Th_Load );

    //System.out.println( "Out GW="+GW+" FlowRateairOutRM="+this.airOutRM.getFlowRate()+ " mState="+this.mState );
}


private static void calc_Stop1( BMinSpec2015 bminspec ){
    bminspec.S_Load = 0.0;
    bminspec.T_Load = 0.0;
    bminspec.Sc_Load = 0.0;
```

```
        bminspec.Tc_Load = 0.0;
        bminspec.Sh_Load = 0.0;
        bminspec.Th_Load = 0.0;
        bminspec.LPrate = 1.0;
        bminspec.PPE = bminspec.in_TKEa_S;
        if( bminspec.isOAProcessingUnit ){//20150319
            bminspec.DBin = bminspec.DBoa;
            bminspec.XGin = bminspec.XGoa;
            bminspec.WBin = bminspec.WBoa;
        }else{
            bminspec.DBin = bminspec.DBra;
            bminspec.XGin = bminspec.XGra;
            bminspec.WBin = bminspec.WBra;
        }

        //bminspec.airOutRM.copyAllState( bminspec.airInRM );
        //bminspec.airOutRM.setFlowRate( 0. );
        bminspec.GW = 0;
        bminspec.Moa = 0.;
    }

    /**
     * ビルマルチ計算/室外機から補正比率を受け取ってから後の処理
     *
     */

    //@Override
    public static void update( AbstractBestModule abs, IBestStateMessage stateNodes,
            IBestControlMessage commandNodes, IBestRecordMessage recordNodes, BMinSpec2015 bminspec ) {

        if( !bminspec.isCalc ){
            return;
        }

        bminspec.rmlinemap
        = (Map<String,BM_EHPdata>) stateNodes.getState( abs.getConnectionNode( S_NODE_valOutLine));
        bminspec.RMdata = bminspec.rmlinemap.get( bminspec.name );
        //this.CAPrate = this.RMdata.get_CAPrate();
        bminspec.CAPrateC = bminspec.RMdata.get_CAPrateC();
        bminspec.CAPrateH = bminspec.RMdata.get_CAPrateH();
//      this.CAPrate = (Double)super.sm.getState( super.getConnectionNode( this.S_NODE_valOutHosei));
        //this.S_Load_RM = this.CAPrate * this.S_Load;
        //this.T_Load_RM = this.CAPrate * this.T_Load;
        bminspec.Sc_Load_RM = bminspec.CAPrateC * bminspec.Sc_Load;
        bminspec.Tc_Load_RM = bminspec.CAPrateC * bminspec.Tc_Load;
        bminspec.Sh_Load_RM = bminspec.CAPrateH * bminspec.Sh_Load;
        bminspec.Th_Load_RM = bminspec.CAPrateH * bminspec.Th_Load;

        //    System.out.println("update() CAPrateC="+CAPrateC+" /Sc_Load_RM="+Sc_Load_RM+" /Sc_Load"+Sc_Load+ " "+mState);

        //    System.out.println("update() CAPrateH="+CAPrateH+" /Sh_Load_RM="+Sh_Load_RM+" /Sh_Load"+Sh_Load+ " "+mState);


        switch( bminspec.mState ){
        case -1:
            //異常停止
            bminspec.DBout = bminspec.DBin;//乾球温度
            bminspec.XGout = bminspec.XGin;//絶対湿度
            bminspec.GW = 0.0;
            bminspec.Moa = 0.0;//20150409
            bminspec.D_Wrate = 0.0;
            bminspec.CW_Wrate = 0.0;
            bminspec.S_Load_RM = 0.0;
            bminspec.T_Load_RM = 0.0;
            //
            if( bminspec.isRecord ){
                bminspec.message.append("(C)異常処理");
            }
            bminspec.PIDrate.setValue( 0. );
            break;
        case 0:
            //停止
```

```
    bminspec.DBout = bminspec.DBin;//乾球温度
    bminspec.XGout = bminspec.XGin;//絶対湿度
    bminspec.GW = 0.0;
    bminspec.Moa = 0.0;//20150409
    bminspec.D_Wrate = 0.0;
    bminspec.CW_Wrate = 0.0;
    bminspec.S_Load_RM = 0.0;
    bminspec.T_Load_RM = 0.0;
    //
    if( bminspec.isRecord ){
       bminspec.message.append("(C)停止処理");
    }
    bminspec.PIDrate.setValue( 0. );
    break;
 case 1:
    //冷房
    //if( this.isFanOnOff ){
    // this.GW = this.in_Va_S * this.PIDrate.getValue();
    //}else{
    // this.GW = this.in_Va_S;
    //}
    if( Airswc.isON( bminspec.swcInOA )){//20150409
       bminspec.Moa = bminspec.in_OA_S;
    }else{
       bminspec.Moa = 0;
    }
    cooling_supply( bminspec );
    bminspec.D_Wrate = ( bminspec.XGra - bminspec.XGout ) * bminspec.GW;//g/s
    if( bminspec.D_Wrate < 0.0 ){
       bminspec.D_Wrate = 0.0;//追加090619
    }
    bminspec.CW_Wrate = 0.0;
    bminspec.S_Load_RM = bminspec.Sc_Load_RM;
    bminspec.T_Load_RM = bminspec.Tc_Load_RM;
    if( bminspec.CAPrateC == 0 ){
       bminspec.PIDrate.setValue( 0. );
    }
    if( bminspec.isRecord ){
       bminspec.message.append("(C)冷房処理PID="+AirFormat.df_2( bminspec.PIDrate.getValue()));
    }
    break;
 case 2:
    //暖房
    bminspec.GW = bminspec.in_Va_S;
    if( Airswc.isON( bminspec.swcInOA )){//20150409
       bminspec.Moa = bminspec.in_OA_S;
    }else{
       bminspec.Moa = 0;
    }
    heating_supply( bminspec );
    bminspec.D_Wrate = 0.0;
    bminspec.S_Load_RM = bminspec.Sh_Load_RM;
    bminspec.T_Load_RM = bminspec.Th_Load_RM;
    if( bminspec.CAPrateH == 0 ){
       bminspec.PIDrate.setValue( 0. );
    }
    if( bminspec.isRecord ){
       bminspec.message.append("(C)暖房処理PID="+AirFormat.df_2( bminspec.PIDrate.getValue()));
    }
    break;
 case 3:
    //換気
    bminspec.GW = bminspec.in_Va_S;
    if( Airswc.isON( bminspec.swcInOA )){//20150409
       bminspec.Moa = bminspec.in_OA_S;
    }else{
       bminspec.Moa = 0;
    }
    ventilation_supply( bminspec );
    bminspec.D_Wrate = 0.0;
    bminspec.S_Load_RM = 0.0;
    bminspec.T_Load_RM = 0.0;
```

```
            //
            bminspec.PIDrate.setValue( 0. );
            if( bminspec.isRecord ){
               bminspec.message.append("(C)換気処理");
            }
            break;
         default:
            System.out.println( bminspec.moduleName + ">>Error<< onOff*modeが範囲外");
         }
         //    System.out.println("DBout="+DBout);

         bminspec.airOutRM.setTempDB( bminspec.DBout );
         bminspec.airOutRM.setHumi( bminspec.XGout );
         bminspec.airOutRM.setFlowRate( bminspec.GW );

         if( bminspec.isUseZoneAir ){
            bminspec.zoneAirforSSinside._update( bminspec.airOutRM );
         }

         bminspec.watOutD.setFlowRate( bminspec.D_Wrate );
         bminspec.watOutD.setTemp( bminspec.DBout );
         bminspec.watInCW.setFlowRate( bminspec.CW_Wrate );
         //    this.airInRM.setFlowRate(GW);
         //    System.out.println("DBout="+DBout+" "+XGout+"  "+airOutRM.getFlowRate());

         //出力側ノード
         stateNodes.setState( abs.getConnectionNode( S_NODE_airOutRM ), bminspec.airOutRM);
         stateNodes.setState( abs.getConnectionNode( S_NODE_eleIn ), bminspec.eleIn);
         stateNodes.setState( abs.getConnectionNode( S_NODE_watOutD ), bminspec.watOutD);
//       stateNodes.setState( abs.getConnectionNode( S_NODE_valOutHosei ), bminspec.CAPrate);
         stateNodes.setState( abs.getConnectionNode( S_NODE_valOutLine ), bminspec.rmlinemap);
         stateNodes.setState( abs.getConnectionNode( S_NODE_watInCW ), bminspec.watInCW);//090407追加



         //グラフデータ追加
         if( bminspec.isGVisible ){
            bminspec.gBMIn.addData( BestTimeManager.getDateWeatherTime(),
                  bminspec.Sc_Load_RM,
                  bminspec.Tc_Load_RM,
                  bminspec.Sh_Load_RM,
                  bminspec.Th_Load_RM,
                  bminspec.PPE,
                  bminspec.DBoa,
                  bminspec.DBra,
                  bminspec.DBin,
                  bminspec.DBout,
                  bminspec.XGoa,
                  bminspec.XGra,
                  bminspec.XGin,
                  bminspec.XGout,
                  bminspec.GW,
                  bminspec.D_Wrate );
         }

         //記録ノード
         if( bminspec.isRecord && recordNodes != null ){
            record( abs, recordNodes, bminspec );
         }

         bminspec.message.setLength(0);

         //*************************----------------
         if( bminspec.isAdjust2012 ){
            //移動平均の最大値で調整していく
            bminspec.daiList.removeLast();
            if( bminspec.isUseAdjustList ){
               bminspec.daiList.addFirst( bminspec.PIDrate.getValue() );
            }else{
               bminspec.daiList.addFirst( bminspec.in_daiAdjust * bminspec.PIDrate.getValue() );
            }
            //
```

```
            double sum_in_dai = 0;
            for( int i=0; i<bminspec.numAdjustSteps; i++ ){
               sum_in_dai += bminspec.daiList.get( i );
            }
            bminspec.avedaiAdjust = sum_in_dai / bminspec.numAdjustSteps;
            //
            if( sum_in_dai > bminspec.maxAdjustdai * bminspec.numAdjustSteps && !bminspec.isUseAdjustList ){
               //従来の調整
               bminspec.maxAdjustdai = sum_in_dai / bminspec.numAdjustSteps;

               //System.out.println(" maxAdjustdai="+bminspec.maxAdjustdai);

               //室内機の最大調整台数は１台とする20140624
               if( bminspec.maxAdjustdai - bminspec.in_daiAdjust > 1 ){
                  bminspec.in_daiAdjust += 1.;
               }else{
                  bminspec.in_daiAdjust = bminspec.maxAdjustdai;
               }
               bminspec.in_run_stop_Num = bminspec.in_run_stop / bminspec.in_daiAdjust;

               bminspec.in_Qc_S    = bminspec.in_Qc_S_1 * bminspec.in_daiAdjust;
               bminspec.in_Qh_S    = bminspec.in_Qh_S_1 * bminspec.in_daiAdjust;
               bminspec.in_QhL_S   = bminspec.in_QhL_S_1 * bminspec.in_daiAdjust;
               bminspec.in_Va_S    = bminspec.in_Va_S_1 * bminspec.in_daiAdjust;
               bminspec.in_PPEac_S = bminspec.in_PPEac_S_1 * bminspec.in_daiAdjust;
               bminspec.in_PPEah_S = bminspec.in_PPEah_S_1 * bminspec.in_daiAdjust;
               bminspec.in_HUM_mx  = bminspec.in_HUM_mx_1 * bminspec.in_daiAdjust;

               bminspec.airInRM.setFlowRate(bminspec.in_Va_S);
               bminspec.airOutRM.setFlowRate(bminspec.in_Va_S);

            }else if( bminspec.avedaiAdjust >= bminspec.maxAdjustdai && bminspec.isUseAdjustList ){
               //リストで調整するとき
               int start = bminspec.numList;

               //System.out.println("(useAdjustList) maxAdjustdai="+bminspec.maxAdjustdai+"
avedaiAdjust="+bminspec.avedaiAdjust + " in_daiAdjust="+bminspec.in_daiAdjust + " in_Qc_S="+bminspec.in_Qc_S );

               if( bminspec.adjustQcListkW[ bminspec.adjustQcListkW.length - 1 ] * 1000. * bminspec.in_daiAdjust<
                     bminspec.in_Qc_S * bminspec.avedaiAdjust ){
                  bminspec.in_daiAdjust += 1.;
                  start = 0;
               }

               int n=0;
               double inQ = bminspec.in_Qc_S;
               for( n=start; n<bminspec.adjustQcListkW.length; n++ ){

                  if( bminspec.adjustQcListkW[n] * 1000. * bminspec.in_daiAdjust
                        > bminspec.in_Qc_S ){

                     bminspec.in_Qc_S    = bminspec.adjustQcListkW[n] * 1000. * bminspec.in_daiAdjust;//[kW]->[W]
                     bminspec.in_Qh_S    = bminspec.adjustQhListkW[n] * 1000. * bminspec.in_daiAdjust;//[kW]->[W]
                     bminspec.in_QhL_S   = 0 * bminspec.in_daiAdjust;
                     bminspec.in_Va_S    = bminspec.adjustfRateListm3min[n] * 60 / 3 * bminspec.in_daiAdjust;//[m3/min]-
>[g/s]
                     bminspec.in_PPEac_S = bminspec.adjustPPEcListkW[n] * 1000. * bminspec.in_daiAdjust;//[kW]->[W]
                     bminspec.in_PPEah_S = bminspec.adjustPPEhListkW[n] * 1000. * bminspec.in_daiAdjust;//[kW]->[W]
                     bminspec.in_HUM_mx  = bminspec.in_HUM_mx_1 * bminspec.in_daiAdjust;

                     bminspec.airInRM.setFlowRate(bminspec.in_Va_S);
                     bminspec.airOutRM.setFlowRate(bminspec.in_Va_S);

                  // bminspec.in_Qc_S_1    = bminspec.in_Qc_S / bminspec.in_daiAdjust;
                  // bminspec.in_Qh_S_1    = bminspec.in_Qh_S / bminspec.in_daiAdjust;
                  // bminspec.in_QhL_S_1   = bminspec.in_QhL_S / bminspec.in_daiAdjust;
                  // bminspec.in_Va_S_1    = bminspec.in_Va_S / bminspec.in_daiAdjust;
                  // bminspec.in_PPEac_S_1 = bminspec.in_PPEac_S / bminspec.in_daiAdjust;
                  // bminspec.in_PPEah_S_1 = bminspec.in_PPEah_S / bminspec.in_daiAdjust;
                  // bminspec.in_HUM_mx_1  = bminspec.in_HUM_mx / bminspec.in_daiAdjust;

                     bminspec.numList = n;
```

```
                for( int i=0; i<bminspec.daiList.size(); i++ ){
                   bminspec.daiList.set(i, bminspec.daiList.get(i) * inQ / bminspec.in_Qc_S );
                }
                //System.out.println("  daiList補正");
                break;
             }
          }
          bminspec.maxAdjustdai = bminspec.avedaiAdjust;//.in_daiAdjust;

          bminspec.in_run_stop_Num = bminspec.in_run_stop / bminspec.in_daiAdjust;


       }

    }
    //*************************----------------

 }

 /**
  * 記録
  */
 private static void record( AbstractBestModule abs, IBestRecordMessage recordNodes, BMinSpec2015 bminspec ){
    //記録ノード
    if( recordNodes != null ){
    if( CheckPrintModule.isPrintMessage ){
       //message
       recordNodes.setRecord( abs.getConnectionNode(R_NODE),
             RECORD_message, bminspec.name, bminspec.message.toString() );
    }

    if( CheckPrintModule.isPrintEnergy ){
       //記録ノードに室内機消費電力を設定
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_ID3, bminspec.name, AirFormat.df_2(bminspec.PPE));
    }

    if( CheckPrintModule.isPrintLoad ){
       //記録ノードに室内機処理熱量（顕熱）を設定
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_ID1,  bminspec.name,
AirFormat.df_2(bminspec.S_Load_RM));
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_ID1c, bminspec.name,
AirFormat.df_2(bminspec.Sc_Load_RM));
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_ID1h, bminspec.name,
AirFormat.df_2(bminspec.Sh_Load_RM));
       //記録ノードに室内機処理熱量（全熱）を設定
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_ID2,  bminspec.name,
AirFormat.df_2(bminspec.T_Load_RM));
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_ID2c, bminspec.name,
AirFormat.df_2(bminspec.Tc_Load_RM));
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_ID2h, bminspec.name,
AirFormat.df_2(bminspec.Th_Load_RM));
    }

    if( CheckPrintModule.isPrintStateOut ){
       //記録ノードに吹出口乾球温度を設定
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_Tsa, bminspec.name,
AirFormat.df_2(bminspec.DBout));
       //記録ノードに吹出口絶対湿度を設定
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_Xsa, bminspec.name,
AirFormat.df_5(bminspec.XGout));
       //記録ノードに吹出口風量を設定
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_Msa, bminspec.name, AirFormat.df_2(bminspec.GW));
       //記録ノードに吹出口CO2濃度を設定//20131029
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_CO2ppmsa, bminspec.name,
AirFormat.df_0(bminspec.airOutRM.getCO2ppm()));
       //記録ノードにドレン量を設定
       recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_ID7, bminspec.name,
AirFormat.df_2(bminspec.D_Wrate));
    }

    if( CheckPrintModule.isPrintStateMy ){
       //全熱交処理熱量
```

```
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_qSTHEX, bminspec.name,
AirFormat.df_2( bminspec.qSTHEX ));
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_qLTHEX, bminspec.name,
AirFormat.df_2( bminspec.qLTHEX ));
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_qTTHEX, bminspec.name,
AirFormat.df_2( bminspec.qTTHEX ));
        }

        if( CheckPrintModule.isPrintStateIn ){
            //記録ノードに加湿給水量を設定
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_ID8, bminspec.name,
AirFormat.df_2(bminspec.CW_Wrate));
            //記録ノードに還気乾球温度を設定
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_DBra, bminspec.name,
AirFormat.df_2(bminspec.DBra ));
            //記録ノードに還気湿球温度を設定
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_WBra, bminspec.name,
AirFormat.df_2(bminspec.WBra ));
            //記録ノードに還気絶対湿度を設定
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_Xra, bminspec.name,
AirFormat.df_5(bminspec.XGra ));
            //記録ノードに還気CO2濃度を設定//20131029
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_CO2ppmra, bminspec.name,
AirFormat.df_0(bminspec.airInRM.getCO2ppm()));
            //記録ノードに入口乾球温度を設定
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_DBin, bminspec.name,
AirFormat.df_2(bminspec.DBin ));
            //記録ノードに入口湿球温度を設定
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_WBin, bminspec.name,
AirFormat.df_2(bminspec.WBin ));
            //記録ノードに入口絶対湿度を設定
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_Xin, bminspec.name,
AirFormat.df_5(bminspec.XGin ));
            //記録ノードにOA乾球温度を設定
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_DBoa, bminspec.name,
AirFormat.df_2(bminspec.DBoa));
            //記録ノードにOA絶対湿度を設定
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_Xoa, bminspec.name,
AirFormat.df_5(bminspec.XGoa ));
            //記録ノードにOA質量流量を設定
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_Moa, bminspec.name, AirFormat.df_2(bminspec.Moa ));
            //記録ノードにOACO2濃度を設定//20131029
            recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_CO2ppmoa, bminspec.name,
AirFormat.df_0(bminspec.airInOA.getCO2ppm()));
        }


        if( CheckPrintModule.isPrintAdjust ){
            if( bminspec.isAdjust2012 ){
                //記録ノードに調整台数を設定
                recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_in_Qc_Adjust, bminspec.name,
AirFormat.df_2(bminspec.in_Qc_S) );
                recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_in_Qh_Adjust, bminspec.name,
AirFormat.df_2(bminspec.in_Qh_S) );
                recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_in_daiAdjust, bminspec.name,
AirFormat.df_2(bminspec.in_daiAdjust) );
                recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_avedaiAdjust, bminspec.name,
AirFormat.df_2(bminspec.avedaiAdjust) );
                recordNodes.setRecord( abs.getConnectionNode(R_NODE), RECORD_in_specAdjust, bminspec.name,
                      "Qc="+AirFormat.df_3(bminspec.in_Qc_S / 1000.)
                      +"[kw] PPEc="+AirFormat.df_3(bminspec.in_PPEac_S / 1000.)
                      +"Qh="+AirFormat.df_3(bminspec.in_Qh_S / 1000.)
                      +"[kw] PPEh="+AirFormat.df_3(bminspec.in_PPEah_S / 1000.)
                      +"fRate="+AirFormat.df_3(bminspec.in_Va_S * 3.)
                      +"[m3/h]"
                      );
            }
        }

        //System.out.println( "Rec GW="+GW+" FlowRateairOutRM="+this.airOutRM.getFlowRate()+ " mState="+this.mState );

        //System.out.println( "S="+((DBin-DBout)*GW*1.006)+" SL="+S_Load_RM+" Ssa="+(((DBin-DBout)*GW*1.006)-S_Load_RM) +"
```

```
Fan="+(PPE)
      //    +"    L="+((XGin-XGout)*GW*2501)+" LL="+(T_Load_RM-S_Load_RM) +" Lsa="+(((XGin-XGout)*GW*2501)-(T_Load_RM-
S_Load_RM)));
      }
   }


/* private void calc_Stop(){
      //冷暖停止処理
      this.S_Load = 0.0;
      this.T_Load = 0.0;
      this.Sc_Load = 0.0;
      this.Tc_Load = 0.0;
      this.Sh_Load = 0.0;
      this.Th_Load = 0.0;

   }
*/

   private static void cooling_cal( BMinSpec2015 bminspec ){
      double heatCapacity=0.0,rCode1 = 0.0,rCode2 = 0.0,rCode3 = 0.0,rCode4 = 0.0;
      double opeCapacity;
      double IAout1,DBout1,XGout1,S_heat,T_heat;
      double HEXrate;

      HEXrate = bminspec.in_EX_S;

      //バイパス制御 追加 エンタルピで判断
      if( bminspec.isHexbypass == true ){ //20101212nino
         bminspec.IAra = Psychrometrics.FNH( bminspec.DBra, bminspec.XGra );//20130109
         bminspec.IAoa = Psychrometrics.FNH( bminspec.DBoa, bminspec.XGoa );
         if( bminspec.IAra > bminspec.IAoa && bminspec.PIDrate.getValue() > 0.001 && bminspec.HEXselect==1 ){//20130109
            HEXrate = 0.0;
         }
         if( bminspec.DBra > bminspec.DBoa && bminspec.PIDrate.getValue() > 0.001 && bminspec.HEXselect==2 ){//20130109
            HEXrate = 0.0;
         }
      }
      if( Airswc.isON( bminspec.swcInOA )){//20101212nino
         if( bminspec.isRecord ){
            bminspec.message.append( "/OAon" );
         }
         //吸込み状態を求める
         bminspec.DBin = (( bminspec.GW - bminspec.Moa ) * bminspec.DBra + (( bminspec.DBoa - bminspec.DBra ) * ( 1.0 -
HEXrate ) + bminspec.DBra ) * bminspec.Moa ) / bminspec.GW;//20130109
         bminspec.XGin = (( bminspec.GW - bminspec.Moa ) * bminspec.XGra + (( bminspec.XGoa - bminspec.XGra ) * ( 1.0 -
HEXrate ) + bminspec.XGra ) * bminspec.Moa ) / bminspec.GW;//20130109

         bminspec.qSTHEX = ( bminspec.DBra - bminspec.DBoa ) * HEXrate * bminspec.Moa;
         bminspec.qLTHEX = ( bminspec.XGra - bminspec.XGoa ) * HEXrate * bminspec.Moa * 2501.;
         bminspec.qTTHEX = bminspec.qSTHEX + bminspec.qLTHEX;

      }else{//20101212nino
         if( bminspec.isRecord ){
            bminspec.message.append( "/OAoff" );
         }
         bminspec.DBin=bminspec.DBra;//20130109
         bminspec.XGin=bminspec.XGra;//20130109
      }
      bminspec.WBin = Psychrometrics.FNWbtx( bminspec.DBin, bminspec.XGin );//20130109
            //System.out.println("C/DBoa="+DBoa+" XGoa="+XGoa+"  ");
            //System.out.println("C/DBra="+DBra+" XGra="+XGra+"  ");
            //System.out.println("C/DBin="+DBin+" XGin="+XGin+" WBin= "+WBin);
      //    System.out.println("  ");
      if( Double.isNaN( bminspec.WBin ) || bminspec.WBin == -9999. ){//20150305
         BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
               SystemMessageConstants.CAL_ERROR2 ,
               new String[]{ "WBin 入口空気WBが異常値(DBin="+bminspec.DBin+" XGin="+bminspec.XGin+")",
                     bminspec.name+" ("+bminspec.moduleName+"_cooling_cal_WBin)" + AirSystemControl.getTimeStr() } ));
         BestLogger.info( "(E)WBin 入口空気WBが異常値(DBin="+bminspec.DBin+" XGin="+bminspec.XGin+")"
               + "__"+bminspec.name+"__("+bminspec.moduleName+"_cooling_cal_WBin)" + AirSystemControl.getTimeStr());
         if( bminspec.isRecord ){
```

```
            bminspec.message.append( "(W)WBin 入口空気WBが異常値");
        }
    }


    //上下限のチェック 20120821nino
    //double dbOA = this.DBoa;
    //double twatIn = this.t_watInHS;
    double dbOA   = bminspec.RMdata.getairInHS().getTempDB();//20120823nino
    double wbOA   = bminspec.RMdata.getairInHS().getTempWB();//20130621  外調機で使用
    double twatIn = bminspec.RMdata.getwatInHS().getTemp();//20120823nino
    double wbIn   = bminspec.WBin;//20130109
    double dbIn   = bminspec.DBin;//20141114

    double dCheck = 0;

    if( bminspec.RMdata.isWaterCooled() ){
        //水冷タイプ
        dCheck = bminspec.maxTwatIn_C_FACKctw;
        if( twatIn > dCheck ){
            //外気乾球温度＞上限の時停止
            if( bminspec.isRecord ){
                bminspec.message.append( "(C)熱源水入口温度"+twatIn+"＞"+dCheck+"上限→停止");
            }
            calc_Stop1( bminspec );
            bminspec.mState = 3;//能力＝０→換気モードとする20130415
            return;
        }
        dCheck = bminspec.minTwatIn_C_FACKctw;
        if( twatIn < dCheck ){
            //外気乾球温度＜下限の時  下限値の特性で運転
            if( bminspec.isRecord ){
                bminspec.message.append( "(C)熱源水入口温度"+twatIn+"＜"+dCheck+"下限→下限値で特性計算");
            }
            twatIn = dCheck;
        }

    }else if( bminspec.RMdata.isBESTESTtable() ){
        //BESTEST  特性table
        dCheck =
BM_EHPdata.outDBListTC[bminspec.RMdata.getN_outDB()][ BM_EHPdata.outDBListTC[bminspec.RMdata.getN_outDB()].length-1];
        if( dbOA > dCheck ){
            //外気乾球温度＞上限の時停止
            if( bminspec.isRecord ){
                bminspec.message.append( "(C)外気DB＞上限["+dCheck+"]→停止");
            }
            calc_Stop1( bminspec);
            bminspec.mState = 3;//能力＝０→換気モードとする20130415
            return;
        }
        dCheck = BM_EHPdata.outDBListTC[bminspec.RMdata.getN_outDB()][0];
        if( dbOA < dCheck ){
            //外気乾球温度＜下限の時  下限値の特性で運転
            if( bminspec.isRecord ){
                bminspec.message.append( "(C)外気DB＜下限["+dCheck+"]→下限値で特性計算");
            }
            dbOA = dCheck;
        }

    }else if( bminspec.RMdata.isHighSH() ){
        //高顕熱型
        dCheck = bminspec.maxTdbOA_C_FACKctai;
        if( dbOA > dCheck ){
            //外気乾球温度＞上限の時停止
            if( bminspec.isRecord ){
                bminspec.message.append( "(C)外気DB＞"+dCheck+"上限→停止");
            }
            calc_Stop1( bminspec );
            bminspec.mState = 3;//能力＝０→換気モードとする20130415
            return;
        }
        dCheck = bminspec.minTdbOA_C_FACKctai;
```

```
            if( dbOA < dCheck ){
               //外気乾球温度＜下限の時　下限値の特性で運転
               if( bminspec.isRecord ){
                  bminspec.message.append( "(C)外気DB＜"+dCheck+"下限→下限値で特性計算");
               }
               dbOA = dCheck;
            }

         }else{
            dCheck = bminspec.maxTdbOA_C_FACKcta;
            if( dbOA > dCheck ){
               //外気乾球温度＞上限の時停止
               if( bminspec.isRecord ){
                  bminspec.message.append( "(C)外気DB＞"+dCheck+"上限→停止");
               }
               calc_Stop1( bminspec );
               bminspec.mState = 3;//能力＝０→換気モードとする20130415
               return;
            }
            dCheck = bminspec.minTdbOA_C_FACKcta;
            if( dbOA < dCheck ){
               //外気乾球温度＜下限の時　下限値の特性で運転
               if( bminspec.isRecord ){
                  bminspec.message.append( "(C)外気DB＜"+dCheck+"下限→下限値で特性計算");
               }
               dbOA = dCheck;
            }
         }


         if( bminspec.RMdata.isBESTESTtable() ){
            //BESTEST　の特性table
            dCheck =
bminspec.pacDB.getformulaRangeMaxRaw( bminspec.pacDB.getFRACKcw( BM_EHPdata.outDBstrListSC[bminspec.RMdata.getN_outDB()][
0] ) );
            if( wbIn > dCheck ){
               //室内湿球温度＞上限の時　上限値の特性で運転
               if( bminspec.isRecord ){
                  bminspec.message.append( "(C)室WB＞"+dCheck+"上限→上限値で特性計算");
               }
               wbIn = dCheck;
            }
            dCheck =
bminspec.pacDB.getformulaRangeMinRaw( bminspec.pacDB.getFRACKcw( BM_EHPdata.outDBstrListSC[bminspec.RMdata.getN_outDB()][
0]));
            if( wbIn < dCheck){
               //室内湿球温度＜下限の時　下限値の特性で運転
               if( bminspec.isRecord ){
                  bminspec.message.append( "(C)室WB＜"+dCheck+"下限→下限値で特性計算");
               }
               wbIn = dCheck;
            }

            //System.out.println("
getformulaRangeMaxRaw="+this.pacDB.getformulaRangeMaxRaw( this.pacDB.getFRACKcw( BM_EHPdata.outDBstrListSC[0][0] ))
            //    +"getformulaRangeMinRaw="+
this.pacDB.getformulaRangeMinRaw( this.pacDB.getFRACKcw( BM_EHPdata.outDBstrListSC[0][0]))
            //    +"
getformulaRangeMaxCol="+this.pacDB.getformulaRangeMaxCol( this.pacDB.getFRACKcw( BM_EHPdata.outDBstrListSC[0][0] ))
            //    +"getformulaRangeMinCol="+
this.pacDB.getformulaRangeMinCol( this.pacDB.getFRACKcw( BM_EHPdata.outDBstrListSC[0][0]))
            //    +" MaxOutDB="+BM_EHPdata.outDBListTC[0][ BM_EHPdata.outDBListTC[0].length-1]
            //    +" MinOutDB="+BM_EHPdata.outDBListTC[0][0] );

         }else if( bminspec.RMdata.isHighSH() ){
            //高顕熱型
            dCheck = bminspec.maxTwbIn_C_FACKctai;
            if( wbIn > dCheck ){//20130109
               //室内湿球温度＞上限の時　上限値の特性で運転
               if( bminspec.isRecord ){
                  bminspec.message.append( "(C)室WB＞"+dCheck+"上限→上限値で特性計算");
               }
```

```
            wbIn = dCheck;//20130109
        }
        dCheck = bminspec.minTwbIn_C_FACKctai;
        if( wbIn < dCheck ){//20130109
            //室内湿球温度＜下限の時　下限値の特性で運転
            if( bminspec.isRecord ){
                bminspec.message.append( "(C)室WB＜"+dCheck+"下限→下限値で特性計算");
            }
            wbIn = dCheck;//20130109
        }

    }else{
        dCheck = bminspec.maxTwbIn_C_FACKcti;
        if( wbIn > dCheck ){//20130109
            //室内湿球温度＞上限の時　上限値の特性で運転
            if( bminspec.isRecord ){
                bminspec.message.append( "(C)室WB＞"+dCheck+"上限→上限値で特性計算");
            }
            wbIn = dCheck;//20130109
        }
        dCheck = bminspec.minTwbIn_C_FACKcti;
        if( wbIn < dCheck ){//20130109
            //室内湿球温度＜下限の時　下限値の特性で運転
            if( bminspec.isRecord ){
                bminspec.message.append( "(C)室WB＜"+dCheck+"下限→下限値で特性計算");
            }
            wbIn = dCheck;//20130109
        }
    }



    //能力補正
    if( !bminspec.isOAProcessingUnit && bminspec.RMdata.isWaterCooled() ){
        //rCode1 = this.RMdata.get_out_c_CAPrevision();
        rCode1 = bminspec.pacDB.getKctw( twatIn );//20121823nino
        //20181018誘導基準申請計算室内温度特性
        if( bminspec.is2018SinseiOA ) {
            rCode2 = 1;
        }else {
            rCode2 = bminspec.pacDB.getKcti( wbIn );//20130109
        }
    }else if( !bminspec.isOAProcessingUnit ){
        if( bminspec.RMdata.isBESTESTtable() ){
            //BESTESTtable 能力
            double r1 = bminspec.pacDB.getRACKcw(wbIn, dbIn, dbOA, BM_EHPdata.outDBListTC[bminspec.RMdata.getN_outDB()],
BM_EHPdata.outDBstrListTC[bminspec.RMdata.getN_outDB()] );//20130722
            double r2 = bminspec.pacDB.getRACKcw(wbIn, dbIn, dbOA, BM_EHPdata.outDBListSC[bminspec.RMdata.getN_outDB()],
BM_EHPdata.outDBstrListSC[bminspec.RMdata.getN_outDB()] );//20130722
            rCode1 = ( r1 > r2 )? r1 : r2;
            //System.out.println(" r1="+r1+"  r2="+r2 +" wbIn="+wbIn+" dbIn="+dbIn+" dbOA="+dbOA);
            rCode2 = 1.;
        }else if( bminspec.RMdata.isHighSH() ){
            //高顕熱型
            rCode1 = bminspec.pacDB.getKctai( dbOA, wbIn );//20130722
            rCode2 = 1.;
        }else{
            rCode1 = bminspec.pacDB.getKcta( dbOA );
            //20181018誘導基準申請計算室内温度特性
            if( bminspec.is2018SinseiOA ) {
                rCode2 = 1;
            }else {
                rCode2 = bminspec.pacDB.getKcti( wbIn );//20130109
            }
        }

    }else{
        //外気処理エアコン（全熱交は無い）
        if( dbOA < 19 ){
            //外気乾球温度＜下限の時停止
            if( bminspec.isRecord ){
                bminspec.message.append( "(C)外気DB＜下限19→停止換気");
```

```
    }
    calc_Stop1( bminspec );
    bminspec.mState = 3;//能力＝０→換気モードとする20150319
    return;
}else if( dbOA > 43 ){
    //外気乾球温度＞上限の時停止
    if( bminspec.isRecord ){
        bminspec.message.append( "(C)外気DB＞上限43→停止換気");
    }
    calc_Stop1( bminspec );
    bminspec.mState = 3;//能力＝０→換気モードとする20150319
    return;
}
if( Psychrometrics.FNRhtx(dbOA, bminspec.XGoa ) < 30 ){
    //外気相対湿度＜下限の時停止
    if( bminspec.isRecord ){
        bminspec.message.append( "(C)外気RH＜下限30→停止換気");
    }
    calc_Stop1( bminspec );
    bminspec.mState = 3;//能力＝０→換気モードとする20150319
    return;
}else if( Psychrometrics.FNRhtx(dbOA, bminspec.XGoa ) > 90 ){
    //外気相対湿度＞上限の時停止
    if( bminspec.isRecord ){
        bminspec.message.append( "(C)外気RH＞上限90→停止換気");
    }
    calc_Stop1( bminspec );
    bminspec.mState = 3;//能力＝０→換気モードとする20150319
    return;
}

bminspec.pacDB.setEquipmentName( bminspec.filenames[1] );

dCheck = bminspec.maxTdbOA_C_FACKctdb;
if( dbOA > dCheck ){//21030621
    //外気乾球温度＞上限の時停止
    if( bminspec.isRecord ){
        bminspec.message.append( "(C)外気DB＞"+dCheck+"上限→上限値で計算");
    }
    dbOA = dCheck;//21030621
}
dCheck = bminspec.minTdbOA_C_FACKctdb;
if( dbOA < dCheck ){//21030621
    //外気乾球温度＜下限の時　下限値の特性で運転
    if( bminspec.isRecord ){
        bminspec.message.append( "(C)外気DB＜"+dCheck+"下限→下限値で特性計算");
    }
    dbOA = dCheck;//21030621
}

dCheck = bminspec.maxTwbOA_C_FACKctwb;
if( wbOA > dCheck ){//20130621
    //外気湿球温度＞上限の時　上限値の特性で運転
    if( bminspec.isRecord ){
        bminspec.message.append( "(C)外気WB＞"+dCheck+"上限→上限値で特性計算");
    }
    wbOA = dCheck;//20130621
}
dCheck = bminspec.minTwbOA_C_FACKctwb;
if( wbOA < dCheck ){//20130621
    //外気湿球温度＜下限の時　下限値の特性で運転
    if( bminspec.isRecord ){
        bminspec.message.append( "(C外気WB＜"+dCheck+"下限→下限値で特性計算");
    }
    wbOA = dCheck;//20130621
}

rCode1 = bminspec.pacDB.getKctdb( dbOA );
//20181018誘導基準申請計算室内温度特性
if( bminspec.is2018SinseiOA ) {
    rCode2 = 1;
}else {
```

```
            rCode2 = bminspec.pacDB.getKctwb( wbOA );//20130109//20130621
        }
        bminspec.pacDB.setEquipmentName( bminspec.filenames[0] );
    }

    rCode3 = bminspec.pacDB.getKcLpi( bminspec.in_Lp );
    if(bminspec.in_Lh>=0){
        rCode4 = bminspec.pacDB.getKchu( bminspec.in_Lh );
    }else{
        rCode4 = bminspec.pacDB.getKchu( bminspec.in_Lh );
    }

    rCode4 = bminspec.pacDB.getKchu( bminspec.in_Lh );
    bminspec.LPrate = rCode3 * rCode4;

    heatCapacity = rCode1 * rCode2 * rCode3 * rCode4 * bminspec.in_Qc_S;

          //System.out.println("heatCapacity ="+rCode1 * rCode2 * rCode3 * rCode4+" "+heatCapacity+"  ");
    if( bminspec.isRecord ){
        bminspec.message.append( "Qc="+AirFormat.df_0(bminspec.in_Qc_S)+"rCode1~4 [1]="+AirFormat.df_3(rCode1)
        +"[2]="+AirFormat.df_3(rCode2)+"[3]="+AirFormat.df_3(rCode3)+"[4]="+AirFormat.df_3(rCode4) );
    }

    opeCapacity = heatCapacity * bminspec.PIDrate.getValue();//負荷率をここで適用
        //System.out.println("heatCapacity ="+AirFormat.df_0( heatCapacity/1000 )+"kW  opeCapacity="+
AirFormat.df_0( opeCapacity/1000 )+"kW");

    if( opeCapacity == 0 ){
        bminspec.T_C_heat = 0.0;
        bminspec.S_Load = 0.0;
        bminspec.T_Load = 0.0;
        bminspec.Sc_Load = 0.0;
        bminspec.Tc_Load = 0.0;
        bminspec.Sh_Load = 0.0;
        bminspec.Th_Load = 0.0;

    }else
    if( bminspec.isOAProcessingUnit ){
        //外調機の場合
        //吸込状態
        bminspec.IAin = Psychrometrics.FNH( dbOA, bminspec.XGin );//20130109
        IAout1 = bminspec.IAin - opeCapacity / bminspec.GW * 1000.0;//20130109
        //出口状態
        DBout1 = Psychrometrics.FNDbrh( 90.0, IAout1 );//乾球温度
        XGout1 = Psychrometrics.FNxtr( DBout1, 90.0 );//絶対湿度


        if( bminspec.XGin < XGout1 ){//20130109
            //System.out.println( "XGout1="+XGout1+"-->"+XGin);

            XGout1 = bminspec.XGin; //20130109
            DBout1 = Psychrometrics.FNDbxh( XGout1, IAout1 );
        }
        S_heat = ( dbOA - DBout1 ) * bminspec.GW * 1.006;/////////修正//20130109

        if( S_heat > opeCapacity ){
            S_heat = opeCapacity;
        }
        if(Math.abs( bminspec.XGin - XGout1 ) < 0.00001 ){//20130109
            T_heat=S_heat;
        }else{
            T_heat = opeCapacity;
        }

        bminspec.T_C_heat = T_heat;//******************090725

        //if( this.PIDrate.getValue() > 1. ){//20130415
        // this.S_Load = S_heat;
        // this.T_Load = T_heat;
        //}else{
        bminspec.S_Load = S_heat;//ここから負荷率を削除
        bminspec.T_Load = T_heat;//ここから負荷率を削除
```

```
            //}
            bminspec.Sc_Load = bminspec.S_Load;
            bminspec.Tc_Load = bminspec.T_Load;
            bminspec.Sh_Load = 0.0;
            bminspec.Th_Load = 0.0;

        }else{
            //吸込状態
            bminspec.IAin = Psychrometrics.FNH( bminspec.DBin, bminspec.XGin );//20130109
            //出口状態
            if( bminspec.numCalcSA == 1 ){//20141110 BESTEST
               //可能冷却能力の時の出口heatCapacity
               //if( this.isFanOnOff ){
               // IAout1 = IAin - ( heatCapacity - this.in_PPEac_S * this.PIDrate.getValue()  ) / this.GW *
1000.0;//20130109
               //
               //}else{
               // IAout1 = IAin - ( heatCapacity - this.in_PPEac_S  ) / this.GW * 1000.0;//20130109
               //}
               IAout1 = bminspec.IAin - ( heatCapacity  ) / bminspec.GW * 1000.0;//20130109


               //System.out.println("this.valCalcSA="+this.valCalcSA+ "");
               //バイパスファクター　0.049～0.08　　＝＝＞0.065
               //ルームエアコンは　バイパスファクター20%で吹き出しポイントを求める　　//BMは相対湿度90%としている
               double IAout100RH = bminspec.IAin - ( bminspec.IAin - IAout1 ) / ( 1 - bminspec.valCalcSA );//)0.935;
               double DBout100RH = Psychrometrics.FNDbrh( 100.0, IAout100RH );
               double XGout100RH = Psychrometrics.FNxtr( DBout100RH, 100.0);//XGout1 = Psychrometrics.FNxtr( DBout1,
90.0);//絶対湿度
               if( XGout100RH > bminspec.XGin ){
                  //System.out.println( "------ sa ="+(XGout100RH-XGin));
                  XGout100RH = bminspec.XGin;
                  XGout1 = bminspec.XGin;
                  if( bminspec.isFanOnOff ){
                     //DBout1 = DBin - ( opeCapacity - this.in_PPEac_S * this.PIDrate.getValue() )/ this.GW / 1.006;
                     DBout1 = bminspec.DBin - ( opeCapacity - bminspec.in_PPEac_S * bminspec.PIDrate.getValue() )/
bminspec.GW / 1.006;
                  }else{
                     //DBout1 = DBin - ( opeCapacity - this.in_PPEac_S )/ this.GW / 1.006;
                     DBout1 = bminspec.DBin - ( opeCapacity - bminspec.in_PPEac_S )/ bminspec.GW / 1.006;
                  }
               }else{
                  XGout1 = XGout100RH + ( bminspec.XGin - XGout100RH ) * bminspec.valCalcSA;//可能能力時
                  DBout1 = DBout100RH + ( bminspec.DBin - DBout100RH ) * bminspec.valCalcSA;//可能能力時
                  //部分負荷時を計算
                  if( heatCapacity == 0 ){
                     XGout1 = bminspec.XGin;
                     DBout1 = bminspec.DBin;
                  }else{
                     XGout1 = bminspec.XGin - ( bminspec.XGin - XGout1 ) * opeCapacity / heatCapacity;
                     DBout1 = bminspec.DBin - ( bminspec.DBin - DBout1 ) * opeCapacity / heatCapacity;
                  }
                  if( bminspec.isFanOnOff ){
                     DBout1 += bminspec.in_PPEac_S * bminspec.PIDrate.getValue() / bminspec.GW / 1.006;
                  }else{
                     DBout1 += bminspec.in_PPEac_S / bminspec.GW / 1.006;
                  }
               }

               //System.out.println( "++++opeCapacity="+opeCapacity+ " IAin_out="+((IAin-IAout1)*GW/1000)+" DBsa="+((DBin-
Psychrometrics.FNDbxh( XGout1, IAout1 ))*GW*1.006));
               if( bminspec.isFanOnOff ){
                  S_heat = ( bminspec.DBin - DBout1 ) * bminspec.GW * 1.006 + bminspec.in_PPEac_S *
bminspec.PIDrate.getValue();
               }else{
                  S_heat = ( bminspec.DBin - DBout1 ) * bminspec.GW * 1.006 + bminspec.in_PPEac_S;

               }
            }else
            if( bminspec.numCalcSA == 11 ){//20141110 BESTEST
               //必要熱量時の出口opeCapacity
               IAout1 = bminspec.IAin - ( opeCapacity ) / bminspec.GW * 1000.0;//20130109
```

```
            //System.out.println("this.valCalcSA="+this.valCalcSA+ "");
            //バイパスファクター　0.049～0.08　　＝＝＞0.065
            //ルームエアコンは　バイパスファクター20%で吹き出しポイントを求める　　//BMは相対湿度90%としている
            double IAout100 = bminspec.IAin - ( bminspec.IAin - IAout1 ) / ( 1 - bminspec.valCalcSA );//)0.935;
            double DBout100 = Psychrometrics.FNDbrh( 100.0, IAout100 );
            double XGout100 = Psychrometrics.FNxtr( DBout100, 100.0);//XGout1 = Psychrometrics.FNxtr( DBout1, 90.0);//絶
対湿度
            if( XGout100 > bminspec.XGin ){
               XGout100 = bminspec.XGin;
               //System.out.println( "------");
               XGout1 = bminspec.XGin;
               if( bminspec.isFanOnOff ){
                  DBout1 = bminspec.DBin - ( opeCapacity - bminspec.in_PPEac_S * bminspec.PIDrate.getValue() )/
bminspec.GW / 1.006;
               }else{
                  DBout1 = bminspec.DBin - ( opeCapacity - bminspec.in_PPEac_S )/ bminspec.GW / 1.006;
               }
            }else{
               XGout1 = XGout100 + ( bminspec.XGin - XGout100 ) * bminspec.valCalcSA;
               if( bminspec.isFanOnOff ){
                  DBout1 = Psychrometrics.FNDbxh( XGout1, IAout1 ) + bminspec.in_PPEac_S * bminspec.PIDrate.getValue() /
bminspec.GW / 1.006;
               }else{
                  DBout1 = Psychrometrics.FNDbxh( XGout1, IAout1 ) + bminspec.in_PPEac_S / bminspec.GW / 1.006;
               }
            }

            //System.out.println( "++++opeCapacity="+opeCapacity+ " IAin_out="+((IAin-IAout1)*GW/1000)+" DBsa="+((DBin-
Psychrometrics.FNDbxh( XGout1, IAout1 ))*GW*1.006));
            if( bminspec.isFanOnOff ){
               S_heat = ( bminspec.DBin - DBout1 ) * bminspec.GW * 1.006 + bminspec.in_PPEac_S *
bminspec.PIDrate.getValue();
            }else{
               S_heat = ( bminspec.DBin - DBout1 ) * bminspec.GW * 1.006 + bminspec.in_PPEac_S;

            }

         }else{// if( this.numCalcSA == 0 ){
            //吹出し状態の相対湿度を設定
            IAout1 = bminspec.IAin - opeCapacity / bminspec.GW * 1000.0;//20130109

            DBout1 = Psychrometrics.FNDbrh( bminspec.valCalcSA * 100, IAout1 );//乾球温度
            XGout1 = Psychrometrics.FNxtr( DBout1, bminspec.valCalcSA * 100 );//絶対湿度

            if( bminspec.XGin < XGout1 ){//20130109
               //valCalcSAの相対湿度線に到達していない＝顕熱処理だけ
               //System.out.println( "XGout1="+XGout1+"-->"+XGin);

               XGout1 = bminspec.XGin; //20130109
               //DBout1 = Psychrometrics.FNDbxh( XGout1, IAout1 );
               DBout1 = bminspec.DBin - opeCapacity / bminspec.GW / 1.006;//20150414　誤差低減化（空気線図関数の使用から
変更）
               S_heat = opeCapacity;//20150414　誤差低減化

            }else{
               S_heat = ( bminspec.DBin - DBout1 ) * bminspec.GW * 1.006;/////////修正//20130109
            }

         }


         //    System.out.println("IAra="+IAra+"  IAout1="+IAout1+"  in_Va_S="+this.in_Va_S);


         // System.out.println( "S_heat="+S_heat + " DBra="+ this.DBra + " DBout1="+DBout1);

         if( S_heat > opeCapacity ){
            //System.out.println( "******S_heat > opeCapacity  sa="+(S_heat-opeCapacity) +" S_heat="+S_heat +"
opeCapacity="+opeCapacity);
            S_heat = opeCapacity;
            //System.out.println( "******S_heat > opeCapacity  XG sa="+(XGout1-XGin) );
```

```
            XGout1 = bminspec.XGin;
        }
        if(Math.abs( bminspec.XGin - XGout1 ) < 0.0 ){//20130109
            //System.out.println( "******Math.abs( this.XGin - XGout1 ) < 0.0000" );
            T_heat = S_heat;
        }else{
            T_heat = opeCapacity;
            //System.out.println( "S_heat="+ S_heat+" T_heat="+T_heat+ " sa="+(T_heat-S_heat));
        }

        bminspec.T_C_heat = T_heat;//******************090725

        //if( this.PIDrate.getValue() > 1. ){//20130415
        // this.S_Load = S_heat;
        // this.T_Load = T_heat;
        //}else{
        bminspec.S_Load = S_heat;//ここから負荷率を削除
        bminspec.T_Load = T_heat;//ここから負荷率を削除
        //}
        bminspec.Sc_Load = bminspec.S_Load;
        bminspec.Tc_Load = bminspec.T_Load;
        bminspec.Sh_Load = 0.0;
        bminspec.Th_Load = 0.0;

        if( bminspec.PIDrate.getValue() < bminspec.in_run_stop ){
            bminspec.S_Load = 0.0;
            bminspec.T_Load = 0.0;
            bminspec.Sc_Load = 0.0;
            bminspec.Tc_Load = 0.0;
            bminspec.Sh_Load = 0.0;
            bminspec.Th_Load = 0.0;
            if( bminspec.isRecord ){
                bminspec.message.append( "(C)負荷率＜停止負荷率で停止");
            }
            bminspec.mState = 3;//能力＝０→換気モードとする20130415
        }
    }

    //    System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate.getValue());


}



  private static void heating_cal( BMinSpec2015 bminspec ){
    double heatCapacity=0.0,rCode1 = 0.0,rCode2 = 0.0,rCode3 = 0.0,rCode4 = 0.0;
    double opeCapacity;
    double IAout1,DBout1,XGout1,S_heat;
    double HEXrate;

    HEXrate = bminspec.in_EX_S;
    //バイパス制御 追加 20101212nino
    if( bminspec.isHexbypass == true ){
        bminspec.IAra = Psychrometrics.FNH( bminspec.DBra, bminspec.XGra );//20130109
        bminspec.IAoa = Psychrometrics.FNH( bminspec.DBoa, bminspec.XGoa );
        if( bminspec.IAra < bminspec.IAoa && bminspec.PIDrate.getValue() > 0.001 && bminspec.HEXselect == 1 ){//20130109
            HEXrate = 0.0;
        }
        if( bminspec.DBra < bminspec.DBoa && bminspec.PIDrate.getValue() > 0.001 && bminspec.HEXselect == 2 ){//20130109
            HEXrate = 0.0;
        }
    // if( bminspec.IAra > bminspec.IAoa && bminspec.PIDrate.getValue() < 0.001 && bminspec.HEXselect == 1 ){//20130109
//20160920暖房時負荷なしの時の冷房運転は考慮しないこととする
    //    HEXrate = 0.0;
    // }
    // if( bminspec.DBra > bminspec.DBoa && bminspec.PIDrate.getValue() < 0.001 && bminspec.HEXselect == 2 ){//20130109
//20160920暖房時負荷なしの時の冷房運転は考慮しないこととする
    //    HEXrate = 0.0;
    // }
    }
    if( Airswc.isON( bminspec.swcInOA )){//20101212nino
```

```
            if( bminspec.isRecord ){
               bminspec.message.append( "/OAon" );
            }
            bminspec.DBin = (( bminspec.in_Va_S - bminspec.Moa ) * bminspec.DBra + ( bminspec.DBra - ( bminspec.DBra -
bminspec.DBoa ) * ( 1.0 - HEXrate )) * bminspec.Moa ) / bminspec.in_Va_S;//20130109
            bminspec.XGin = (( bminspec.in_Va_S - bminspec.Moa ) * bminspec.XGra + ( bminspec.XGra - ( bminspec.XGra -
bminspec.XGoa ) * ( 1.0 - HEXrate )) * bminspec.Moa ) / bminspec.in_Va_S;//20130109

            bminspec.qSTHEX = ( bminspec.DBra - bminspec.DBoa ) * HEXrate * bminspec.Moa;
            bminspec.qLTHEX = ( bminspec.XGra - bminspec.XGoa ) * HEXrate * bminspec.Moa * 2501.;
            bminspec.qTTHEX = bminspec.qSTHEX + bminspec.qLTHEX;

         }else{//20101212nino
            if( bminspec.isRecord ){
               bminspec.message.append( "/OAoff" );
            }
            //外気処理専用の場合は　外気//20150319
            if( bminspec.isOAProcessingUnit ){//20150319
               bminspec.DBin = bminspec.DBoa;
               bminspec.XGin = bminspec.XGoa;
            }else{
               bminspec.DBin = bminspec.DBra;//20130109
               bminspec.XGin = bminspec.XGra;//20130109
            }
         }
         bminspec.WBin = Psychrometrics.FNWbtx(bminspec.DBin,bminspec.XGin);//20130109

         //    System.out.println("H/DBoa="+DBoa+" "+XGoa+"  ");
         //    System.out.println("H/DBra="+DBra+" "+XGra+"  ");
         //    System.out.println("H/DBin="+DBin+" "+XGin+"  ");
         //    System.out.println("  ");

         //    rCode1 = pacDB.getKhta(DBoa);
         //    rCode2 = pacDB.getKhti(WBin);

         //上下限のチェック 20120821nino
         //double wbOA = bminspec.WBoa;
         //double twatIn = bminspec.t_watInHS;
         double dbOA = bminspec.RMdata.getairInHS().getTempDB();//20130621外調機で使用
         double wbOA = bminspec.RMdata.getairInHS().getTempWB();//20120823nino
         double twatIn = bminspec.RMdata.getwatInHS().getTemp();//20120823nino
         double dbIn = bminspec.DBin;//20130109

         //System.out.println( " wbOA="+wbOA +"  dbIn="+dbIn + "  DBra="+DBra);

         double dCheck = 0;

         if( bminspec.RMdata.isWaterCooled() ){
            dCheck = bminspec.maxTwatIn_H_FACKhtw;
            if( twatIn > dCheck ){
               //熱源水入口温度＞上限の時　上限値の特性で運転
               if( bminspec.isRecord ){
                  bminspec.message.append( "(C)熱源水入口温度"+twatIn+"＞"+dCheck+"上限→上限値で特性計算");
               }
               twatIn = dCheck;
            }
            dCheck = bminspec.minTwatIn_H_FACKhtw;
            if( twatIn < dCheck ){
               //熱源水入口温度＜下限の時　停止
               if( bminspec.isRecord ){
                  bminspec.message.append( "(C)熱源水入口温度"+twatIn+"＜"+dCheck+"下限→停止");
               }
               calc_Stop1( bminspec );
               bminspec.mState = 3;//能力＝０→換気モードとする20130415
               return;
            }
         }else if( bminspec.RMdata.isHighSH() ){
            //高顕熱型
            dCheck = bminspec.maxTwbOA_H_FACKhtai;
            if( wbOA > dCheck ){
               //外気湿球温度＞上限の時　上限値の特性で運転
               if( bminspec.isRecord ){
```

```
            bminspec.message.append( "(C)外気WB＞"+dCheck+"上限→上限値で特性計算");
        }
        wbOA = dCheck;
    }
    dCheck = bminspec.minTwbOA_H_FACKhtai;
    if( wbOA < dCheck ){
        //外気湿球温度＜下限の時　停止
        if( bminspec.isRecord ){
            bminspec.message.append( "(C)外気WB＜"+dCheck+"下限→停止");
        }
        calc_Stop1( bminspec );
        bminspec.mState = 3;//能力＝０→換気モードとする20130415
        return;
    }
}else{
    dCheck = bminspec.maxTwbOA_H_FACKhta;
    if( wbOA > dCheck ){
        //外気湿球温度＞上限の時　上限値の特性で運転
        if( bminspec.isRecord ){
            bminspec.message.append( "(C)外気WB＞"+dCheck+"上限→上限値で特性計算");
        }
        wbOA = dCheck;
    }
    dCheck = bminspec.minTwbOA_H_FACKhta;
    if( wbOA < dCheck ){
        //外気湿球温度＜下限の時　停止
        if( bminspec.isRecord ){
            bminspec.message.append( "(C)外気WB＜"+dCheck+"下限→停止");
        }
        calc_Stop1( bminspec );
        bminspec.mState = 3;//能力＝０→換気モードとする20130415
        return;
    }
}

if( bminspec.RMdata.isHighSH() ){
    //高顕熱型
    dCheck = bminspec.maxTdbIn_H_FACKhtai;
    if( dbIn > dCheck ){//20130109
        //室内乾球温度＞上限の時　上限値の特性で運転
        if( bminspec.isRecord ){
            bminspec.message.append( "(C)室DB＞"+dCheck+"上限→上限値で特性計算");
        }
        dbIn = dCheck;//20130109
    }
    dCheck = bminspec.minTdbIn_H_FACKhtai;
    if( dbIn < dCheck ){//20130109
        //室内乾球温度＜下限の時　下限値の特性で運転
        if( bminspec.isRecord ){
            bminspec.message.append( "(C)室DB＜"+dCheck+"下限→下限値で特性計算");
        }
        dbIn = dCheck;//20130109
    }

}else{
    dCheck = bminspec.maxTdbIn_H_FACKhti;
    if( dbIn > dCheck ){//20130109
        //室内乾球温度＞上限の時　上限値の特性で運転
        if( bminspec.isRecord ){
            bminspec.message.append( "(C)室DB＞"+dCheck+"上限→上限値で特性計算");
        }
        dbIn = dCheck;//20130109
    }
    dCheck = bminspec.minTdbIn_H_FACKhti;
    if( dbIn < dCheck ){//20130109
        //室内乾球温度＜下限の時　下限値の特性で運転
        if( bminspec.isRecord ){
            bminspec.message.append( "(C)室DB＜"+dCheck+"下限→下限値で特性計算");
        }
        dbIn = dCheck;//20130109
    }
```

```
}

//能力補正
if( !bminspec.isOAProcessingUnit && bminspec.RMdata.isWaterCooled() ){
   //rCode1 = bminspec.RMdata.get_out_h_CAPrevision();
   rCode1 = bminspec.pacDB.getKhtw( twatIn );//20120823nino
   //20181018誘導基準申請計算室内温度特性
   if( bminspec.is2018SinseiOA ) {
      rCode2 = 1;
   }else {
      rCode2 = bminspec.pacDB.getKhti( dbIn );//20120823nino Kcti-->khti WBra-->DBra//20130109
   }
}else if( !bminspec.isOAProcessingUnit ){
   //System.out.println(" wbOA="+wbOA +"  dbIn="+dbIn);
   if( bminspec.RMdata.isHighSH() ){
      rCode1 = bminspec.pacDB.getKhtai( wbOA, dbIn );//20130109
      rCode2 = 1.;

   }else{
      rCode1 = bminspec.pacDB.getKhta( wbOA );
      //20181018誘導基準申請計算室内温度特性
      if( bminspec.is2018SinseiOA ) {
         rCode2 = 1;
      }else {
         rCode2 = bminspec.pacDB.getKhti( dbIn );//20130109
      }

   }
}else{
   //外気処理エアコン（全熱交なし）//20110113nino
   if( dbOA < -5 ){
      //外気乾球温度＜下限の時停止
      if( bminspec.isRecord ){
         bminspec.message.append( "(C)外気DB＜下限-5→停止換気");
      }
      calc_Stop1( bminspec );
      bminspec.mState = 3;//能力＝０→換気モードとする20150319
      return;
   }else if( dbOA > 15 ){
      //外気乾球温度＞上限の時停止
      if( bminspec.isRecord ){
         bminspec.message.append( "(C)外気DB＞上限15→停止換気");
      }
      calc_Stop1( bminspec );
      bminspec.mState = 3;//能力＝０→換気モードとする20150319
      return;
   }
   if( Psychrometrics.FNRhtx(dbOA, bminspec.XGoa ) < 20 ){
      //外気相対湿度＜下限の時停止
      if( bminspec.isRecord ){
         bminspec.message.append( "(C)外気RH＜下限20→停止換気");
      }
      calc_Stop1( bminspec );
      bminspec.mState = 3;//能力＝０→換気モードとする20150319
      return;
   }else if( Psychrometrics.FNRhtx(dbOA, bminspec.XGoa ) > 90 ){
      //外気相対湿度＞上限の時停止
      if( bminspec.isRecord ){
         bminspec.message.append( "(C)外気RH＞上限90→停止換気");
      }
      calc_Stop1( bminspec );
      bminspec.mState = 3;//能力＝０→換気モードとする20150319
      return;
   }

   bminspec.pacDB.setEquipmentName( bminspec.filenames[1] );

   dCheck = bminspec.maxTdbOA_H_FACKhtdb;
   if( dbOA > dCheck ){//20130621
      //室内乾球温度＞上限の時　上限値の特性で運転
      if( bminspec.isRecord ){
         bminspec.message.append( "(C)室DB＞"+dCheck+"上限→上限値で特性計算");
```

```
        }
        dbOA = dCheck;//20130621
    }
    dCheck = bminspec.minTdbOA_H_FACKhtdb;
    if( dbOA < dCheck ){//20130621
        //室内乾球温度＜下限の時　下限値の特性で運転
        if( bminspec.isRecord ){
            bminspec.message.append( "(C)室DB＜"+dCheck+"下限→下限値で特性計算");
        }
        dbOA = dCheck;//20130621
    }
    dCheck = bminspec.maxTwbOA_H_FACKhtwb;
    if( wbOA > dCheck ){//21030621
        //外気湿球温度＞上限の時　上限値の特性で運転
        if( bminspec.isRecord ){
            bminspec.message.append( "(C)外気WB＞"+dCheck+"上限→上限値で特性計算");
        }
        wbOA = dCheck;//21030621
    }
    dCheck = bminspec.minTwbOA_H_FACKhtwb;
    if( wbOA < dCheck ){//21030621
        //外気湿球温度＜下限の時　停止
        if( bminspec.isRecord ){
            bminspec.message.append( "(C)外気WB＜"+dCheck+"下限→下限値で計算");
        }
        wbOA = dCheck;//21030621
    }
    rCode1 = bminspec.pacDB.getKhtwb( wbOA );
    //20181018誘導基準申請計算室内温度特性
    if( bminspec.is2018SinseiOA ) {
        rCode2 = 1;
    }else {
        rCode2 = bminspec.pacDB.getKhtdb( dbOA );//20130621
    }
    bminspec.pacDB.setEquipmentName( bminspec.filenames[0] );
}

rCode3 = bminspec.pacDB.getKhLpi( bminspec.in_Lp );
if( bminspec.in_Lh >= 0 ){
    rCode4 = bminspec.pacDB.getKhhu( bminspec.in_Lh );
}else{
    rCode4 = bminspec.pacDB.getKhhu( bminspec.in_Lh );
}
bminspec.LPrate = rCode3 * rCode4;

heatCapacity = rCode1 * rCode2 * rCode3 * rCode4 * bminspec.in_Qh_S;
//    System.out.println("heatCapacity ="+rCode1+" "+rCode2+" "+rCode3+" "+rCode4+" "+heatCapacity+"  ");

if( heatCapacity == 0 ){
    if( bminspec.isRecord ){
        bminspec.message.append( "(C)heatCapacity=0");
    }
}

//    System.out.println("heatCapacity ="+rCode1 * rCode2 * rCode3 * rCode4+" "+heatCapacity+"  ");
if( bminspec.isRecord ){
    bminspec.message.append( "Qh="+AirFormat.df_0(bminspec.in_Qh_S)+"rCode1~4 [1]="+AirFormat.df_3(rCode1)
    +"[2]="+AirFormat.df_3(rCode2)+"[3]="+AirFormat.df_3(rCode3)+"[4]="+AirFormat.df_3(rCode4) );
}

opeCapacity = heatCapacity * bminspec.PIDrate.getValue();//20130109 //負荷率をここで適用20130703

if( opeCapacity == 0 ){
    bminspec.T_C_heat = 0.0;
    bminspec.S_Load = 0.0;
    bminspec.T_Load = 0.0;
    bminspec.Sc_Load = 0.0;
    bminspec.Tc_Load = 0.0;
    bminspec.Sh_Load = 0.0;
    bminspec.Th_Load = 0.0;

}else
```

```
        if( bminspec.isOAProcessingUnit ){//20130621
            //外調機の場合
            //吸込状態
            bminspec.IAin = Psychrometrics.FNH( dbOA, bminspec.XGin );//20130109
            IAout1 = bminspec.IAin + opeCapacity / bminspec.in_Va_S * 1000.0;

            //外気処理エアコンの場合はここで加湿能力を計算する
            bminspec.pacDB.setEquipmentName( bminspec.filenames[1] );
            S_heat = opeCapacity - bminspec.in_QhL_S * bminspec.PIDrate.getValue() * bminspec.pacDB.getKhutdb( dbOA ) *
bminspec.pacDB.getKhutwb( wbOA );//負荷率をここで適用20130703
            XGout1 = bminspec.XGin + (opeCapacity - S_heat) / 2500 / bminspec.in_Va_S;//20130109
            DBout1 = Psychrometrics.FNDbxh(XGout1,IAout1);
            bminspec.pacDB.setEquipmentName( bminspec.filenames[0] );

            bminspec.S_Load = S_heat;//負荷率をここから削除20130703
            bminspec.T_Load = opeCapacity;

            bminspec.Sc_Load = 0.0;
            bminspec.Tc_Load = 0.0;
            bminspec.Sh_Load = bminspec.S_Load;
            bminspec.Th_Load = bminspec.T_Load;


        }else{
            //吸込状態
            bminspec.IAin = Psychrometrics.FNH( bminspec.DBin, bminspec.XGin );//20130109
            IAout1 = bminspec.IAin + opeCapacity / bminspec.in_Va_S * 1000.0;//20130109

            XGout1 = bminspec.XGin;//20130109
            DBout1 = Psychrometrics.FNDbxh(XGout1,IAout1);
            S_heat = opeCapacity;

            bminspec.S_Load = S_heat;
            bminspec.T_Load = opeCapacity;

            bminspec.Sc_Load = 0.0;
            bminspec.Tc_Load = 0.0;
            bminspec.Sh_Load = bminspec.S_Load;
            bminspec.Th_Load = bminspec.T_Load;

            if( bminspec.PIDrate.getValue() < bminspec.in_run_stop){
                bminspec.S_Load = 0.0;
                bminspec.T_Load = 0.0;
                bminspec.Sc_Load = 0.0;
                bminspec.Tc_Load = 0.0;
                bminspec.Sh_Load = 0.0;
                bminspec.Th_Load = 0.0;
                if( bminspec.isRecord ){
                    bminspec.message.append( "(C)負荷率＜停止負荷率で停止");
                }
                bminspec.mState = 3;//能力＝０→換気モードとする20130415
            }
        }

        //    System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate+" DBout="+DBout1);

    }


    private static void ventilation_cal(  BMinSpec2015 bminspec ){
        //double heatCapacity=0.0,rCode1 = 0.0,rCode2 = 0.0,rCode3 = 0.0,rCode4 = 0.0,rCode5 = 0.0,rCode6 = 0.0,rCode7 =
0.0,rCode8 = 0.0;
        //double IAra,IAout1,DBout1,XGout1,S_heat,T_heat;
        double HEXrate;

        HEXrate = bminspec.in_EX_S;

        //換気の時は全熱交換器は停止とする
        HEXrate = 0.0;

        //外気と室空気の混合空気の計算
        if( Airswc.isON( bminspec.swcInOA )){//20101212nino
```

```
        if( bminspec.isRecord ){
           bminspec.message.append( "/OAon" );
        }
        bminspec.DBin = (( bminspec.in_Va_S - bminspec.Moa ) * bminspec.DBra + (( bminspec.DBoa - bminspec.DBra ) *
( 1.0 - HEXrate ) + bminspec.DBra ) * bminspec.Moa ) / bminspec.in_Va_S;//20130109
        bminspec.XGin = (( bminspec.in_Va_S - bminspec.Moa ) * bminspec.XGra + (( bminspec.XGoa - bminspec.XGra ) *
( 1.0 - HEXrate ) + bminspec.XGra ) * bminspec.Moa ) / bminspec.in_Va_S;//20130109

        bminspec.qSTHEX = ( bminspec.DBra - bminspec.DBoa ) * HEXrate * bminspec.Moa;
        bminspec.qLTHEX = ( bminspec.XGra - bminspec.XGoa ) * HEXrate * bminspec.Moa * 2501.;
        bminspec.qTTHEX = bminspec.qSTHEX + bminspec.qLTHEX;

     }else{//20101212nino
        if( bminspec.isRecord ){
           bminspec.message.append( "/OAoff" );
        }
        if( bminspec.isOAProcessingUnit ){
           //外気処理専用の場合は　外気//20150319
           bminspec.DBin = bminspec.DBoa;
           bminspec.XGin = bminspec.XGoa;
        }else{
           bminspec.DBin = bminspec.DBra;//20130109
           bminspec.XGin = bminspec.XGra;//20130109
        }
     }
     bminspec.WBin = Psychrometrics.FNWbtx( bminspec.DBin, bminspec.XGin );//20130109
     //    System.out.println("C/DBoa="+DBoa+" "+XGoa+"  ");
     //    System.out.println("C/DBra="+DBra+" "+XGra+"  ");
     //    System.out.println("C/DBin="+DBin+" "+XGin+"  ");
     //    System.out.println("  ");

     //    System.out.println("DBra="+DBra+" "+XGra+"  ");

     bminspec.LPrate = 1.0;

     bminspec.T_C_heat = 0.0;

     bminspec.S_Load = 0.0;
     bminspec.T_Load = 0.0;
     bminspec.Sc_Load = 0.0;
     bminspec.Tc_Load = 0.0;
     bminspec.Sh_Load = 0.0;
     bminspec.Th_Load = 0.0;

     // System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate.getValue());
  }



  private static void cooling_supply(  BMinSpec2015 bminspec ){
     /*
     double IAra,IAout;
     IAra=Psychrometrics.FNH(DBra,XGra);
     IAout=IAra-T_Load_RM/in_Va_S*1000.0;
     DBout=DBra-S_Load_RM/in_Va_S;//乾球温度
     XGout= Psychrometrics.FNXth(DBout, IAout);//絶対湿度
      */
     //******************090725-s
     double IAra,IAout,sRc;
     double SS_rm;

     //IAin = Psychrometrics.FNH( bminspec.DBin, bminspec.XGin );//20130109
     if( bminspec.T_C_heat > 0.0 ){
        sRc = bminspec.Tc_Load_RM / bminspec.T_C_heat;
     }else{
        sRc = 0.0;
     }

     if( bminspec.numCalcSA == 1 ){//20141110BESTEST
        //IAout = IAin - ( bminspec.Tc_Load_RM - bminspec.in_PPEac_S * bminspec.PIDrate.getValue() ) / bminspec.GW *
1000.0;//20130109
        //IAout = IAin - ( bminspec.Tc_Load_RM - bminspec.in_PPEac_S * bminspec.PIDrate.getValue() ) / bminspec.GW *
```

```
1000.0;//20130109
         //SS_rm = bminspec.Sc_Load_RM + ( bminspec.Tc_Load_RM - bminspec.Sc_Load_RM ) * ( 1.0 - sRc ) *
bminspec.saijohatsu / 100.0;
         if( bminspec.isFanOnOff ){
            SS_rm = bminspec.Sc_Load_RM - bminspec.in_PPEac_S * bminspec.PIDrate.getValue();
            if( SS_rm > bminspec.Tc_Load_RM - bminspec.in_PPEac_S * bminspec.PIDrate.getValue() ){
               SS_rm = bminspec.Tc_Load_RM - bminspec.in_PPEac_S * bminspec.PIDrate.getValue();
            }
         }else{
            SS_rm = bminspec.Sc_Load_RM - bminspec.in_PPEac_S;
            if( SS_rm > bminspec.Tc_Load_RM - bminspec.in_PPEac_S ){
               SS_rm = bminspec.Tc_Load_RM - bminspec.in_PPEac_S;
            }
         }
      }else{
         //IAout = IAin - bminspec.Tc_Load_RM / bminspec.GW * 1000.0;//20130109
         SS_rm = bminspec.Sc_Load_RM + ( bminspec.Tc_Load_RM - bminspec.Sc_Load_RM ) * ( 1.0 - sRc ) *
bminspec.saijohatsu / 100.0;

         if( SS_rm > bminspec.Tc_Load_RM ){
            SS_rm = bminspec.Tc_Load_RM;
         }
      }

      //System.out.println( " SS_rm="+SS_rm);

      bminspec.DBout = bminspec.DBin - SS_rm / ( bminspec.GW * 1.006 );//乾球温度//20130109
//    if( bminspec.DBout > bminspec.DBin ){
//       System.out.println( "     66666666666666 sa=" +(bminspec.DBout-bminspec.DBin) );
//       bminspec.DBout = bminspec.DBin;
//    }

      //bminspec.XGout = Psychrometrics.FNXth( bminspec.DBout, IAout);//絶対湿度
      bminspec.XGout = bminspec.XGin - ( bminspec.Tc_Load_RM - bminspec.Sc_Load_RM ) / 2501. / bminspec.GW;
      if( bminspec.XGout > bminspec.XGin ){
         //System.out.println( "     77777777777777 sa="+(bminspec.XGout- bminspec.XGin));
         bminspec.XGout = bminspec.XGin;
      }

      if( bminspec.XGout < 0 ){//20150414
         //System.out.println( "     8888888888 XGout=" + bminspec.XGout + " XGin=" + bminspec.XGin );
         bminspec.XGout = 0;
      }

   // if( bminspec.XGout > bminspec.XGin ){
   //    System.out.println("①bminspec.XGout - bminspec.XGin=" +( bminspec.XGout - bminspec.XGin));
   //    bminspec.XGout = bminspec.XGin;
   // }
      /*
      if(saijohatsu>0.0 && Psychrometrics.FNRhtx(DBin, XGin)<40.0){
         System.out.println("40以下"+(++u40)+"S_Load_RM="+S_Load_RM+" T_Load=_RM"+T_Load_RM+" sRc="+sRc+" SS_rm="+SS_rm);

         System.out.println("   DBout0="+(DBra-S_Load_RM/in_Va_S)+"  DBout1="+(DBra-SS_rm/in_Va_S));
         System.out.println("   XXout0="+Psychrometrics.FNXth((DBra-S_Load_RM/in_Va_S), IAout)+"
XXout1="+Psychrometrics.FNXth((DBra-SS_rm/in_Va_S), IAout));
         System.out.println("   DBin  ="+DBin+"  XGin  ="+XGin+"  RHin  ="+Psychrometrics.FNRhtx(DBin, XGin));
         System.out.println("   DBoa  ="+DBoa+"  XGoa  ="+XGoa+"  RHoa  ="+Psychrometrics.FNRhtx(DBoa, XGoa));
         System.out.println("   DBra  ="+DBra+"  XGra  ="+XGra+"  RHra  ="+Psychrometrics.FNRhtx(DBra, XGra));

      }
       */
      //*****************090725-e
      //    ********************************************************************
      //    System.out.println("DBout="+DBout+" "+XGout+"  ");
   }



   private static void heating_supply(  BMinSpec2015 bminspec ){
      double IAra,IAout;
```

```
bminspec.RHra = Psychrometrics.FNRhtx( bminspec.DBra, bminspec.XGra );//20130109
bminspec.Sh_Load_RM = bminspec.CAPrateH * bminspec.Sh_Load;
bminspec.Th_Load_RM = bminspec.CAPrateH * bminspec.Th_Load;

if( bminspec.isOAProcessingUnit ){
   //外気処理エアコンの場合こっこでは加湿計算はしない
   bminspec.IAin = Psychrometrics.FNH( bminspec.DBin, bminspec.XGin );//20130109
   IAout = bminspec.IAin +bminspec.Th_Load_RM / bminspec.in_Va_S * 1000.0;//20130109
   bminspec.DBout = bminspec.DBin + bminspec.Sh_Load_RM / bminspec.in_Va_S;//乾球温度//20130109
   if( bminspec.in_HUM_mx == 0 ){   //20160223
      //加湿能力＝０の時　絶対湿度変化なし
      bminspec.XGout = bminspec.XGin;//絶対湿度
   }else{
      bminspec.XGout = Psychrometrics.FNXth( bminspec.DBout, IAout );//絶対湿度
   }
   if( bminspec.in_HUM_eff != 0 ){//20110113nino
      bminspec.CW_Wrate = ( bminspec.XGout - bminspec.XGin ) * bminspec.GW / bminspec.in_HUM_eff;//20130109
   }else{
      bminspec.CW_Wrate = bminspec.in_HUM_mx;
   }

//}else if( bminspec.in_HUM_on <= bminspec.RHra || bminspec.in_HUM_mx < 0.00001  ){
}else if( bminspec.Moa <= 0 || bminspec.in_HUM_mx < 0.00001  ){
   //加湿計算はしない
   bminspec.IAin = Psychrometrics.FNH( bminspec.DBin, bminspec.XGin );//20130109
   IAout = bminspec.IAin +bminspec.Th_Load_RM / bminspec.in_Va_S * 1000.0;//20130109
   bminspec.DBout = bminspec.DBin + bminspec.Sh_Load_RM / bminspec.in_Va_S;//乾球温度//20130109
   //bminspec.XGout = Psychrometrics.FNXth( bminspec.DBout, IAout );//絶対湿度
   bminspec.XGout = bminspec.XGin;//20150412
   bminspec.CW_Wrate = 0.0;
   //    System.out.println("1 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
   //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
} else {
   bminspec.IAin = Psychrometrics.FNH( bminspec.DBin, bminspec.XGin );//20130109
   IAout = bminspec.IAin + bminspec.Th_Load_RM / bminspec.in_Va_S * 1000.0;//20130109
   if( bminspec.in_HUM_mx == 0 ){   //20160223
      //加湿能力＝０の時　絶対湿度変化なし
      bminspec.XGout = bminspec.XGin;//絶対湿度
      bminspec.DBout =Psychrometrics.FNDbxh( bminspec.XGout, IAout);
   }else{
      bminspec.DBout = Psychrometrics.FNDbrh( bminspec.in_HUM_rt, IAout);
      bminspec.XGout = Psychrometrics.FNxtr( bminspec.DBout, bminspec.in_HUM_rt);
   }
   if( bminspec.in_HUM_eff != 0 ){//20110113nino
      //bminspec.CW_Wrate = ( bminspec.XGout - bminspec.XGin ) * bminspec.GW / bminspec.in_HUM_eff;//20130109
      //必要加湿量//20200109
      double dRH = bminspec.in_HUM_on - bminspec.RHra;
      double XG = Psychrometrics.FNxtr( bminspec.DBra, bminspec.in_HUM_on + dRH );

      bminspec.CW_Wrate = ( XG - bminspec.XGin ) * bminspec.GW / bminspec.in_HUM_eff;//20130109
      //XGout,DBoutをin_HUM_mxで見直す。
      bminspec.XGout = bminspec.CW_Wrate * bminspec.in_HUM_eff / bminspec.GW+ bminspec.XGin;//20200109
      bminspec.DBout =Psychrometrics.FNDbxh( bminspec.XGout, IAout);


   }else{
      bminspec.CW_Wrate = bminspec.in_HUM_mx;
   }

   //定格加湿量をオーバーするときの処理
   //if( bminspec.CW_Wrate > bminspec.in_HUM_mx ){
   if( bminspec.CW_Wrate >= bminspec.in_HUM_mx ){//20150412
      //XGout,DBoutをin_HUM_mxで見直す。
      //bminspec.XGout = bminspec.in_HUM_mx / bminspec.GW+ bminspec.XGin;//20130109
      bminspec.XGout = bminspec.in_HUM_mx * bminspec.in_HUM_eff / bminspec.GW+ bminspec.XGin;//20200109
      bminspec.DBout =Psychrometrics.FNDbxh( bminspec.XGout, IAout);
      bminspec.CW_Wrate = bminspec.in_HUM_mx;
   }
   if( bminspec.XGout < bminspec.XGin){//20130109
      bminspec.IAin = Psychrometrics.FNH( bminspec.DBin, bminspec.XGin );//20130109
      IAout = bminspec.IAin + bminspec.Th_Load_RM / bminspec.in_Va_S * 1000.0;//20130109
      bminspec.DBout = bminspec.DBin + bminspec.Sh_Load_RM / bminspec.in_Va_S;//乾球温度//20130109
```

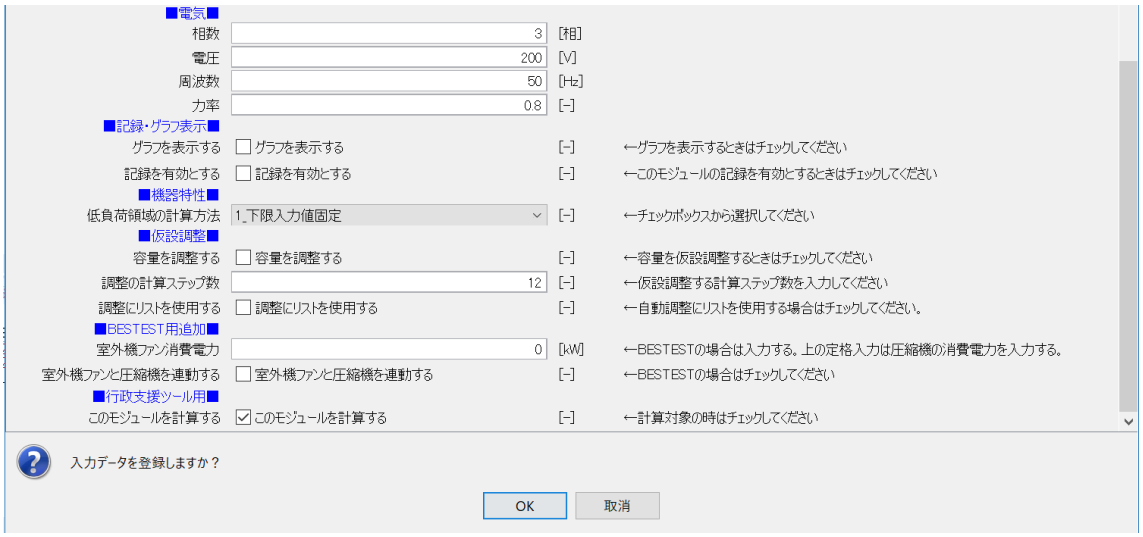
```
            //bminspec.XGout = Psychrometrics.FNXth( bminspec.DBout, IAout);//絶対湿度
            bminspec.XGout = bminspec.XGin;//20150412
            bminspec.CW_Wrate = 0.0;
        }
        //    System.out.println("2 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
        //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
    }
    //    bminspec.watInCW.setFlowRate(CW_Wrate);
}


private static void ventilation_supply(  BMinSpec2015 bminspec ){

    bminspec.Sh_Load_RM = 0.0;
    bminspec.Th_Load_RM = 0.0;

    bminspec.DBout = bminspec.DBin;//乾球温度//20130109
    bminspec.XGout = bminspec.XGin;//絶対湿度
    bminspec.CW_Wrate = 0.0;

}



private static double WB_cal(double ddb,double xx){
    double wwb,x1,de=1;
    int kk=-1,j=0;
    wwb = ddb;

    do{
        j++;
        x1 = Psychrometrics.FNXtw( ddb, wwb );
        if( Math.abs( x1 - xx ) < 0.000003 ) {
            //    System.out.println(" j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
            return wwb;
        }
        if( x1 < xx ) {
            if( kk == -1 ){
                de = de / 2.0;
            }
            wwb = wwb + de;
            kk = 1;
        }
            if( x1 >= xx ) {
            if( kk == 1 ){
                de = de / 2.0;
            }
            wwb = wwb - de;
            kk = -1;
        }

    }while( j < 1000 );
    //    System.out.println("*j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
    return wwb;
}

}
```

「BMo EHP 室外機 201012」（場所：設備 2015／個別分散 2015／）

モジュール名	BMo EHP 室外機 201012
クラス	BuillMultiEHPOut_SModule20101212

（1）入力画面

・スペック



（2）モジュールの概要

ビル用マルチの EHP 室外機モジュールである。

運転能力は、接続された室内機の要求処理負荷の合計値にて決定する。要求処理負荷の合計値が室外機の運転能力を超え能力不足となる場合は、個々の室内機の要求処理負荷を負荷率で割り引いた処理負荷を返す。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

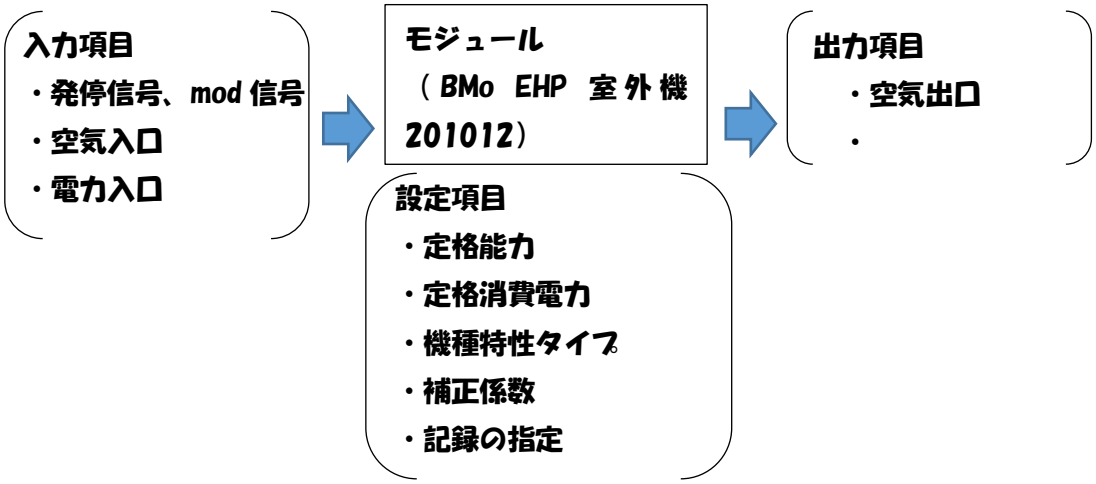


図１　モジュールの設定項目・入力項目・出力項目

（４）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	－	[-]	－	－		
1	機器番号	String	m_no			－	－		
2	機器種別	String	m_ty	0_201707_EHP_BM_標準_冷暖切替、3_201707_EHP_店舗用_冷暖切替、0_201303_EHP_BM_標準_冷暖切替、1_201303_EHP_BM_標準_冷暖切替_寒冷地対応、2_201303_EHP_BM_標準_冷暖同時、3_201303_EHP_店舗用_冷暖切替、4_201303_EHP_店舗用_冷暖切替_寒冷地対応、5_201303_EHP_設備用_冷暖切替、6_201303_EHP_BM_高顕熱型_	[-]	－	－		←チェックボックスから選択してください

				冷暖切替、7_201303_EHP_BM_外調機用_冷暖切替、8_201303_EHP_BM_高顕熱型_冷暖同時、0_EHP_BM_標準_冷暖切替200811、1_EHP_BM_標準_冷暖切替_寒冷地対応200903、2_EHP_BM_標準_冷暖同時200912、3_EHP_店舗用_冷暖切替200903、4_EHP_店舗用_冷暖切替_寒冷地対応200906、5_EHP_設備用_冷暖切替200903、Ua_ユーザー特性_BM_冷暖切替、Ub_ユーザー特性_BM_冷暖同時、Uc_ユーザー特性_BM_高顕熱型、					

				Xa_2014_BESTEST_EHP_BM、Xc_2014_BESTEST_EHP_BM、Xe_2014_BESTEST_EHP_BM、Xe_2014_BESTEST_EHP_BMts、Xf_2014_BESTEST_EHP_BM					
3	機器型式	String	m_kt	#File 、BMList¥¥BMEHPoutListA.csv		―	―		←機器型式指定した場合は、以下の「定格能力など」の入力は無視します
4	■ユーザー機器特性■					―	―		
5	機器特性ユーザー定義ファイル名	String	uTokusei		[-]	―	―		←機器特性ユーザー定義ファイル名をフルパスで入力してください。
6	■定格能力など■					―	―		←以下は１台当たりの仕様を入力してください
7	定格冷房能力	double	out_Qc_S	100	[kW]	―	0		
8	中間冷房能力	double	out_Qc_C	0	[kW]	―	0		←任意入力項目です
9	定格暖房能力	double	out_Qh_S	100	[kW]	―	0		
10	中間暖房能力	double	out_Qh_C	0	[kW]	―	0		←任意入力項目です
11	低温暖房能力	double	out_Qh_L	0	[kW]	―	0		←任意入力項目です
12	定格冷房入力(電力)	double	out_PPEc_S	30	[kW]	―	0		
13	中間冷房入力(電力)	double	out_PPEc_C	0	[kW]	―	0		←任意入力項目です
14	定格暖房入力(電力)	double	out_PPEh_S	30	[kW]	―	0		
15	中間暖房入力(電力)	double	out_PPEh_C	0	[kW]	―	0		←任意入力項目です
16	低温暖房入力(電力)	double	out_PPEh_L	0	[kW]	―	0		←任意入力項目です

17	風量	double	fRate_airEA	3600	[m3/h(a)]	―	0		←airOutEA の温度の計算に使用します
18	機器起動停止負荷率	double	in_run_stop	30	[%]	100	0		←部分負荷率を入力してください
19	冷房能力補正係数	double	out_kQc_S	1	[-]	―	0		←定格冷房能力を補正します
20	冷房入力補正係数	double	out_kPPEc_S	1	[-]	―	0		←定格冷房入力を補正します
21	暖房能力補正係数	double	out_kQh_S	1	[-]	―	0		←定格暖房能力を補正します
22	暖房入力補正係数	double	out_kPPEh_S	1	[-]	―	0		←定格暖房入力を補正します
23	■電気■					―	―		
24	相数	int	phase	3	[相]	―	1		
25	電圧	double	voltage	200	[V]	―	100		
26	周波数	double	frequency	50	[Hz]	60	50		
27	力率	double	powerFactor	0.8	[-]	1	0		
28	■記録・グラフ表示■					―	―		
29	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
30	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください
31	■機器特性■					―	―		
32	低負荷領域の計算方法	String	calcTypeLowerRangeLoad	0_発停運転、1_下限入力値固定、2_下限 COP 値固定、3_下限入力値と中間切片	[-]	―	―		←チェックボックスから選択してください
33	■仮設調整■					―	―		
34	容量を調整する	boolean	isAdjust2012	FALSE	[-]	―	―		←容量を仮設調整するときはチェックしてください

35	調整の計算ステップ数	int	numAdjustSteps	12	[-]	―	1		←仮設調整する計算ステップ数を入力してください
36	調整にリストを使用する	boolean	isUseAdjustList	FALSE	[-]	―	―		←自動調整にリストを使用する場合はチェックしてください。
37	<html><font color=blue> ■BESTEST 用追加■					―	―		
38	室外機ファン消費電力	double	out_PPEfan	0	[kW]	―	0		←BESTEST の場合は入力する。上の定格入力は圧縮機の消費電力を入力する。
39	室外機ファンと圧縮機を連動する	boolean	isFanOnOff	FALSE	[-]	―	―		←BESTEST の場合はチェックしてください
40	■行政支援ツール用■					―	―		
41	このモジュールを計算する	boolean	isCalc	TRUE	[-]	―	―		←計算対象の時はチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	外気入口	L0_airInOA	airInOA	-	状態	空気	入口	
5	排気出口	L0_airOutEA	airOutEA	-	状態	空気	出口	
6	冷媒入口	L0_valInLine	valInLine	-	値	値	入口	
7	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	室外機 Message#-#-	メッセージ	—	メッセージ
2	室内機冷却要求顕熱量合計#W#熱量	室内機冷却要求顕熱量合計	W	My
3	室内機加熱要求顕熱量合計#W#熱量	室内機加熱要求顕熱量合計	W	My
4	室内機冷却要求全熱量合計#W#熱量	室内機冷却要求全熱量合計	W	My
5	室内機加熱要求全熱量合計#W#熱量	室内機加熱要求全熱量合計	W	My
6	室内機冷却能力補正比率	室内機冷却能力補正比率	—	My
7	室内機加熱能力補正比率	室内機加熱能力補正比率	—	My
8	室外機消費電力#W#消費電力	室外機消費電力	W	エネルギー消費
9	室外機圧縮機消費電力#W#消費電力	室外機圧縮機消費電力	W	エネルギー消費
10	室外機ファン消費電力#W#消費電力	室外機ファン消費電力	W	エネルギー消費
11	室外機冷却可能熱量/最大#W#熱量	室外機冷却可能熱量/最大	W	My
12	室外機加熱可能熱量/最大#W#熱量	室外機加熱可能熱量/最大	W	My
13	室外機冷却実処理熱量/現状#W#熱量	室外機冷却実処理熱量/現状	W	負荷
14	室外機加熱実処理熱量/現状#W#熱量	室外機加熱実処理熱量/現状	W	負荷
15	処理全熱量合計負荷#W#-	処理全熱量合計負荷	W	My
16	室外機外気入口 DB#℃#-	室外機外気入口 DB	℃	入口
17	室外機外気 WB#℃#-	室外機外気 WB	℃	入口
18	室外機外気流量#g/s#-	室外機外気流量	g/s	入口
19	室外機排気 DB#℃#-	室外機排気 DB	℃	出口
20	室外機排気 WB#℃#-	室外機排気 WB	℃	出口
21	室外機排気流量#g/s#-	室外機排気流量	g/s	出口
22	室外機調整冷却能力#W#熱量	室外機調整冷却能力	W	調整
23	室外機調整加熱能力#W#熱量	室外機調整加熱能力	W	調整
24	室外機調整冷却特性補正#-#-	室外機調整冷却特性補正	—	調整
25	室外機調整加熱特性補正#-#-	室外機調整加熱特性補正	—	調整

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

//import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Map;
import java.util.Set;

import jp.or.ibec.best.DO.BM_EHPdata;
import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.client.bestgui.file.service.FileConstants;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.essentials.utility.Psychrometrics;

/**
 * @author SUGANAGA 2008/04/07 /20080428 NINOMIYA/20101212
 * ビルマルチ：室外機クラス
 *
 * 冷暖同時追加/20101212nino
 *
 * 20130327　機器特性201303　を追加
 */
public class BuillMultiEHPOut_SModule20101212 extends AbstractBestModule implements
IBestMetaModule  {

   /**
    * 論理アドレス
    */
   private final String moduleName="(BuillMultiEHPOut_SModule20101212)";
   private final String S_NODE_airInOA    = "L0_airInOA";//外気
   private final String S_NODE_airOutEA   = "L0_airOutEA";//排気
   private final String S_NODE_valInLine  = "L0_valInLine";//単線
   private final String S_NODE_eleIn      = "L0_eleIn";//電力
   private final String C_NODE_swcIn = "L1_swcIn";//運転状態：off/on
   private final String C_NODE_modIn = "L1_modIn";//空調モード：停止/冷却/加熱
   private final String R_NODE = "L2_recOut";
   private Map<String,BM_EHPdata> rmlinemap=null;

   /**
    * 外部定義
    */
   private final String SPEC_name          = "名称";

   private final String SPEC_isCalc = "このモジュールを計算する";//行政ツール用

   private final String SPEC_uTokusei      = "機器特性UserFile名";
   private final String SPEC_m_no          = "機器番号";
   private final String SPEC_m_ty          = "機器種別";
   private final String SPEC_m_kt          = "機器型式";
   private final String SPEC_out_Qc_S      = "定格冷房能力[W]";
   private final String SPEC_out_Qc_C      = "中間冷房能力[W]";
   private final String SPEC_out_PPEc_S    = "定格冷房入力(電力)[W]";
   private final String SPEC_out_PPEc_C    = "中間冷房入力(電力)[W]";
   private final String SPEC_out_Qh_S      = "定格暖房能力[W]";
   private final String SPEC_out_Qh_C      = "中間暖房能力[W]";
   private final String SPEC_out_Qh_L      = "低温暖房能力[W]";
   private final String SPEC_out_PPEh_S    = "定格暖房入力(電力)[W]";
```

```
    private final String SPEC_out_PPEh_C    = "中間暖房入力(電力)[W]";
    private final String SPEC_out_PPEh_L    = "低温暖房入力(電力)[W]";

    private final String SPEC_out_kQc_S    = "冷房能力補正係数[-]";
    private final String SPEC_out_kPPEc_S  = "冷房入力補正係数[-]";
    private final String SPEC_out_kQh_S    = "暖房能力補正係数[-]";
    private final String SPEC_out_kPPEh_S  = "暖房入力補正係数[-]";

    private final String SPEC_out_PPEfan    = "室外機ファン消費電力[W]";
    private final String SPEC_isFanOnOff    = "室外機ファン連動";

/*
    private final String SPEC_out_CLEa_R    = "クランクケースヒータ(運転時)[W]";
    private final String SPEC_out_CLEa_S    = "クランクケースヒータ(停止時)[W]";
    private final String SPEC_out_TKEa_R    = "定格待機電力(運転時)[W]";
    private final String SPEC_out_TKEa_T    = "定格待機電力(待機時)[W]";
    private final String SPEC_out_TKEa_S    = "定格待機電力(停止時)[W]";
*/
    private final String SPEC_fRate_airEA   = "風量[g/s]";

    private final String SPEC_in_run_stop  = "機器起動停止負荷率[-]";
    private final String SPEC_Phase       = "相数[-]";
    private final String SPEC_Voltage     = "電圧[V]";
    private final String SPEC_Frequency   = "周波数[Hz]";
    private final String SPEC_PowerFactor = "力率[-]";
    //
    private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
    private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

    private final String SPEC_CalcTypeLowerRangeLoad = "低負荷領域の計算方法";

    private final String SPEC_isAdjust2012   = "容量を調整する";
    private final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";
    private final String SPEC_isUseAdjustList = "is調整リストを使用する";

    private StringBuffer message= new StringBuffer();

    private BestAir airInOA  = null;
    private BestAir airOutEA = null;
    private BestElectricity eleIn = null;  //電力
    private Double CAPrateC =  new Double( 1 );//室内機能力補正比率[-]冷房側
    private Double CAPrateH =  new Double( 1 );//室内機能力補正比率[-]暖房側
    private BM_EHPdata RMdata = null;   //電力
    private PACDBManager pacDB=null;

    private int modIn  ;
    private int swcIn ;
    private int mState;  //停止、冷房、暖房フラグ

    private boolean isUserTokusei = false;//ユーザー定義の機器特性の場合=true
    private boolean isCOOLandHEATout = false;//冷暖同時機種の場合=true
    private boolean isHighSH = false;//高顕熱型の場合=true
    private boolean isBESTESTtable = false;//BESTESTの特性tableの場合=true
    private boolean is2018SinseiCHFree = false;//2018冷暖同時誘導基準申請計算モード
    private int dbflg=0;


    /**
     * 変数 (外部定義に対応)
     */

    private String name             ;// "機器名称";

    private boolean isCalc = true;//このモジュールを計算する

    private String uTokusei = null;//     = "機器特性UserFile名";

    private String m_no             ;//= "機器番号";
    private String m_ty             ;//= "機器種別";
    private String m_kt             ;//= "機器型式";
    private double out_Qc_S      ;//= "定格冷房能力";//kW
    private double out_Qc_C      ;//= "中間冷房能力";//kW
```

```
    private double out_PPEc_S    ;//= "定格冷房入力(電力)";//kW
    private double out_PPEc_C    ;//= "中間冷房入力(電力)";//kW
    private double out_Qh_S      ;//= "定格暖房能力";//kW
    private double out_Qh_C      ;//= "中間暖房能力";//kW
    private double out_Qh_L      ;//= "低温暖房能力";//kW
    private double out_PPEh_S    ;//= "定格暖房入力(電力)";//kW
    private double out_PPEh_C    ;//= "中間暖房入力(電力)";//kW
    private double out_PPEh_L    ;//= "低温暖房入力(電力)";//kW

    private double out_kQc_S;//   = "冷房能力補正係数[-]";
    private double out_kPPEc_S;//    = "冷房入力補正係数[-]";
    private double out_kQh_S;//   = "暖房能力補正係数[-]";
    private double out_kPPEh_S;//    = "暖房入力補正係数[-]";

    private double out_PPEfan;
    private boolean isFanOnOff = false;// "室外機ファン連動";

/*
    private double out_CLEa_R=0.0    ;//= "クランクケースヒータ(運転時)";//W
    private double out_CLEa_S=0.0    ;//= "クランクケースヒータ(停止時)";//W
    private double out_TKEa_R =0.0   ;//= "定格待機電力(運転時)";//W
    private double out_TKEa_T =0.0   ;//= "定格待機電力(待機時)";//W
    private double out_TKEa_S =0.0  ;//= "定格待機電力(停止時)";//W
*/
    private double fRate_airEA ;//風量[g/s]

    private double in_run_stop    ;//= "機器起動停止負荷率";//[%]
    private int phase;         //相数[-]
    private double voltage;    //電圧[V]
    private double frequency;  //周波数[Hz]
    private double powerFactor;   //力率[-]

    private double Lp;         //冷媒管長さ[m]
    private double Lh;         //室内機/室外機高低差[m]
    private boolean isGVisible = false;//このグラフを表示する=true
    private boolean isRecord   = false;//記録を有効とする=true

    private String[] rmnm=new String[50];

    /**
     * 出力 (建物)
     */

    private final String RECORD_message = "室外機Message#-#-";
    private final String RECORD_ID1c    = "室内機冷却要求顕熱量合計#W#熱量";
    private final String RECORD_ID1h    = "室内機加熱要求顕熱量合計#W#熱量";
    private final String RECORD_ID2c    = "室内機冷却要求全熱量合計#W#熱量";
    private final String RECORD_ID2h    = "室内機加熱要求全熱量合計#W#熱量";
    private final String RECORD_ID3c    = "室内機冷却能力補正比率";
    private final String RECORD_ID3h    = "室内機加熱能力補正比率";
    private final String RECORD_ID4     = "室外機消費電力#W#消費電力";
    private final String RECORD_PPEcmp = "室外機圧縮機消費電力#W#消費電力";
    private final String RECORD_PPEfan = "室外機ファン消費電力#W#消費電力";
    private final String RECORD_ID5c    = "室外機冷却可能熱量/最大#W#熱量";
    private final String RECORD_ID5h    = "室外機加熱可能熱量/最大#W#熱量";
    private final String RECORD_ID6c    = "室外機冷却実処理熱量/現状#W#熱量";
    private final String RECORD_ID6h    = "室外機加熱実処理熱量/現状#W#熱量";
    private final String RECORD_qT      = "処理全熱量合計負荷#W#-";

    private final String RECORD_DBairInHS  = "室外機外気入口DB#℃#-";
    private final String RECORD_WBairInHS  = "室外機外気WB#℃#-";
    private final String RECORD_M_airInHS  = "室外機外気流量#g/s#-";
    private final String RECORD_DBairOutHS = "室外機排気DB#℃#-";
    private final String RECORD_WBairOutHS = "室外機排気WB#℃#-";
    private final String RECORD_M_airOutHS = "室外機排気流量#g/s#-";

    private final String RECORD_out_Qc_Adjust = "室外機調整冷却能力#W#熱量";
    private final String RECORD_out_Qh_Adjust = "室外機調整加熱能力#W#熱量";
    private final String RECORD_out_Qc_Adjusthosei = "室外機調整冷却特性補正#-#-";
    private final String RECORD_out_Qh_Adjusthosei = "室外機調整加熱特性補正#-#-";

    /**
```

```
 * その他変数
 */

private double Rc;              //部分負荷率　冷房側
private double Rh;              //部分負荷率　暖房側

private double DBoa;          //外気乾球温度[℃]
private double WBoa;          //外気湿球温度[℃]
private double XGoa;          //外気絶対湿度[g/gD.A.]

private double DBra;          //室内機吸込乾球温度[℃]
private double WBra;          //室内機吸込湿球温度[℃]
private double XGra;          //室内機吸込絶対湿度[g/gD.A.]

private double Sc_Loadout;//室外機処理可能熱量（冷却全熱）[W]
private double Sh_Loadout;//室外機処理可能熱量（加熱全熱）[W]
private double Tc_Loadout;//室外機実処理熱量（冷却全熱）[W]
private double Th_Loadout;//室外機実処理熱量（加熱全熱）[W]
private double allSc_heat;//室内機処理熱量（冷却顕熱）[W]
private double allSh_heat;//室内機処理熱量（加熱顕熱）[W]
private double allTc_heat;//室内機処理熱量（冷却全熱）[W]
private double allTh_heat;//室内機処理熱量（加熱全熱）[W]

private double PPE;  //室外機消費電力[W]//
private double PPEcmp;  //室外機消費電力[W]//圧縮機
private double PPEfan;  //室外機消費電力[W]//ファン
private double E_PPE;//室外機無効電力[]

private int num;
private String kiki_file;
private String path="EHP";
private String[] filenames= new String[1];
private String equipmentName;

//private double DBra_main;
//private double WBra_main;
private double Lratemin;
private double fmc;
private double pmc;
private double fmh;
private double pmh;

private boolean isSelectSPECList = false;//機器SPECListの機器を指定している=true

//グラフ表示など
private GraphJFrameBuilMultiOut_S20101212 gBMout = null;

//低負荷領域の計算方法
private String calcTypeLowerRangeLoad = null;//"低負荷領域の計算方法";
private int num_calcTypeLowerRangeLoad;//"低負荷領域の計算方法";

//仮設調整モード
//仮設調整モード2012
private boolean isAdjust2012  = false;//true="台数を調整する";//
//private boolean isAdjustMode = true;//20110517nino 空衛学会OS論文検討用＝true

private double out_daiAdjust = 1;//室外機の調整台数
private boolean isUseAdjustList = false;//true="リストで調整する";
private   int numList = 0;

private double[] adjustQcListkW = {14.0, 16.0, 22.4, 28.0, 33.5, 40.0, 45.0, 50.0, 56.0, 61.5, 67.0, 73.0, 77.5, 85.0,
90.0, 95.0, 100.0, 106.0, 112.0, 118.0, 122.0, 128.0, 136.0, 140.0, 145.0, 150.0  };//
private double[] adjustPPEcListkW = {3.64, 4.47, 6.34, 9.51, 9.87, 11.8, 14.9, 17.2, 17.3, 21.8, 19.8, 24.5, 24.2,
26.7, 29.8, 32.1, 31.6, 36.4, 34.7, 39.4, 38.8, 40.4, 45.4, 47.0, 49.3, 51.6 };//
private double[] adjustQhListkW = {16.0, 18.0, 25.0, 31.5, 37.5, 45.0, 50.0, 56.0, 63.0, 69.0, 77.5, 82.5, 90.0, 95.0,
100.0, 106.0, 112.0, 118.0, 125.0, 132.0, 140.0, 145.0, 150.0, 155.0, 160.0, 165.0 };//
private double[] adjustPPEhListkW = {3.67, 4.32, 6.25, 9.76, 13.1, 14.9, 16.3, 15.7, 21.3, 25.7, 28.0, 26.7, 31.1,
31.2, 32.6, 32.0, 37.0, 41.3, 42.5, 42.7, 47.4, 47.5, 48.9, 47.7, 46.6, 45.5};//
private double[] adjustfRateListm3min = {104, 119, 172, 199, 222, 243, 305, 387, 390, 422, 444, 504, 527, 548, 610,
692, 695, 727, 749, 809, 832, 853, 915, 997, 1079, 1161 };//

private double out_Qc_Adjust;//= "調整冷却能力";
```

```
    private double out_Qh_Adjust;//= "調整加熱能力";
    private double out_Qc_Adjusthosei;//= "調整冷却特性補正";
    private double out_Qh_Adjusthosei;//= "調整加熱特性補正";
    private double fRate_airEAc ;//冷却時風量[g/s]
    private double fRate_airEAh ;//加熱時風量[g/s]

    private int numAdjustSteps;//"調整の計算ステップ数";
    private LinkedList<Double> rcList = null;//必要台数
    private LinkedList<Double> rhList = null;//必要台数
    private double max_rcAdjust = 1.;
    private double max_rhAdjust = 1.;
    private double ave_rcAdjust = 0;
    private double ave_rhAdjust = 0;
    private double out_rcAdjust = 1.;//= "調整率冷房";
    private double out_rhAdjust = 1.;//= "調整率暖房";
// private double in_run_stop_Num;//= "機器起動停止負荷率"台数補正;

    private double out_Qc_S_1      ;//= "定格冷房能力";//kW
    private double out_Qc_C_1      ;//= "中間冷房能力";//kW
    private double out_PPEc_S_1    ;//= "定格冷房入力(電力)";//kW
    private double out_PPEc_C_1    ;//= "中間冷房入力(電力)";//kW
    private double out_Qh_S_1      ;//= "定格暖房能力";//kW
    private double out_Qh_C_1      ;//= "中間暖房能力";//kW
    private double out_Qh_L_1      ;//= "低温暖房能力";//kW
    private double out_PPEh_S_1    ;//= "定格暖房入力(電力)";//kW
    private double out_PPEh_C_1    ;//= "中間暖房入力(電力)";//kW
    private double out_PPEh_L_1    ;//= "低温暖房入力(電力)";//kW


    @Override
    public void setProfile(BestSpecs spec) {
        //    外部定義項目取得
        if(spec == null) {
            return;
        }
        Map<String,String> map=spec.getSpec();
        if(map == null) {
            return;
        }

        //isCalcを取得
        this.isCalc = spec.getSpecValue( this.SPEC_isCalc, true );

        // 機器名称
        this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

        // SPEC_uTokusei      = "機器特性UserFile名";
        this.uTokusei = spec.getSpecValue( this.SPEC_uTokusei, "" );

        // 機器番号
        this.m_no = spec.getSpecValue( this.SPEC_m_no, "" );
        if(null!=map.get(this.SPEC_m_no)){
            this.m_no = (String)map.get(this.SPEC_m_no);
        } else {
            //System.out.println("(W)SPEC_機器番号がありません");
        }

        //  機器種別  0_標準型、1_寒冷地型、2_店舗用、3_設備用
        //this.m_ty = spec.getSpecValue( this.SPEC_m_ty, "0_201303_EHP_BM_標準_冷暖切替" );
        this.m_ty = spec.getSpecValue( this.SPEC_m_ty, "0_201707_EHP_BM_標準_冷暖切替" );//20170707標準機を変更
        if( this.m_ty == null ){
            this.message.append( "(W)機器種別の指定がない→標準型");
            this.m_ty = "0_201707_EHP_BM_標準_冷暖切替";
        }

        //"0_201303_EHP_BM_標準_冷暖切替"
        //this.kiki_file = "EHP_BM_Standard201303";
        this.kiki_file = "EHP_BM_Standard201707";//20170707標準特性を変更

        //機器特性追加201508  ユーザー定義の機器特性ファイルを使用する
        if( this.m_ty.equals("Ua_ユーザー特性_BM_冷暖切替")){
            this.isUserTokusei = true;
```

```
    //this.path = "";
    this.kiki_file = this.uTokusei;
    this.isCOOLandHEATout = false;
    this.isHighSH = false;
}
if( this.m_ty.equals("Ub_ユーザー特性_BM_冷暖同時")){
    this.isUserTokusei = true;
    //this.path = "";
    this.kiki_file = this.uTokusei;
    this.isCOOLandHEATout = true;
    this.isHighSH = false;
}
if( this.m_ty.equals("Uc_ユーザー特性_BM_高顕熱型")){
    this.isUserTokusei = true;
    //this.path = "";
    this.kiki_file = this.uTokusei;
    this.isCOOLandHEATout = false;
    this.isHighSH = true;
}


//機器特性201303追加
if( this.m_ty.equals("Xa_2014_BESTEST_EHP_BM")){
    this.kiki_file = "BESTEST_EHP_BM_2014table5-26a";
    this.isCOOLandHEATout = false;
    this.isHighSH = true;
}
if( this.m_ty.equals("Xc_2014_BESTEST_EHP_BM")){
    this.kiki_file = "BESTEST_EHP_BM_2014table5-26c";
    this.isCOOLandHEATout = false;
    this.isHighSH = true;
}
if( this.m_ty.equals("Xe_2014_BESTEST_EHP_BM")){
    this.kiki_file = "BESTEST_EHP_BM_2014table5-26e";
    this.isCOOLandHEATout = false;
    this.isHighSH = true;
}
if( this.m_ty.equals("Xe_2014_BESTEST_EHP_BMts")){
    this.kiki_file = "BESTEST_EHP_BM_2014table5-26eTS";
    this.isCOOLandHEATout = false;
    this.isBESTESTtable = true;
}
if( this.m_ty.equals("Xf_2014_BESTEST_EHP_BM")){
    this.kiki_file = "BESTEST_EHP_BM_2014table5-31a";
    this.isCOOLandHEATout = false;
    this.isBESTESTtable = true;
}
//20170707特性追加
if( this.m_ty.equals("0_201707_EHP_BM_標準_冷暖切替")){
    this.kiki_file = "EHP_BM_Standard201707";
    this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("3_201707_EHP_店舗用_冷暖切替")){
    this.kiki_file = "EHP_Shop201707";
    this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("0_201303_EHP_BM_標準_冷暖切替")){
    this.kiki_file = "EHP_BM_Standard201303";
    this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("1_201303_EHP_BM_標準_冷暖切替_寒冷地対応")){
    this.kiki_file = "EHP_BM_Standard_ColdArea201303";
    this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("2_201303_EHP_BM_標準_冷暖同時")){
    this.kiki_file = "EHP_BM_Standard_CHFree201303";
    this.isCOOLandHEATout = true;
}
if( this.m_ty.equals("2_201303_EHP_BM_標準_冷暖同時2018申請") ){//20181017
    this.kiki_file = "EHP_BM_Standard_CHFree201303";
    this.isCOOLandHEATout = true;
    this.is2018SinseiCHFree = true;
```

```
}
if( this.m_ty.equals("3_201303_EHP_店舗用_冷暖切替")){
   this.kiki_file = "EHP_Shop201303";
   this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("4_201303_EHP_店舗用_冷暖切替_寒冷地対応")){
   this.kiki_file = "EHP_Shop_ColdArea201303";
   this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("5_201303_EHP_設備用_冷暖切替")){
   this.kiki_file = "EHP_Facilities_201303";
   this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("6_201303_EHP_BM_高顕熱型_冷暖切替")){
   this.kiki_file = "EHP_BM_HighSH201303";
   this.isCOOLandHEATout = false;
   this.isHighSH = true;
}
if( this.m_ty.equals("6_201503_EHP_BM_D_VRVX")){
   this.kiki_file = "EHP_BM_D_VRVX201503";
   this.isCOOLandHEATout = false;
   this.isHighSH = true;
}
if( this.m_ty.equals("7_201303_EHP_BM_外調機用_冷暖切替")){
   this.kiki_file = "EHP_OAProcessingUnit201303";
   this.isCOOLandHEATout = false;
}
//EHP_BM_HighSH_CHFree201303//20181009
if( this.m_ty.equals("8_201303_EHP_BM_高顕熱型_冷暖同時")){
   //冷房、暖房は高顕熱型特性、冷暖同時は標準型の冷暖同時特性
   this.kiki_file = "EHP_BM_HighSH_CHFree201303";
   this.isCOOLandHEATout = true;
   this.isHighSH = true;
}
if( this.m_ty.equals("8_201303_EHP_BM_高顕熱型_冷暖同時2018申請")){//20181017
   //冷房、暖房は高顕熱型特性、冷暖同時は標準型の冷暖同時特性
   this.kiki_file = "EHP_BM_HighSH_CHFree201303";
   this.isCOOLandHEATout = true;
   this.isHighSH = true;
   this.is2018SinseiCHFree = true;
}
//
if( this.m_ty.equals("0_EHP_BM_標準_冷暖切替200811")){
   this.kiki_file = "EHP_BM_Standard200908";
   this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("1_EHP_BM_標準_冷暖切替_寒冷地対応200908")){
   this.kiki_file = "EHP_BM_Standard_ColdArea200908";
   this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("1_EHP_BM_標準_冷暖切替_寒冷地対応200903")){
   this.kiki_file = "EHP_BM_Standard_ColdArea200908";
   this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("2_EHP_BM_標準_冷暖同時200912")){
   this.kiki_file = "EHP_BM_Standard_CHFree200912";
   this.isCOOLandHEATout = true;
}
if( this.m_ty.equals("3_EHP_店舗用_冷暖切替200903") || this.m_ty.equals("3_EHP_店舗用_冷暖切替200908")){
   this.kiki_file = "EHP_Shop200908";
   this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("4_EHP_店舗用_冷暖切替_寒冷地対応200906")){
   this.kiki_file = "EHP_Shop_ColdArea200906";
   this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("5_EHP_設備用_冷暖切替200908")){
   this.kiki_file = "EHP_Facilities_200908";
   this.isCOOLandHEATout = false;
}
if( this.m_ty.equals("5_EHP_設備用_冷暖切替200903")){//20130131
   this.kiki_file = "EHP_Facilities_200908";
```

```
    this.isCOOLandHEATout = false;
}

// 機器型式
this.m_kt = spec.getSpecValue( this.SPEC_m_kt, "" );

FileReadBMSpecModule20110221 bmSpecFR = null;

//機器specの取り込み
if( this.m_kt == null || this.m_kt.equals( "" )){
    //機器型番指定が無い時は、ダイアログ入力値を使用する
}else{
    //機器型番指定の時は、外部SPECListのデータを使用する
    this.isSelectSPECList = true;
    //BM機器SPECListA　のデータの取込み
      String filepathName = FileConstants.DIR_SYSTEM + "/BMList/BMEHPoutSPECListA.csv";
    bmSpecFR = new FileReadBMSpecModule20110221( filepathName );
}

//PerformanceCurves
//Class1
//Class2
//Class3
//MakerName
//SiriesName
//ModelCode
//OutdoorUnitCode
//RefrigerantType
//Fuel
//FrequencyEleIn
//NCCapacity
//NCEleInputMain
//NCEleInputSub
//NCGasInput
//NCOilInput
//NCTairIn
//NCTairOut
//NCFRair
//NCPLossairIn
//NCTwatInCD
//NCTwatOutCD
//NCFRwatInCD
//NCLossPwatInCD
//NHCapacity
//NHEleInputMain
//NHEleInputSub
//NHGasInput
//NHOilInput
//NHTairIn
//NHTairOut
//NHFRair
//NHPLossairIn
//NHTwatInCD
//NHTwatOutCD
//NHFRwatInCD
//NHLossPwatInCD
//RestartDelayTime
//SubPreOpeTime
//SubAftOpeTime
//note


//SPEC_out_kQc_S  = "冷房能力補正係数[-]";
this.out_kQc_S = spec.getSpecValue( this.SPEC_out_kQc_S, 1.0 );
//SPEC_out_kPPEc_S   = "冷房入力補正係数[-]";
this.out_kPPEc_S = spec.getSpecValue( this.SPEC_out_kPPEc_S, 1.0 );
//SPEC_out_kQh_S  = "暖房能力補正係数[-]";
this.out_kQh_S = spec.getSpecValue( this.SPEC_out_kQh_S, 1.0 );
//SPEC_out_kPPEh_S   = "暖房入力補正係数[-]";
this.out_kPPEh_S = spec.getSpecValue( this.SPEC_out_kPPEh_S, 1.0 );

// 定格冷房能力
```

```
this.out_Qc_S = spec.getSpecValue( this.SPEC_out_Qc_S, 0. );

if( this.isSelectSPECList ){
   this.out_Qc_S = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NCCapacity" ) ) * 1000.;//kW->W
}
this.out_Qc_S *= this.out_kQc_S;

// 中間冷房能力
this.out_Qc_C = spec.getSpecValue( this.SPEC_out_Qc_C, 0. );

if( this.isSelectSPECList ){
   this.out_Qc_C = 0.;
}
this.out_Qc_C *= this.out_kQc_S;

// 定格冷房入力(電力)
this.out_PPEc_S = spec.getSpecValue( this.SPEC_out_PPEc_S, 0. );

if( this.isSelectSPECList ){
   this.out_PPEc_S = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NCEleInputMain" ) ) * 1000.;//kW->W;
}
this.out_PPEc_S *= this.out_kPPEc_S;

// 中間冷房入力(電力)
this.out_PPEc_C = spec.getSpecValue( this.SPEC_out_PPEc_C, 0. );

if( this.isSelectSPECList ){
   this.out_PPEc_C = 0.;
}
this.out_PPEc_C *= this.out_kPPEc_S;

// 定格暖房能力
this.out_Qh_S = spec.getSpecValue( this.SPEC_out_Qh_S, 0. );

if( this.isSelectSPECList ){
   this.out_Qh_S = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NHCapacity" ) ) * 1000.;//kW->W;
}
this.out_Qh_S *= this.out_kQh_S;

// 中間暖房能力
this.out_Qh_C = spec.getSpecValue( this.SPEC_out_Qh_C, 0. );

if( this.isSelectSPECList ){
   this.out_Qh_C = 0.;
}
this.out_Qh_C *= this.out_kQh_S;

// 低温暖房能力
this.out_Qh_L = spec.getSpecValue( this.SPEC_out_Qh_L, this.out_Qh_S * 0.7 );

if( this.isSelectSPECList ){
   this.out_Qh_L = this.out_Qh_S * 0.7;
}
this.out_Qh_L *= this.out_kQh_S;

// 定格暖房入力(電力)
this.out_PPEh_S = spec.getSpecValue( this.SPEC_out_PPEh_S, 0. );

if( this.isSelectSPECList ){
   this.out_PPEh_S = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NHEleInputMain" ) ) * 1000.;//kW->W;
}
this.out_PPEh_S *= this.out_kPPEh_S;

// 中間暖房入力(電力)
this.out_PPEh_C = spec.getSpecValue( this.SPEC_out_PPEh_C, 0. );

if( this.isSelectSPECList ){
   this.out_PPEh_C = 0.;
}
this.out_PPEh_C *= this.out_kPPEh_S;

// 低温暖房入力(電力)
```

```
        this.out_PPEh_L = spec.getSpecValue( this.SPEC_out_PPEh_L, 0. );

        if( this.isSelectSPECList ){
           this.out_PPEh_L = 0.;
        }
        this.out_PPEh_L *= this.out_kPPEh_S;

        //SPEC_out_PPEfan    = "室外機ファン消費電力[W]";
        this.out_PPEfan = spec.getSpecValue( this.SPEC_out_PPEfan, 0. );

        //SPEC_isFanOnOff    = "室外機ファン連動";
        this.isFanOnOff = spec.getSpecValue( this.SPEC_isFanOnOff, false );

    /*
        // クランクケースヒータ(運転時)
        if(null!=map.get(this.SPEC_out_CLEa_R)){
           this.out_CLEa_R = Double.parseDouble((String)map.get(this.SPEC_out_CLEa_R));
        } else {
           System.out.println("(E)SPEC_クランクケースヒータ(運転時)がありません");
        }

        // クランクケースヒータ(停止時)
        if(null!=map.get(this.SPEC_out_CLEa_S)){
           this.out_CLEa_S = Double.parseDouble((String)map.get(this.SPEC_out_CLEa_S));
        } else {
           System.out.println("(E)SPEC_クランクケースヒータ(停止時)がありません");
        }

        // 待機電力(運転時)
        if(null!=map.get(this.SPEC_out_TKEa_R)){
           this.out_TKEa_R = Double.parseDouble((String)map.get(this.SPEC_out_TKEa_R));
        } else {
           System.out.println("(E)SPEC_待機電力(運転時)がありません");
        }

        // 待機電力(待機時)
        if(null!=map.get(this.SPEC_out_TKEa_T)){
           this.out_TKEa_T = Double.parseDouble((String)map.get(this.SPEC_out_TKEa_T));
        } else {
           System.out.println("(E)SPEC_待機電力(待機時)がありません");
        }

        // 待機電力(停止時)
        if(null!=map.get(this.SPEC_out_TKEa_S)){
           this.out_TKEa_S = Double.parseDouble((String)map.get(this.SPEC_out_TKEa_S));
        } else {
           System.out.println("(E)SPEC_待機電力(停止時)がありません");
        }
    */
        //SPEC_fRate_airEA   = "風量[g/s]";
        this.fRate_airEA = spec.getSpecValue( this.SPEC_fRate_airEA, this.out_Qc_S * 0.11 );

        if( this.isSelectSPECList ){
           this.fRate_airEA = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NCFRair" ) ) * 1200 / 3600.;//CMH->g/s
           if( this.fRate_airEA == 0 ){
              this.fRate_airEA = this.out_Qc_S * 0.11;//20110505nino
           }
        }



        // 機器起動停止負荷率
        this.in_run_stop = spec.getSpecValue( this.SPEC_in_run_stop, 0.3 );

        // 相数
        this.phase = spec.getSpecValue( this.SPEC_Phase, 3 );

        // 電圧
        this.voltage = spec.getSpecValue( this.SPEC_Voltage, 200. );

        // 周波数
        this.frequency = spec.getSpecValue( this.SPEC_Frequency, 50. );
```

```
if( this.isSelectSPECList ){
   if( bmSpecFR.getSpec( this.m_kt, "FrequencyEleIn" ).equals( "50 60" ) ){
      this.frequency = 50;
   }else{
      this.frequency = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "FrequencyEleIn" ) );
   }
}

// 力率
this.powerFactor = spec.getSpecValue( this.SPEC_PowerFactor, 0.8 );

//isGVisibleを取得
this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

//isRecordを取得
this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );


//低負荷領域の計算方法
this.calcTypeLowerRangeLoad = spec.getSpecValue( this.SPEC_CalcTypeLowerRangeLoad, "0_発停運転" );
//calcTypeLowerRangeLoad
//  0_発停運転
//  1_下限入力値固定
//  2_下限COP値固定
//  3_下限入力値と中間切片
if( this.calcTypeLowerRangeLoad.equals( "0_発停運転" )){
   this.num_calcTypeLowerRangeLoad = 0;
}else if(this.calcTypeLowerRangeLoad.equals( "1_下限入力値固定" )){
   this.num_calcTypeLowerRangeLoad = 1;
}else if(this.calcTypeLowerRangeLoad.equals( "2_下限COP値固定" )){
   this.num_calcTypeLowerRangeLoad = 2;
}else if(this.calcTypeLowerRangeLoad.equals( "3_下限入力値と中間切片" )){
   this.num_calcTypeLowerRangeLoad = 3;
}else{
   this.num_calcTypeLowerRangeLoad = 0;
}


if(out_Qc_S<0.001)      fmc= 0.0;
   else           fmc= out_Qc_C/out_Qc_S;
if(out_PPEc_S<0.001) pmc= 0.0;
   else           pmc= out_PPEc_C/out_PPEc_S;

if(out_Qh_S<0.001)      fmh= 0.0;
   else           fmh= out_Qh_C/out_Qh_S;
if(out_PPEh_S<0.001) pmh= 0.0;
   else           pmh= out_PPEh_C/out_PPEh_S;

if(pmc>0.9999) {pmc=0.0;fmc=0.0;}
if(pmh>0.9999) {pmh=0.0;fmh=0.0;}

//isAdjust2012 = "台数を調整する";//
this.isAdjust2012 = spec.getSpecValue( this.SPEC_isAdjust2012, false );
//自動調整する場合は０％まで運転とする。20200108
if( this.isAdjust2012 ) {
   this.in_run_stop = 0;
}

// 調整の計算ステップ数
this.numAdjustSteps = spec.getSpecValue( this.SPEC_NumAdjustSteps, 12 );

//isUseAdjustList = "is調整リストを使用する";//
this.isUseAdjustList = spec.getSpecValue( this.SPEC_isUseAdjustList, false );

//仮設調整
this.fRate_airEAc = this.fRate_airEA;
this.fRate_airEAh = this.fRate_airEA;

this.rcList = new LinkedList<Double>();
this.rhList = new LinkedList<Double>();
for( int i=0; i<this.numAdjustSteps; i++ ){
```

```
        this.rcList.add( 0. );
        this.rhList.add( 0. );
    }

    this.out_Qc_S_1   = this.out_Qc_S;//= "定格冷房能力";//kW
    this.out_Qc_C_1   = this.out_Qc_C;//= "中間冷房能力";//kW
    this.out_PPEc_S_1 = this.out_PPEc_S;//= "定格冷房入力(電力)";//kW
    this.out_PPEc_C_1 = this.out_PPEc_C;//= "中間冷房入力(電力)";//kW
    this.out_Qh_S_1   = this.out_Qh_S;//= "定格暖房能力";//kW
    this.out_Qh_C_1   = this.out_Qh_C;//= "中間暖房能力";//kW
    this.out_Qh_L_1   = this.out_Qh_L;//= "低温暖房能力";//kW
    this.out_PPEh_S_1 = this.out_PPEh_S;//= "定格暖房入力(電力)";//kW
    this.out_PPEh_C_1 = this.out_PPEh_C;//= "中間暖房入力(電力)";//kW
    this.out_PPEh_L_1 = this.out_PPEh_L;//= "低温暖房入力(電力)";//kW

    //自動調整
    if( this.isAdjust2012 && this.isUseAdjustList ){
        this.out_Qc_S    = this.adjustQcListkW[0] * 1000. * this.out_daiAdjust;//[kW]->[W]
        this.out_Qh_S    = this.adjustQhListkW[0] * 1000. * this.out_daiAdjust;//[kW]->[W]
        //this.out_QhL_S   = 0 * this.out_daiAdjust;
        this.out_PPEc_S = this.adjustPPEcListkW[0] * 1000. * this.out_daiAdjust;//[kW]->[W]
        this.out_PPEh_S = this.adjustPPEhListkW[0] * 1000. * this.out_daiAdjust;//[kW]->[W]

        this.fRate_airEA    = this.adjustfRateListm3min[0] * 60 / 3 * this.out_daiAdjust;//[m3/min]->[g/s]
    }
}


@Override
public void initialize(IBestStateMessage stateNodes,
        IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
        IBestDomainSchedule schedule) {
    //System.out.println(this.moduleName + " BuillMultiOutイニシャライズ");
    //状態ノードを受け取る
    super.sm = stateNodes;
    //制御ノードを受け取る
    super.cm = commandNodes;
    //記録ノードを受け取る
    super.rm = recordNodes;

    //eleIn
    this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

    //airInOA
    this.airInOA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInOA );

    //private final String S_NODE_airOutEA   = "LO_airOutEA";//排気
    this.airOutEA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutEA );

    //private final String S_NODE_valInLine  = "LO_valInLine";  //単線
    if( super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine )) != null ){

        //System.out.println( " BM_out S_NODE_valInLine登録済み" );

        this.rmlinemap
        =  (Map<String, BM_EHPdata>) super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine));

        if( this.rmlinemap.size() == 0 ){
            //
            //System.out.println( "rmlinemap のサイズが０ ");
            this.RMdata = new BM_EHPdata();
            this.RMdata.set_path( this.path );
            this.RMdata.set_kiki( this.kiki_file );
            this.RMdata.setUserTokusei( this.isUserTokusei );
            this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
            this.RMdata.setHighSH(this.isHighSH );
            this.RMdata.setBESTESTtable(this.isBESTESTtable );
            this.RMdata.setN_outDB( this.kiki_file );
            this.RMdata.setairInHS( this.airInOA );//20120823nino
            this.rmlinemap.put( "Parent", this.RMdata );

            super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );
```

```
        }else{
            this.num = 0;
            Set<String> LineEntry = this.rmlinemap.keySet();
            for(Iterator<String> i = LineEntry.iterator();i.hasNext();){
                String key = (String) i.next();
                //          System.out.println(" name="+name+" key="+key+" num="+num);
                this.RMdata = this.rmlinemap.get(key);
                this.rmnm[this.num]=key;
                //          System.out.println("key="+key+" 0="+rmdata.[0]+" 1="+rmdata[1]+" 2="+rmdata[2]+"
3="+rmdata[3]+" 4="+rmdata[4]+" 5="+rmdata[5]+" 6="+rmdata[6]);
                this.num++;
            }
            //20150409
            double sum_in_Qc_S = 0;
            double sum_in_Qh_S = 0;
            double sum_in_dai = 0;
            //出力設定
            for(int i=0; i<this.num; i++){
                //System.out.println( "kiki_file Name = " +kiki_file );
                this.RMdata = this.rmlinemap.get( this.rmnm[i] );
                this.RMdata.set_kiki(kiki_file);
                this.RMdata.set_path(this.path);
                this.RMdata.setUserTokusei( this.isUserTokusei );
                this.RMdata.setCOOLandHEAT( this.isCOOLandHEATout );
                this.RMdata.setHighSH(this.isHighSH );
                this.RMdata.setBESTESTtable(this.isBESTESTtable );
                this.RMdata.setN_outDB( this.kiki_file );
                this.RMdata.setairInHS( this.airInOA );//20120823nino
                rmlinemap.put(rmnm[i],this.RMdata);
                //20150409
                sum_in_Qc_S += this.RMdata.get_in_Qc_S();
                sum_in_Qh_S += this.RMdata.get_in_Qh_S();
                sum_in_dai += this.RMdata.get_in_dai();
            }
            //
            //System.out.println( "rmlinemap のサイズが "+this.num );
            this.RMdata = new BM_EHPdata();
            this.RMdata.set_path( this.path );
            this.RMdata.set_kiki( this.kiki_file );
            this.RMdata.setUserTokusei( this.isUserTokusei );
            this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
            this.RMdata.setHighSH(this.isHighSH );
            this.RMdata.setBESTESTtable(this.isBESTESTtable );
            this.RMdata.setN_outDB( this.kiki_file );
            this.RMdata.setairInHS( this.airInOA );//20120823nino
            //20150409
            this.RMdata.set_in_Qc_S( sum_in_Qc_S );
            this.RMdata.set_in_Qh_S( sum_in_Qh_S );
            this.RMdata.set_in_dai( sum_in_dai );

            this.rmlinemap.put( "Parent", this.RMdata );

            //20150409
        /*  if( this.out_Qc_S > 0 ){
                if( sum_in_Qc_S / this.out_Qc_S > 1.5 ){
                    BestLogger.info( "(W)"+AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
                          + "接続定格冷却容量"+AirFormat.df_1( sum_in_Qc_S )+"[W]/室外機定格容量" + this.out_Qc_S + "[W]("
                                +AirFormat.df_3( sum_in_Qc_S / this.out_Qc_S ) + "倍)>1.5倍" );
                }else{
                    BestLogger.info( "(C)"+AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
                          + "接続定格冷却容量"+AirFormat.df_1( sum_in_Qc_S )+"[W]/室外機定格容量" + this.out_Qc_S + "[W]("
                                +AirFormat.df_3( sum_in_Qc_S / this.out_Qc_S ) + "倍)" );
                }
            }
            if( this.out_Qh_S > 0 ){
                if( sum_in_Qh_S / this.out_Qh_S > 1.5 ){
                    BestLogger.info( "(W)"+AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
                          + "接続定格加熱容量"+AirFormat.df_1( sum_in_Qh_S )+"[W]/室外機定格容量" + this.out_Qh_S + "[W]("
                                +AirFormat.df_3( sum_in_Qh_S / this.out_Qh_S ) + "倍)>1.5倍" );
                }else{
                    BestLogger.info( "(C)"+AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
```

```
                    + "接続定格加熱容量"+AirFormat.df_1( sum_in_Qh_S )+"[W]/室外機定格容量" + this.out_Qh_S + "[W]("
                        +AirFormat.df_3( sum_in_Qh_S / this.out_Qh_S ) + "倍)" );
            }

         }
         BestLogger.info( "(C)"+AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
         + "接続台数="+AirFormat.df_0( sum_in_dai )+"[台]" );
      */
         //接続容量、台数チェック
         this.RMdata.checkConnect( this.out_Qc_S, this.out_Qh_S, sum_in_Qc_S, sum_in_Qh_S, sum_in_dai,
               this.moduleName, "initialize", this.name, this.isRecord, this.message );

         super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );
         this.dbflg = 1;
      }
   }else{

      //System.out.println( " BM_out S_NODE_valInLine未登録" );

      //System.out.println( this.moduleName + ">>Warning<< rmlinemap is null !!" );
      message.append( "(W)単線の接続なし→作成");
      this.rmlinemap = new HashMap<String,BM_EHPdata>();
      //
      this.RMdata = new BM_EHPdata();
      this.RMdata.set_path( this.path );
      this.RMdata.set_kiki( this.kiki_file );
      this.RMdata.setUserTokusei( this.isUserTokusei );
      this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
      this.RMdata.setHighSH(this.isHighSH );
      this.RMdata.setBESTESTtable(this.isBESTESTtable );
      this.RMdata.setN_outDB( this.kiki_file );
      this.RMdata.setairInHS( this.airInOA );//20120823nino
      this.rmlinemap.put( "Parent", this.RMdata );

      super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );
   }

   //System.out.println( " Parent path="+ this.rmlinemap.get( "Parent" ).get_path() );

   //グラフ表示の準備
   if( this.isGVisible ){
      gBMout = new GraphJFrameBuilMultiOut_S20101212( this.name, 300,  out_Qc_S * 1.5, 0 );
   }



   filenames[0]=kiki_file;
   equipmentName=kiki_file;
   pacDB=new PACDBManager( path, filenames, this.isUserTokusei );
   pacDB.setEquipmentName(equipmentName);

}



/**
 * ビルマルチ計算及び結果出力/各室内機よりも先に動きます。
 */
@Override
public void outputs() {

   if( !this.isCalc ){
      return;
   }

   if (null == super.cm || null == super.sm )
      return;

   if( this.dbflg == 0 ){

      //System.out.println( "    BM_out  outputsで  室側データの設定" );
```

```
        //System.out.println( " Parent path="+ this.rmlinemap.get( "Parent" ).get_path() );

        if( super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine )) != null ){
           this.rmlinemap
           =  (Map<String, BM_EHPdata>) super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine));
           //
           this.num = 0;
           Set<String> LineEntry = this.rmlinemap.keySet();
           for(Iterator<String> i = LineEntry.iterator();i.hasNext();){
              String key = (String) i.next();
              //          System.out.println(" name="+name+" key="+key);
              this.RMdata = this.rmlinemap.get(key);
              this.rmnm[this.num]=key;
              //          System.out.println("key="+key+" 0="+rmdata[0]+" 1="+rmdata[1]+" 2="+rmdata[2]+"
3="+rmdata[3]+" 4="+rmdata[4]+" 5="+rmdata[5]+" 6="+rmdata[6]);
              this.num++;
           }
           //20150409
           double sum_in_Qc_S = 0;
           double sum_in_Qh_S = 0;
           double sum_in_dai = 0;
           //出力設定
           for(int i=0; i<this.num; i++){
              //System.out.println( "kiki_file Name = " +kiki_file );
              this.RMdata = this.rmlinemap.get( this.rmnm[i] );
              this.RMdata.set_kiki(kiki_file);
              this.RMdata.set_path(this.path);
              this.RMdata.setUserTokusei( this.isUserTokusei );
              this.RMdata.setCOOLandHEAT( this.isCOOLandHEATout );
              this.RMdata.setHighSH(this.isHighSH );
              this.RMdata.setBESTESTtable(this.isBESTESTtable );
              this.RMdata.setN_outDB( this.kiki_file );
              this.RMdata.setairInHS( this.airInOA );//20120823nino
              rmlinemap.put(rmnm[i],this.RMdata);
              //20150409
              sum_in_Qc_S += this.RMdata.get_in_Qc_S();
              sum_in_Qh_S += this.RMdata.get_in_Qh_S();
              sum_in_dai += this.RMdata.get_in_dai();
           }
           //
           //20150409
           this.RMdata = this.rmlinemap.get( "Parent" );
           //20150409
           this.RMdata.set_in_Qc_S( sum_in_Qc_S );
           this.RMdata.set_in_Qh_S( sum_in_Qh_S );
           this.RMdata.set_in_dai( sum_in_dai );

           this.rmlinemap.put( "Parent", this.RMdata );

           //20150409
     /*    BestLogger.info( "(C)"+AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
           + "接続定格冷却容量"+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量" + this.out_Qc_S + "[W]("
                 +AirFormat.df_3( sum_in_Qc_S / this.out_Qc_S ) + "倍)" );
           BestLogger.info( "(C)"+AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
           + "接続定格加熱容量"+AirFormat.df_1( sum_in_Qh_S )+"[W]室外機定格容量" + this.out_Qh_S + "[W]("
                 +AirFormat.df_3( sum_in_Qh_S / this.out_Qh_S ) + "倍)" );
           BestLogger.info( "(C)"+AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
           + "接続台数="+AirFormat.df_0( sum_in_dai )+"[台]" );
        */
           //接続容量、台数チェック
           this.RMdata.checkConnect( this.out_Qc_S, this.out_Qh_S, sum_in_Qc_S, sum_in_Qh_S, sum_in_dai,
                 this.moduleName, "outputs", this.name, this.isRecord, this.message );

        }

        this.dbflg = 1;

     }

     //message.setLength(0);
     //ノード接続毎回読込み
     this.airInOA = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airInOA));
```

```
this.swcIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcIn) );
this.modIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modIn));

//外気乾球温度設定
this.DBoa = this.airInOA.getTempDB();

//外気絶対湿度を設定
this.XGoa = this.airInOA.getHumi();

//外気湿球温度を設定
this.WBoa = Psychrometrics.FNWbtx(this.DBoa, this.XGoa);
if( this.WBoa < -9000.0 ){
   this.WBoa = this.WB_cal(this.DBoa, this.XGoa);
}

//初期化
this.Tc_Loadout = 0;
this.Th_Loadout = 0;
this.PPEfan = this.out_PPEfan;

//mState=this.swc*this.mod;
//
if( Airswc.isOFF( this.swcIn )){
   //停止
   this.mState = 0;
}else if( Airswc.isON( this.swcIn )){
   //運転
   if( Airmod.isCOOL( this.modIn ) && Airmod.isHEAT( this.modIn ) ){
      //冷暖同時
      if( this.isCOOLandHEATout ){
         this.mState = 11;
      }else{
         //冷暖同時機種で無い場合は冷房運転とする
         this.mState = 1;
         this.message.append( "(W)冷暖同時機種ではないので冷房運転で計算" );
      }
   }else if( Airmod.isCOOL( this.modIn )){
      //冷房運転
      this.mState = 1;
   }else if( Airmod.isHEAT( this.modIn )){
      //暖房運転
      this.mState = 2;
   }else if( Airmod.isVENTILATE( this.modIn )){
      //換気モード
      this.mState = 3;
   }else{
      //停止
      this.mState = 0;
   }
}


/**
 * 入力
 */

this.rmlinemap
= (Map<String,BM_EHPdata>) super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine ));

this.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
this.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
this.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
this.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
double DBra_mean = 0;
double WBra_mean = 0;
this.Lratemin = 100.0;

double dmy1 = 100.0;

//室内機データの読込
for( int i=0; i < this.num; i++ ){
   this.RMdata = this.rmlinemap.get( this.rmnm[i] );
```

```
            dmy1 = this.RMdata.get_L_rate();
            if( this.Lratemin > dmy1 ){
               this.Lratemin=dmy1;
            }
            //dmy1 = this.RMdata.get_R_wb();
            //if( this.WBra_min > dmy1 && this.RMdata.get_R_Tc_load()>0 ){
            // this.WBra_min=dmy1;
            //}
            //dmy1 = this.RMdata.get_R_db();
            //if( DBra_main < dmy1 && this.RMdata.get_R_Th_load()>0 ){
            // DBra_main=dmy1;
            //}
            DBra_mean += this.RMdata.get_R_db();
            WBra_mean += this.RMdata.get_R_wb();

            this.allSc_heat += this.RMdata.get_R_Sc_load();//室内機処理熱量（冷却顕熱）[W]
            this.allTc_heat += this.RMdata.get_R_Tc_load();//室内機処理熱量（冷却全熱）[W]
            //
            this.allSh_heat += this.RMdata.get_R_Sh_load();//室内機処理熱量（加熱顕熱）[W]
            this.allTh_heat += this.RMdata.get_R_Th_load();//室内機処理熱量（加熱全熱）[W]

            // System.out.println( "this.RMdata.get_R_Tc_load()" + this.RMdata.get_R_Tc_load() +
"this.RMdata.get_R_Th_load()" + this.RMdata.get_R_Th_load());
         }

         this.WBra = WBra_mean / this.num;
         this.DBra = DBra_mean / this.num;

         //*************************----------------
         if( this.isAdjust2012 ){
            double rc;
            double c_OutTemp;//冷房能力：温度、冷媒管特性適用
            double c_OutTempList;//冷房能力補正：温度、冷媒管特性適用
            double rh;
            double h_OutTemp;//暖房能力：温度、冷媒管特性適用
            double h_OutTempList;//暖房能力補正：温度、冷媒管特性適用

            //処理可能熱量　仮計算
            if( this.allTc_heat==0 && this.allTh_heat==0 ){
               //冷却・加熱要求なし
               c_OutTemp = this.out_Qc_S_1;
               h_OutTemp = this.out_Qh_S_1;
               c_OutTempList = 1.;
               h_OutTempList = 1.;
            }else if( this.allTh_heat == 0 ){
               //冷却要求のみ coolingOnly_cal()
               c_OutTemp = this.pacDB.getKcta( this.DBoa) * this.pacDB.getKcti( this.WBra) * this.Lratemin *
this.out_Qc_S_1;//仮計算　室外機処理可能熱量（冷却全熱）[W]
               h_OutTemp = this.out_Qh_S_1;
               c_OutTempList = this.pacDB.getKcta( this.DBoa) * this.pacDB.getKcti( this.WBra) * this.Lratemin;//仮計算　補
正：室外機処理可能熱量（冷却全熱）[W]
               h_OutTempList = 1.;
            }else if( this.allTc_heat == 0 ){
               //加熱要求のみ heatingOnly_cal()
               c_OutTemp = this.out_Qc_S_1;
               c_OutTempList = 1.;
               if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L_1 > 0.0001) { //低温入力の場合
                  h_OutTemp  = this.pacDB.getKhta( this.WBoa) * this.pacDB.getKhti( this.DBra) * this.Lratemin *
this.out_Qh_L_1;//仮計算　室外機処理可能熱量（加熱全熱）[W]
                  h_OutTempList  = this.pacDB.getKhta( this.WBoa) * this.pacDB.getKhti( this.DBra) * this.Lratemin;//仮計算
補正：室外機処理可能熱量（加熱全熱）[W]
               }else{
                  h_OutTemp  = this.pacDB.getKhta( this.WBoa) * this.pacDB.getKhti( this.DBra) * this.Lratemin *
this.out_Qh_S_1;//仮計算　室外機処理可能熱量（加熱全熱）[W]
                  h_OutTempList  = this.pacDB.getKhta( this.WBoa) * this.pacDB.getKhti( this.DBra) * this.Lratemin;//仮計算
補正：室外機処理可能熱量（加熱全熱）[W]
               }
            }else if( this.allTc_heat >= this.allTh_heat ){
               //冷却主体運転 cooling_and_heating_cal()
               c_OutTemp = this.pacDB.getKcmta( this.DBoa) * this.pacDB.getKcti( this.WBra) * this.Lratemin *
this.out_Qc_S_1;//仮計算　室外機処理可能熱量（冷却全熱）[W]
               h_OutTemp = this.pacDB.getKhti( this.DBra ) * this.Lratemin * this.out_Qh_S_1;//室外機処理可能熱量（加熱全
```

```
熱）[W]
            c_OutTempList = this.pacDB.getKcmta( this.DBoa) * this.pacDB.getKcti( this.WBra) * this.Lratemin;//仮計算　補
正：室外機処理可能熱量（冷却全熱）[W]
            h_OutTempList = this.pacDB.getKhti( this.DBra ) * this.Lratemin;//補正：室外機処理可能熱量（加熱全熱）[W]
         }else{
            //加熱主体運転 heating_and_cooling_cal()
            c_OutTemp = this.pacDB.getKcti( this.WBra) * this.Lratemin * out_Qc_S_1;//室外機処理可能熱量（冷却全熱）[W]
            c_OutTempList = this.pacDB.getKcti( this.WBra) * this.Lratemin;//補正：室外機処理可能熱量（冷却全熱）[W]

            if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L_1 > 0.0001) { //低温入力の場合
               h_OutTemp  = this.pacDB.getKhta( this.WBoa ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin *
out_Qh_L_1;//室外機処理可能熱量（加熱全熱）[W]
               h_OutTempList  = this.pacDB.getKhta( this.WBoa ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin *
out_Qh_L_1 / out_Qh_S_1;//補正：室外機処理可能熱量（加熱全熱）[W]
               double Rch = this.allTc_heat / h_OutTemp / this.out_rhAdjust;
               double deltaWB = this.pacDB.getKdwb( Rch );
               //補正暖房容量
               h_OutTemp  = this.pacDB.getKhmta( this.WBoa + deltaWB ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin
* out_Qh_L_1;//室外機処理可能熱量（加熱全熱）[W]
               h_OutTempList  = this.pacDB.getKhmta( this.WBoa + deltaWB ) * this.pacDB.getKhti( this.DBra ) *
this.Lratemin * out_Qh_L_1 / out_Qh_S_1;//補正：室外機処理可能熱量（加熱全熱）[W]
            }else{
               h_OutTemp  = this.pacDB.getKhta( this.WBoa ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin *
out_Qh_S_1;//室外機処理可能熱量（加熱全熱）[W]
               h_OutTempList  = this.pacDB.getKhta( this.WBoa ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin;//補
正：室外機処理可能熱量（加熱全熱）[W]
               double Rch = this.allTc_heat / h_OutTemp / this.out_rhAdjust;
               double deltaWB = this.pacDB.getKdwb( Rch );
               //補正暖房容量
               h_OutTemp  = this.pacDB.getKhmta( this.WBoa + deltaWB ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin
* out_Qh_S_1;//室外機処理可能熱量（加熱全熱）[W]
               h_OutTempList  = this.pacDB.getKhmta( this.WBoa + deltaWB ) * this.pacDB.getKhti( this.DBra ) *
this.Lratemin;//補正：室外機処理可能熱量（加熱全熱）[W]
            }
         }

         this.out_Qc_Adjusthosei = c_OutTempList;
         this.out_Qh_Adjusthosei = h_OutTempList;

         //冷却能力調整
         if( c_OutTemp == 0 ){
            //rc = this.allTc_heat / this.out_Qc_S;
            if( this.isUseAdjustList ){
               rc = this.allTc_heat;
            }else{
               rc = this.allTc_heat / this.out_Qc_S_1;
            }
         }else{
            if( this.isUseAdjustList ){
               rc = this.allTc_heat / c_OutTempList;
            }else{
               rc = this.allTc_heat / c_OutTemp;
            }
         }

         if( rc > 1.1 ){
         // rc = 1.1;
         }

         //加熱能力調整
         if( h_OutTemp == 0 ){
            //rh = this.allTh_heat / this.out_Qh_S;
            if( this.isUseAdjustList ){
               rh = this.allTh_heat;
            }else{
               rh = this.allTh_heat / this.out_Qh_S_1;
            }
         }else{
            if( this.isUseAdjustList ){
               rh = this.allTh_heat / h_OutTempList;
            }else{
               rh = this.allTh_heat / h_OutTemp;
```

```
    }
}

if( rh > 1.1 ){
// rh = 1.1;
}

//*************************----------------
if( true ){
   //冷却側の調整
   //移動平均の最大値で調整していく
   this.rcList.removeLast();
   this.rcList.addFirst( rc );
   //
   double sum_rc = 0;
   for( int i=0; i<this.numAdjustSteps; i++ ){
      sum_rc += this.rcList.get( i );
   }
   this.ave_rcAdjust = sum_rc / this.numAdjustSteps;
   //
   if( sum_rc > this.max_rcAdjust * this.numAdjustSteps  && !this.isUseAdjustList ){
      this.max_rcAdjust = sum_rc / this.numAdjustSteps;
      //従来の調整
      this.out_rcAdjust = this.max_rcAdjust;
      //this.in_run_stop_Num = this.in_run_stop / this.out_rcAdjust;

      this.out_Qc_S   = this.out_Qc_S_1    * this.out_rcAdjust;
      this.out_Qc_C   = this.out_Qc_C_1    * this.out_rcAdjust;
      this.out_PPEc_S = this.out_PPEc_S_1  * this.out_rcAdjust;
      this.out_PPEc_C = this.out_PPEc_C_1  * this.out_rcAdjust;

      this.fRate_airEAc = this.fRate_airEA * this.out_rcAdjust;

   }else if( this.ave_rcAdjust > this.max_rcAdjust  && this.isUseAdjustList ){
      //リストを使用するとき
      int start = this.numList;
      if( this.adjustQcListkW[ this.adjustQcListkW.length-1 ] * 1000. * this.out_daiAdjust
            < this.ave_rcAdjust ){
         this.out_daiAdjust += 1.;
         start = 0;
         System.out.println(" リスト比較  リセット  台数+1-->"+this.out_daiAdjust);
      }

      int n=0;
      for( n=start; n<this.adjustQcListkW.length; n++ ){

         if( this.adjustQcListkW[n] * 1000. * this.out_daiAdjust
               > this.ave_rcAdjust ){

            this.out_Qc_S    = this.adjustQcListkW[n] * 1000. * this.out_daiAdjust;//[kW]->[W]
            this.out_Qh_S    = this.adjustQhListkW[n] * 1000. * this.out_daiAdjust;//[kW]->[W]
            //this.out_QhL_S   = 0 * this.out_daiAdjust;
            this.out_PPEc_S = this.adjustPPEcListkW[n] * 1000. * this.out_daiAdjust;//[kW]->[W]
            this.out_PPEh_S = this.adjustPPEhListkW[n] * 1000. * this.out_daiAdjust;//[kW]->[W]

            this.fRate_airEA    = this.adjustfRateListm3min[n] * 60 / 3 * this.out_daiAdjust;//[m3/min]->[g/s]

            this.out_Qc_S_1   = this.out_Qc_S;//= "定格冷房能力";//kW
            this.out_Qc_C_1   = this.out_Qc_C;//= "中間冷房能力";//kW
            this.out_PPEc_S_1 = this.out_PPEc_S;//= "定格冷房入力(電力)";//kW
            this.out_PPEc_C_1 = this.out_PPEc_C;//= "中間冷房入力(電力)";//kW
            this.out_Qh_S_1   = this.out_Qh_S;//= "定格暖房能力";//kW
            this.out_Qh_C_1   = this.out_Qh_C;//= "中間暖房能力";//kW
            this.out_Qh_L_1   = this.out_Qh_L;//= "低温暖房能力";//kW
            this.out_PPEh_S_1 = this.out_PPEh_S;//= "定格暖房入力(電力)";//kW
            this.out_PPEh_C_1 = this.out_PPEh_C;//= "中間暖房入力(電力)";//kW
            this.out_PPEh_L_1 = this.out_PPEh_L;//= "低温暖房入力(電力)";//kW

            this.numList = n;
            break;
         }
      }
```

```
            this.max_rcAdjust = this.out_Qc_S;

            this.out_rcAdjust = this.max_rcAdjust;
        }
    }


    if( true ){
        //加熱側の調整
        //移動平均の最大値で調整していく
        this.rhList.removeLast();
        this.rhList.addFirst( rh );
        //
        double sum_rh = 0;
        for( int i=0; i<this.numAdjustSteps; i++ ){
            sum_rh += this.rhList.get( i );
        }
        this.ave_rhAdjust = sum_rh / this.numAdjustSteps;

        //System.out.println(" sum_rh="+sum_rh+ "  max_rhAdjust="+max_rhAdjust );
        //
        if( sum_rh > this.max_rhAdjust * this.numAdjustSteps && !this.isUseAdjustList ){
            //従来方法
            this.max_rhAdjust = sum_rh / this.numAdjustSteps;

            this.out_rhAdjust = this.max_rhAdjust;
            //this.in_run_stop_Num = this.in_run_stop / this.out_rcAdjust;

            this.out_Qh_S   = this.out_Qh_S_1   * this.out_rhAdjust;
            this.out_Qh_C   = this.out_Qh_C_1   * this.out_rhAdjust;
            this.out_Qh_L   = this.out_Qh_L_1   * this.out_rhAdjust;
            this.out_PPEh_S = this.out_PPEh_S_1 * this.out_rhAdjust;
            this.out_PPEh_C = this.out_PPEh_C_1 * this.out_rhAdjust;
            this.out_PPEh_L = this.out_PPEh_L_1 * this.out_rhAdjust;

            this.fRate_airEAh = this.fRate_airEA* this.out_rhAdjust;

        }else if( this.ave_rhAdjust > this.max_rhAdjust && this.isUseAdjustList ){
            //リストを使用するとき
            int start = this.numList;
            if( this.adjustQhListkW[ this.adjustQhListkW.length-1 ] * 1000. * this.out_daiAdjust
                  < this.ave_rhAdjust ){
                this.out_daiAdjust += 1.;
                start = 0;
                //System.out.println(" リスト比較  リセット  台数+1-->"+this.out_daiAdjust);
            }

            int n=0;
            for( n=start; n<this.adjustQhListkW.length; n++ ){

                if( this.adjustQhListkW[n] * 1000. * this.out_daiAdjust
                      >= this.ave_rhAdjust ){

                    //System.out.println("  out_Qh_S="+out_Qh_S+" -->> "+(this.adjustQhListkW[n] * 1000. *
this.out_daiAdjust));
                    this.out_Qc_S    = this.adjustQcListkW[n] * 1000. * this.out_daiAdjust;//[kW]->[W]
                    this.out_Qh_S    = this.adjustQhListkW[n] * 1000. * this.out_daiAdjust;//[kW]->[W]
                    this.out_Qh_C   = 0 * this.out_daiAdjust;
                    this.out_Qh_L   = 0 * this.out_daiAdjust;
                    this.out_PPEc_S = this.adjustPPEcListkW[n] * 1000. * this.out_daiAdjust;//[kW]->[W]
                    this.out_PPEh_S = this.adjustPPEhListkW[n] * 1000. * this.out_daiAdjust;//[kW]->[W]
                    this.out_PPEh_C   = 0 * this.out_daiAdjust;
                    this.out_PPEh_L   = 0 * this.out_daiAdjust;

                    this.fRate_airEA    = this.adjustfRateListm3min[n] * 60 / 3 * this.out_daiAdjust;//[m3/min]->[g/s]

                    this.out_Qc_S_1   = this.out_Qc_S;//= "定格冷房能力";//kW
                    this.out_Qc_C_1   = this.out_Qc_C;//= "中間冷房能力";//kW
                    this.out_PPEc_S_1 = this.out_PPEc_S;//= "定格冷房入力(電力)";//kW
                    this.out_PPEc_C_1 = this.out_PPEc_C;//= "中間冷房入力(電力)";//kW
                    this.out_Qh_S_1   = this.out_Qh_S;//= "定格暖房能力";//kW
```

```
                this.out_Qh_C_1   = this.out_Qh_C;//= "中間暖房能力";//kW
                this.out_Qh_L_1   = this.out_Qh_L;//= "低温暖房能力";//kW
                this.out_PPEh_S_1 = this.out_PPEh_S;//= "定格暖房入力(電力)";//kW
                this.out_PPEh_C_1 = this.out_PPEh_C;//= "中間暖房入力(電力)";//kW
                this.out_PPEh_L_1 = this.out_PPEh_L;//= "低温暖房入力(電力)";//kW

                this.numList = n;
                break;
              }
            }

            //this.max_rhAdjust = sum_rh / this.numAdjustSteps;
            this.max_rhAdjust = this.out_Qh_S;

            this.out_rhAdjust = this.max_rhAdjust;

          }
        }
        //*************************----------------


        //調整補正
/*      if( rc > 1.0 ){
          this.out_Qc_S *= rc;
          this.out_Qc_C *= rc;
          this.out_PPEc_S *= rc;
          this.out_PPEc_C *= rc;
          //
          this.fRate_airEAc *= rc;
        }
*/
        //調整補正
/*      if( rh > 1.0 ){
          this.out_Qh_S *= rh;
          this.out_Qh_C *= rh;
          this.out_Qh_L *= rh;
          this.out_PPEh_S *= rh;
          this.out_PPEh_C *= rh;
          this.out_PPEh_L *= rh;
          //
          this.fRate_airEAh *= rh;
        }
*/
        this.out_Qc_Adjust = this.out_Qc_S;
        this.out_Qh_Adjust = this.out_Qh_S;

        //System.out.println( "BM out  out_Qc_Adjust=" +out_Qc_Adjust+"  out_Qh_Adjust="+out_Qh_Adjust);
      }


      //*************************----------------


      switch( this.mState ){
      case 3://20120313nino
        //換気
        //室外機は停止とする
      case 0:
        //停止
        this.message.append("(C)停止");
        this.calc_Stop();
        break;
      case 1:
        //冷房
        if( this.out_Qc_S_1 > 0 ){
          this.message.append("(C)冷房運転");
          this.coolingOnly_cal();
        }else{
          //定格冷房能力が<=0の時　冷房できない
          this.message.append("(C)冷能力=0停止");//20130625
          this.calc_Stop();
        }
```

```
    break;
case 2:
    //暖房
    if( this.out_Qh_S_1 > 0 ){
        this.message.append("(C)暖房運転");
        this.heatingOnly_cal();
    }else{
        //定格暖房能力が<=0の時　暖房できない
        this.message.append("(C)暖能力=0停止");//20130625
        this.calc_Stop();
    }

    break;
case 11:
    //冷暖同時
    this.message.append("(C)冷暖房運転");
    if( this.allTh_heat == 0 && this.out_Qc_S_1 > 0 ){
        //冷房運転
        this.message.append("(C)冷専");
        this.coolingOnly_cal();

    }else if( this.allTc_heat == 0 && this.out_Qh_S_1 > 0 ){
        //暖房運転
        this.message.append("(C)暖専");
        this.heatingOnly_cal();

    }else if( this.allTc_heat > this.allTh_heat && this.out_Qc_S_1 > 0){
        //冷房主体運転
        this.message.append("(C)冷主");
        //20181017
        if( this.is2018SinseiCHFree ) {
            this.message.append("(C)申請");
            this.coolingOnly_cal();
        }else {
            this.cooling_and_heating_cal();
        }

    }else if( this.allTh_heat > this.allTc_heat && this.out_Qh_S_1 > 0){
        //暖房主体運転
        this.message.append("(C)暖主");
        //20181017
        if( this.is2018SinseiCHFree ) {
            this.message.append("(C)申請");
            this.heatingOnly_cal();
        }else {
            this.heating_and_cooling_cal();
        }
    }else{
        this.message.append("(C)能力=0停止");//20130625
        this.calc_Stop();
    }

    break;
default:
    this.message.append("(C)運転モード？停止");//20130625
    this.calc_Stop();
    //System.out.println(this.moduleName + ">>Error<< onOff*modeが範囲外");
}

if( this.isFanOnOff && this.PPEcmp == 0 ){
    this.PPEfan = 0;
}
this.PPE = this.PPEcmp + this.PPEfan;
this.E_PPE = this.PPE * Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 );
this.eleIn.setActivePower( this.PPE );
this.eleIn.setReactivePower( this.E_PPE );
this.eleIn.setPhase( this.phase );
this.eleIn.setVoltage( this.voltage );
this.eleIn.setFrequency( this.frequency );

//出力設定
```

```
        for(int i=0;i<num;i++){
            this.RMdata = this.rmlinemap.get( this.rmnm[i] );
            this.RMdata.set_CAPrateC( this.CAPrateC );
            this.RMdata.set_CAPrateH( this.CAPrateH );
            this.rmlinemap.put( this.rmnm[i], this.RMdata );
        }
        super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );

        super.sm.setState( super.getConnectionNode( this.S_NODE_airOutEA ), this.airOutEA );

        //グラフデータ追加
        if( this.isGVisible ){
            this.gBMout.addData( BestTimeManager.getDateWeatherTime(),
            this.Sc_Loadout,//室外機処理可能熱量（冷却）[W]
            this.Sh_Loadout,//室外機処理可能熱量（加熱）[W]
            this.allTc_heat,//室内機処理要求熱量（冷却）[W]
            this.allTh_heat,//室内機処理要求熱量（加熱）[W]
            this.PPE,
            this.DBoa,
            this.DBra,
            this.XGoa,
            this.XGra,
            this.CAPrateC.doubleValue() );
        }

        //記録ノード
        if( this.isRecord && super.rm != null ){
            this.record();
        }

        this.message.setLength(0);
    }


    private void calc_Stop(){
        this.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
        this.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
        this.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
        this.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.PPEcmp = 0.0;
        this.PPEfan = 0.0;
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );
    }


    /**
     * 記録
     */
    private void record(){
        //記録ノード
        if( super.rm != null ){
        if( CheckPrintModule.isPrintMessage ){
            //message
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_message, this.name, message.toString() );
        }

        if( CheckPrintModule.isPrintEnergy ){
            //記録ノードに室外機消費電力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID4, this.name, AirFormat.df_2(this.PPE));
            //記録ノードに室外機消費電力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_PPEcmp, this.name,
AirFormat.df_2(this.PPEcmp));
            //記録ノードに室外機消費電力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_PPEfan, this.name,
```

```
AirFormat.df_2(this.PPEfan));
      }

      if( CheckPrintModule.isPrintLoad ){
         //記録ノードに室外機実処理熱量（全熱）を設定
         if( this.Tc_Loadout > 0 ){
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT, this.name, AirFormat.df_2(-
this.Tc_Loadout));//室外機冷却実処理熱量/現状#W#熱量
         }else{
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT, this.name,
AirFormat.df_2(this.Th_Loadout));//室外機加熱実処理熱量/現状#W#熱量
         }
      }

      if( CheckPrintModule.isPrintStateOut ){
         //記録ノードに室外機外気入口DBを設定
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBairOutHS, this.name,
AirFormat.df_2(this.airOutEA.getTempDB()) );
         //記録ノードに室外機外気入口WBを設定
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_WBairOutHS, this.name,
AirFormat.df_2(this.airOutEA.getTempWB()) );
         //記録ノードに室外機外気入口流量を設定
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_M_airOutHS, this.name,
AirFormat.df_2(this.airOutEA.getFlowRate()) );
      }

      if( CheckPrintModule.isPrintStateMy ){
         //記録ノードに室内機加熱能力補正比率を設定
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID3c, this.name,
AirFormat.df_2(this.CAPrateC));
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID3h, this.name,
AirFormat.df_2(this.CAPrateH));
         //記録ノードに室内機要求顕熱量合計を設定
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID1c, this.name,
AirFormat.df_2(this.allSc_heat));//室内機冷却要求顕熱量合計#W#熱量
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID1h, this.name,
AirFormat.df_2(this.allSh_heat));//室内機加熱要求顕熱量合計#W#熱量
         //記録ノードに室内機要求全熱量合計を設定
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID2c, this.name,
AirFormat.df_2(this.allTc_heat));//室内機冷却要求全熱量合計#W#熱量
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID2h, this.name,
AirFormat.df_2(this.allTh_heat));//室内機加熱要求全熱量合計#W#熱量
         //記録ノードに室外機処理可能熱量（全熱）を設定
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID5c, this.name,
AirFormat.df_2(this.Sc_Loadout));//室外機冷却可能熱量/最大#W#熱量
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID5h, this.name,
AirFormat.df_2(this.Sh_Loadout));//室外機加熱可能熱量/最大#W#熱量
         //記録ノードに室外機実処理熱量（全熱）を設定
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID6c, this.name,
AirFormat.df_2(this.Tc_Loadout));//室外機冷却実処理熱量/現状#W#熱量
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID6h, this.name,
AirFormat.df_2(this.Th_Loadout));//室外機加熱実処理熱量/現状#W#熱量
         //
      }

      if( CheckPrintModule.isPrintStateIn ){
         //記録ノードに室外機外気入口DBを設定
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBairInHS, this.name,
AirFormat.df_2(this.airInOA.getTempDB()) );
         //記録ノードに室外機外気入口WBを設定
         super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_WBairInHS, this.name,
AirFormat.df_2(this.airInOA.getTempWB()) );
         //記録ノードに室外機外気入口流量を設定
         //super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_M_airInHS, this.name,
AirFormat.df_2(this.airInOA.getFlowRate()) );
      }

      if( CheckPrintModule.isPrintAdjust ){
         if( this.isAdjust2012 ){
            //記録ノードに室外機調整能力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qc_Adjust, this.name,
this.out_Qc_Adjust );//室外機調整能力[W]
```

```
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qh_Adjust, this.name,
this.out_Qh_Adjust );//室外機調整能力[W]
            //記録ノードに室外機調整能力を設定
            //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rcAdjust", this.name, this.out_rcAdjust );//
室外機調整率[-]
            //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rhAdjust", this.name, this.out_rhAdjust );//
室外機調整率[-]
            //記録ノードに室外機調整特性補正を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qc_Adjusthosei, this.name,
this.out_Qc_Adjusthosei );//室外機調整能力特性補正[-]
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qh_Adjusthosei, this.name,
this.out_Qh_Adjusthosei );//室外機調整能力特性補正[-]
         }

      }

      }

   }

   /**
    * ビルマルチ計算/各室内機の集計後
    */

   @Override
   public void update() {

      if( !this.isCalc ){
         return;
      }

      double rCode0 = 1.;
      double rCode1 = 1.;

      //能力補正係数のうち室外機側分を求める
      //機種によっては次のステップで室内機側の能力設定に使用することがある

      if( this.isHighSH || this.isBESTESTtable ){
         return;
      }
      //CoolingOnly
      rCode1 = this.pacDB.getKcta( this.DBoa);
      for( int i=0; i<this.num; i++ ){//20110623nino
         this.RMdata = this.rmlinemap.get( this.rmnm[i] );
         this.RMdata.set_out_c_CAPrevision( rCode0 * rCode1 );
      }
      //this.RMdata.set_out_c_CAPrevision( rCode0 * rCode1 );

      //Cooling and heating
      //rCode1 = this.pacDB.getKcmta( this.DBoa);//***

      //Heating_only
      rCode1 = this.pacDB.getKhta( this.WBoa);
      for( int i=0; i<this.num; i++ ){//20110623nino
         this.RMdata = this.rmlinemap.get( this.rmnm[i] );
         this.RMdata.set_out_h_CAPrevision( rCode0 * rCode1 );
      }
      //  this.RMdata.set_out_h_CAPrevision( rCode0 * rCode1 );

      //Heating and cooling
       //rCode1 = this.pacDB.getKhta( this.WBoa );

   }

   public Object viewInternal(TestCommand cmd) {

      ArrayList<Object> result = new ArrayList<Object>();
      result.add(super.sm);
      result.add(super.cm);
      result.add(super.rm);
      //外部定義
      result.add(this.name);
```

```
    result.add(this.out_Qc_S);
    result.add(this.out_Qc_C);
    result.add(this.out_PPEc_S);
    result.add(this.out_PPEc_C);
    result.add(this.out_Qh_S);
    result.add(this.out_Qh_C);
    result.add(this.out_Qh_L);
    result.add(this.out_PPEh_S);
    result.add(this.out_PPEh_C);
    result.add(this.out_PPEh_L);

    result.add(this.Lp);
    result.add(this.Lh);

    result.add(this.DBoa);
    result.add(this.XGoa);
    result.add(this.DBra);
    result.add(this.XGra);

    return result;
}


private void coolingOnly_cal(){
    if( this.isBESTESTtable ){
        this.coolingOnly_cal_BESTESTtable();

    }else if( this.isHighSH ){
        this.coolingOnly_cal_HighSH();

    }else{
        this.coolingOnly_cal_S();
    }
}

private void coolingOnly_cal_S(){
    double rCode0,rCode1,rCode2,rCode3;
    double D_fmc,D_pmc,alph,beta;
    double Rp;

    D_fmc    = this.fmc;
    D_pmc    = this.pmc;

    //上下限のチェック 20120821nino
    double dbOA = this.DBoa;
    double wbRA = this.WBra;

    double dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcta() );
    if( dbOA > dCheck ){
        //外気乾球温度＞上限の時停止
        this.message.append( "(C)外気DB＞"+dCheck+"上限→停止");
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.PPEcmp = 0.0;
        this.PPEfan = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );
        return;
    }
    dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcta() );
    if( dbOA < dCheck ){
        //外気乾球温度＜下限の時　下限値の特性で運転
        this.message.append( "(C)外気DB＜"+dCheck+"下限→下限値で特性計算");
        dbOA = dCheck;
    }
    dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
    if( wbRA > dCheck ){
```

```
   //室内湿球温度＞上限の時　上限値の特性で運転
   this.message.append( "(C)室WB＞"+dCheck+"上限→上限値で特性計算");
   wbRA = dCheck;
}
dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
if( wbRA < dCheck ){
   //室内湿球温度＜下限の時　下限値の特性で運転
   this.message.append( "(C)室WB＜"+dCheck+"下限→下限値で特性計算");
   wbRA = dCheck;
}

//室温補正
//能力補正

rCode1 = this.pacDB.getKcta( dbOA );
rCode2 = this.pacDB.getKcti( wbRA );
rCode3 = this.Lratemin;

this.Sc_Loadout = rCode1 * rCode2 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

         //System.out.println("Tc_Loadout="+Tc_Loadout+" Lratemin"+Lratemin+" / allTc_heat="+this.allTc_heat );

this.Rc = this.allTc_heat / this.Sc_Loadout;
this.CAPrateC = 1.0;
this.CAPrateH = 0.0;
this.Tc_Loadout = this.allTc_heat;//室外機実処理熱量（冷却全熱）[W]//20110622nino
if( this.Rc > 1.0 ) {
   this.CAPrateC = 1.0/ this.Rc;
   this.Rc = 1.0;
   this.Tc_Loadout = this.Sc_Loadout;//室外機実処理熱量（冷却全熱）[W]//20110622nino
}

//低負荷領域の計算仕分け:Rh
if( this.Rc < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = this.Rc;
}

         //System.out.println("Rc="+Rc+" "+D_fmc+" "+D_pmc);

alph = this.pacDB.getAlphac( Rp );
beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc );

if( D_pmc * D_fmc < 0.001 ){
   beta = 1.0;
}

rCode0 = alph * beta;
rCode1 = this.pacDB.getKchpid( Rp );
rCode2 = this.pacDB.getKcwta( dbOA );

//       System.out.println("WBra="+WBra);

rCode3 = this.pacDB.getKcwti( wbRA );

this.PPEcmp = rCode0 *rCode1 * rCode2 * rCode3 * this.out_PPEc_S;
//低負荷領域の計算仕分け:PPE
if( this.Rc < this.in_run_stop && Rp > 0 ){
```

```
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
               this.PPEcmp = 0.0;
               this.PPEfan = 0.0;
               break;

            case 1://  1_下限入力値固定
               //何もしない
               break;

            case 2://  2_下限COP値固定
               this.PPEcmp *= ( this.Rc / Rp );
               this.PPEfan = this.out_PPEfan;
               break;

            case 3://  3_下限入力値と中間切片
               this.PPEcmp *= ( 0.5 + this.Rc / Rp / 2. );
               this.PPEfan = this.out_PPEfan;
               break;

            default:
            }
         }else{
            //何もしない
         }

         if( this.isFanOnOff ){//20141009 BESTEST
            this.PPEfan *= this.Rc;
         }


         //airOutEA
         this.airOutEA.setFlowRate( this.fRate_airEAc );
         this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPEcmp + this.PPEfan ) / this.fRate_airEAc +
this.airInOA.getTempDB() );


         if( this.Rc < 0.0001 ){
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPEcmp = 0.0;
            this.PPEfan = 0.0;
            this.airOutEA.setFlowRate( 0. );
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );

         }else if( this.Rc < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
               case 0://  0_発停運転
                  this.CAPrateC = 0.0;
                  this.CAPrateH = 0.0;
                  this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
                  this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
                  this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
                  this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
                  this.PPEcmp = 0.0;
                  this.PPEfan = 0.0;
                  this.airOutEA.setFlowRate( 0. );
                  this.airOutEA.setTempDB( this.airInOA.getTempDB() );
                  break;

               case 1://  1_下限入力値固定
                  break;

               case 2://  2_下限COP値固定
                  break;

               case 3://  3_下限入力値と中間切片
                  break;
```

```
            default:
        }

    }


    //        System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

}


/**
 * 高顕熱型　のcoolingOnly_cal
 */
private void coolingOnly_cal_HighSH(){
    double rCode0,rCode1,rCode2,rCode3;
    double D_fmc,D_pmc,alph,beta;
    double Rp;

    D_fmc    = this.fmc;
    D_pmc    = this.pmc;

    //上下限のチェック 20120821nino
    double dbOA = this.DBoa;
    double wbRA = this.WBra;

    double dCheck = this.pacDB.getformulaRangeMaxRaw( this.pacDB.getFACKctai() );
    if( dbOA > dCheck ){
        //外気乾球温度＞上限の時停止
        this.message.append( "(C)外気DB＞"+dCheck+"上限→停止");
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.PPEcmp = 0.0;
        this.PPEfan = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );
        return;
    }
    dCheck = this.pacDB.getformulaRangeMinRaw( this.pacDB.getFACKctai() );
    if( dbOA < dCheck ){
        //外気乾球温度＜下限の時　下限値の特性で運転
        this.message.append( "(C)外気DB＜"+dCheck+"下限→下限値で特性計算");
        dbOA = dCheck;
    }
    dCheck = this.pacDB.getformulaRangeMaxCol( this.pacDB.getFACKctai() );
    if( wbRA > dCheck ){
        //室内湿球温度＞上限の時　上限値の特性で運転
        this.message.append( "(C)室WB＞"+dCheck+"上限→上限値で特性計算");
        wbRA = dCheck;
    }
    dCheck = this.pacDB.getformulaRangeMinCol( this.pacDB.getFACKctai() );
    if( wbRA < dCheck ){
        //室内湿球温度＜下限の時　下限値の特性で運転
        this.message.append( "(C)室WB＜"+dCheck+"下限→下限値で特性計算");
        wbRA = dCheck;
    }

    //室温補正
    //能力補正

    rCode1 = this.pacDB.getKctai( dbOA, wbRA );
    rCode3 = this.Lratemin;

    this.Sc_Loadout = rCode1 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

            //System.out.println("Tc_Loadout="+Tc_Loadout+" Lratemin"+Lratemin+" / allTc_heat="+this.allTc_heat );
```

```
this.Rc = this.allTc_heat / this.Sc_Loadout;
this.CAPrateC = 1.0;
this.CAPrateH = 0.0;
this.Tc_Loadout = this.allTc_heat;//室外機実処理熱量（冷却全熱）[W]//20110622nino
if( this.Rc > 1.0 ) {
   this.CAPrateC = 1.0/ this.Rc;
   this.Rc = 1.0;
   this.Tc_Loadout = this.Sc_Loadout;//室外機実処理熱量（冷却全熱）[W]//20110622nino
}

//低負荷領域の計算仕分け:Rh
if( this.Rc < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = this.Rc;
}

         //System.out.println("Rc="+Rc+" "+D_fmc+" "+D_pmc);

alph = this.pacDB.getAlphac( Rp );
beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc );

if( D_pmc * D_fmc < 0.001 ){
   beta = 1.0;
}

rCode0 = alph * beta;
rCode1 = this.pacDB.getKchpid( Rp );
rCode2 = this.pacDB.getKcwtai( dbOA, wbRA );

this.PPEcmp = rCode0 *rCode1 * rCode2 * this.out_PPEc_S;
//低負荷領域の計算仕分け:PPE
if( this.Rc < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.PPEcmp = 0.0;
      this.PPEfan = 0.0;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      this.PPEcmp *= ( this.Rc / Rp );
      this.PPEfan = this.out_PPEfan;
      break;

   case 3://  3_下限入力値と中間切片
      this.PPEcmp *= ( 0.5 + this.Rc / Rp / 2. );
      this.PPEfan = this.out_PPEfan;
      break;

   default:
   }
}else{
   //何もしない
}
```

```
      if( this.isFanOnOff ){//20141009 BESTEST
         this.PPEfan *= this.Rc;
      }

      //airOutEA
      this.airOutEA.setFlowRate( this.fRate_airEAc );
      this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPEcmp + this.PPEfan ) / this.fRate_airEAc +
this.airInOA.getTempDB() );


      if( this.Rc < 0.0001 ){
         this.CAPrateC = 0.0;
         this.CAPrateH = 0.0;
         this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
         this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
         this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
         this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
         this.PPEcmp = 0.0;
         this.PPEfan = 0.0;
         this.airOutEA.setFlowRate( 0. );
         this.airOutEA.setTempDB( this.airInOA.getTempDB() );

      }else if( this.Rc < this.in_run_stop ){
         switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
               this.CAPrateC = 0.0;
               this.CAPrateH = 0.0;
               this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
               this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
               this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
               this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
               this.PPEcmp = 0.0;
               this.PPEfan = 0.0;
               this.airOutEA.setFlowRate( 0. );
               this.airOutEA.setTempDB( this.airInOA.getTempDB() );
               break;

            case 1://  1_下限入力値固定
               break;

            case 2://  2_下限COP値固定
               break;

            case 3://  3_下限入力値と中間切片
               break;

            default:
         }

      }


      //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

   }

   /**
    * BESTESTtable型  のcoolingOnly_cal
    */
   private void coolingOnly_cal_BESTESTtable(){
      double rCode0,rCode1,rCode2,rCode3;
      double D_fmc,D_pmc,alph,beta;
      double Rp;

      D_fmc    = this.fmc;
      D_pmc    = this.pmc;

      //上下限のチェック 20120821nino
      double dbOA = this.DBoa;
      double wbRA = this.WBra;
      double dbRA = this.DBra;
```

```
      double dCheck =
BM_EHPdata.outDBListSC[this.RMdata.getN_outDB()][ BM_EHPdata.outDBListSC[this.RMdata.getN_outDB()].length - 1];
      if( dbOA > dCheck ){
         //外気乾球温度＞上限の時停止
         this.message.append( "(C)外気DB＞"+dCheck+"上限→停止");
         this.CAPrateC = 0.0;
         this.CAPrateH = 0.0;
         this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
         this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
         this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
         this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
         this.PPEcmp = 0.0;
         this.PPEfan = 0.0;
         this.airOutEA.setFlowRate( 0. );
         this.airOutEA.setTempDB( this.airInOA.getTempDB() );
         return;
      }
      dCheck = BM_EHPdata.outDBListSC[this.RMdata.getN_outDB()][0];
      if( dbOA < dCheck ){
         //外気乾球温度＜下限の時　下限値の特性で運転
         this.message.append( "(C)外気DB＜"+dCheck+"下限→下限値で特性計算");
         dbOA = dCheck;
      }

      //BESTEST　の特性table
      dCheck =
this.pacDB.getformulaRangeMaxRaw( this.pacDB.getFRACKcw( BM_EHPdata.outDBstrListSC[this.RMdata.getN_outDB()][0]) );
      if( wbRA > dCheck ){
         //室内湿球温度＞上限の時　上限値の特性で運転
         this.message.append( "(C)室WB＞"+dCheck+"上限→上限値で特性計算");
         wbRA = dCheck;
      }
      dCheck =
this.pacDB.getformulaRangeMinRaw( this.pacDB.getFRACKcw( BM_EHPdata.outDBstrListSC[this.RMdata.getN_outDB()][0]));
      if( wbRA < dCheck ){
         //室内湿球温度＜下限の時　下限値の特性で運転
         this.message.append( "(C)室WB＜"+dCheck+"下限→下限値で特性計算");
         wbRA = dCheck;
      }

      //室温補正
      //能力補正

      //BESTESTtable 能力
      if( wbRA > dbRA ){
         wbRA = dbRA;
      }
      double r1 = this.pacDB.getRACKcw( wbRA, dbRA, dbOA, BM_EHPdata.outDBListTC[this.RMdata.getN_outDB()],
BM_EHPdata.outDBstrListTC[this.RMdata.getN_outDB()] );
      double r2 = this.pacDB.getRACKcw( wbRA, dbRA, dbOA, BM_EHPdata.outDBListSC[this.RMdata.getN_outDB()],
BM_EHPdata.outDBstrListSC[this.RMdata.getN_outDB()] );
      rCode1 = ( r1 > r2 )? r1 : r2;

      //System.out.println( " rCode1 =" + rCode1 + " wbRA="+wbRA+" dbRA="+dbRA+" dbOA="+dbOA);
      //rCode1 = this.pacDB.getKctai( dbOA, wbRA );
      rCode3 = this.Lratemin;

      this.Sc_Loadout = rCode1 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

            //System.out.println("Sc_Loadout="+Sc_Loadout+" rCode1="+rCode1+" Lratemin"+Lratemin+" /
allTc_heat="+this.allTc_heat );

      if( this.Sc_Loadout != 0 ){
         this.Rc = this.allTc_heat / this.Sc_Loadout;
      }else{
         this.Rc = 0;
      }
      this.CAPrateC = 1.0;
      this.CAPrateH = 0.0;
      this.Tc_Loadout = this.allTc_heat;//室外機実処理熱量（冷却全熱）[W]//20110622nino
      if( this.Rc > 1.0 ) {
```

```
            this.CAPrateC = 1.0/ this.Rc;
            this.Rc = 1.0;
            this.Tc_Loadout = this.Sc_Loadout;//室外機実処理熱量（冷却全熱）[W]//20110622nino
        }

        //低負荷領域の計算仕分け:Rh
        if( this.Rc < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                Rp = 0;
                break;

            case 1://  1_下限入力値固定
            case 2://  2_下限COP値固定
            case 3://  3_下限入力値と中間切片
                Rp = this.in_run_stop;
                break;

            default:
                Rp = 0;
            }
        }else{
            Rp = this.Rc;
        }

              // System.out.println("Rc="+Rc+" "+D_fmc+" "+D_pmc+"Rp="+Rp);

        alph = this.pacDB.getAlphac( Rp );
        beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc );

        if( D_pmc * D_fmc < 0.001 ){
            beta = 1.0;
        }

        rCode0 = alph * beta;
        rCode1 = this.pacDB.getKchpid( Rp );
        //rCode2 = this.pacDB.getKcwtai( dbOA, wbRA );
        rCode2 = this.pacDB.getRACKcw( wbRA, dbRA, dbOA, BM_EHPdata.outDBListWC[this.RMdata.getN_outDB()],
BM_EHPdata.outDBstrListWC[this.RMdata.getN_outDB()] );

        //System.out.println( "rCode2= " +rCode2 );
        this.PPEcmp = rCode0 *rCode1 * rCode2 * this.out_PPEc_S;
        //低負荷領域の計算仕分け:PPE
        if( this.Rc < this.in_run_stop && Rp > 0 ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                this.PPEcmp = 0.0;
                this.PPEfan = 0.0;
                break;

            case 1://  1_下限入力値固定
                //何もしない
                break;

            case 2://  2_下限COP値固定
                this.PPEcmp *= ( this.Rc / Rp );
                this.PPEfan = this.out_PPEfan;
                break;

            case 3://  3_下限入力値と中間切片
                this.PPEcmp *= ( 0.5 + this.Rc / Rp / 2. );
                this.PPEfan = this.out_PPEfan;
                break;

            default:
            }
        }else{
            //何もしない
        }


        if( this.isFanOnOff ){//20141009 BESTEST
```

```
            this.PPEfan *= this.Rc;
        }

        //airOutEA
        this.airOutEA.setFlowRate( this.fRate_airEAc );
        this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPEcmp + this.PPEfan ) / this.fRate_airEAc +
this.airInOA.getTempDB() );


        if( this.Rc < 0.0001 ){
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPEcmp = 0.0;
            this.PPEfan = 0.0;
            this.airOutEA.setFlowRate( 0. );
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );

        }else if( this.Rc < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
                case 0://  0_発停運転
                    this.CAPrateC = 0.0;
                    this.CAPrateH = 0.0;
                    this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
                    this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
                    this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
                    this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
                    this.PPEcmp = 0.0;
                    this.PPEfan = 0.0;
                    this.airOutEA.setFlowRate( 0. );
                    this.airOutEA.setTempDB( this.airInOA.getTempDB() );
                    break;

                case 1://  1_下限入力値固定
                    break;

                case 2://  2_下限COP値固定
                    break;

                case 3://  3_下限入力値と中間切片
                    break;

                default:
            }

        }


        //      System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

    }

    private void cooling_and_heating_cal(){
        if( this.isHighSH ){
            this.cooling_and_heating_cal_HighSH();
        }else{
            this.cooling_and_heating_cal_S();
        }
    }


    private void cooling_and_heating_cal_S(){
        double rCode0,rCode1,rCode2,rCode3;
        double D_fmc,D_pmc,alph,beta;
        double Rp;

        D_fmc    = this.fmc;
        D_pmc    = this.pmc;
```

```
//上下限のチェック 20120821nino
double dbOA = this.DBoa;
double wbRA = this.WBra;

double dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcmta() );
if( dbOA > dCheck ){
   //外気乾球温度＞上限の時停止
   this.message.append( "(C)外気DB＞"+dCheck+"上限→停止");
   this.CAPrateC = 0.0;
   this.CAPrateH = 0.0;
   this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
   this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
   this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
   this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
   this.PPEcmp = 0.0;
   this.PPEfan = 0.0;
   this.airOutEA.setFlowRate( 0. );
   this.airOutEA.setTempDB( this.airInOA.getTempDB() );
   return;
}
dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcmta() );
if( dbOA < dCheck ){
   //外気乾球温度＜下限の時　下限値の特性で運転
   this.message.append( "(C)外気DB＜"+dCheck+"下限→下限値で特性計算");
   dbOA = dCheck;
}
dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
if( wbRA > dCheck ){
   //室内湿球温度＞上限の時　上限値の特性で運転
   this.message.append( "(C)室WB＞"+dCheck+"上限→上限値で特性計算");
   wbRA = dCheck;
}
dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
if( wbRA < dCheck ){
   //室内湿球温度＜下限の時　下限値の特性で運転
   this.message.append( "(C)室WB＜"+dCheck+"下限→下限値で特性計算");
   wbRA = dCheck;
}

//冷房側　処理熱量
rCode1 = this.pacDB.getKcmta( dbOA );//***
rCode2 = this.pacDB.getKcti( wbRA );
rCode3 = this.Lratemin;

this.Sc_Loadout = rCode1 * rCode2 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

//容量比
this.Rc = this.allTc_heat / this.Sc_Loadout;

this.CAPrateC = 1.0;
this.Tc_Loadout = this.allTc_heat;//室外機実処理熱量（冷却全熱）[W]//20110622nino
if( this.Rc > 1.0 ) {
   this.CAPrateC = 1.0/ this.Rc;
   this.Rc = 1.0;
   this.Tc_Loadout = this.Sc_Loadout;//室外機実処理熱量（冷却全熱）[W]//20110622nino
}


//消費電力
//低負荷領域の計算仕分け:Rh
if( this.Rc < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;
```

```
   default:
      Rp = 0;
   }
}else{
   Rp = this.Rc;
}

alph = this.pacDB.getAlphac( Rp );
beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc );
//       System.out.println("Rc="+Rc+" "+D_fmc+" "+D_pmc);

if( D_pmc * D_fmc < 0.001 ){
   beta = 1.0;
}

rCode0 = alph * beta;
rCode1 = this.pacDB.getKchpid( Rp );
rCode2 = this.pacDB.getKcmwta( dbOA );//***

//       System.out.println("WBra="+WBra);

rCode3 = this.pacDB.getKcwti( wbRA );

this.PPEcmp = rCode0 *rCode1 * rCode2 * rCode3 * this.out_PPEc_S;
//低負荷領域の計算仕分け:PPE
if( this.Rc < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.PPEcmp = 0.0;
      this.PPEfan = 0.0;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      this.PPEcmp *= ( this.Rc / Rp );
      this.PPEfan = this.out_PPEfan;
      break;

   case 3://  3_下限入力値と中間切片
      this.PPEcmp *= ( 0.5 + this.Rc / Rp / 2. );
      this.PPEfan = this.out_PPEfan;
      break;

   default:
   }
}else{
   //何もしない
}


//暖房側　処理熱量
rCode2 = this.pacDB.getKhti( this.DBra );
rCode3 = this.Lratemin;

this.Sh_Loadout = rCode2 * rCode3 * this.out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

//容量比
this.Rh = this.allTh_heat / this.Sh_Loadout;

this.CAPrateH = 1.0;
this.Th_Loadout = this.allTh_heat;//室外機実処理熱量（加熱全熱）[W]//20110622nino
if( this.Rh > 1.0 ) {
   this.CAPrateH = 1.0/ this.Rh;
   this.Rh = 1.0;
   this.Th_Loadout = this.Sh_Loadout;//室外機実処理熱量（加熱全熱）[W]//20110622nino
}


if( this.isFanOnOff ){//20141009 BESTEST
```

```
            this.PPEfan *= this.Rc;
        }

        //airOutEA
        this.airOutEA.setFlowRate( this.fRate_airEAc );
        this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPEcmp + this.PPEfan ) / this.fRate_airEAc +
this.airInOA.getTempDB() );


        if( this.Rc < 0.0001 ){
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPEcmp = 0.0;
            this.PPEfan = 0.0;
            this.airOutEA.setFlowRate( 0. );
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );

        }else if( this.Rc < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
                case 0://  0_発停運転
                    this.CAPrateC = 0.0;
                    this.CAPrateH = 0.0;
                    this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
                    this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
                    this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
                    this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
                    this.PPEcmp = 0.0;
                    this.PPEfan = 0.0;
                    this.airOutEA.setFlowRate( 0. );
                    this.airOutEA.setTempDB( this.airInOA.getTempDB() );
                    break;

                case 1://  1_下限入力値固定
                    break;

                case 2://  2_下限COP値固定
                    break;

                case 3://  3_下限入力値と中間切片
                    break;

                default:
            }

        }

        //      System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

    }


    private void cooling_and_heating_cal_HighSH(){
        double rCode0,rCode1,rCode2,rCode3;
        double D_fmc,D_pmc,alph,beta;
        double Rp;

        D_fmc    = this.fmc;
        D_pmc    = this.pmc;

        //上下限のチェック 20120821nino
        double dbOA = this.DBoa;
        double wbRA = this.WBra;

        double dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcmta() );
        if( dbOA > dCheck ){
            //外気乾球温度＞上限の時停止
            this.message.append( "(C)外気DB＞"+dCheck+"上限→停止");
            this.CAPrateC = 0.0;
```

```
    this.CAPrateH = 0.0;
    this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
    this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
    this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
    this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
    this.PPEcmp = 0.0;
    this.PPEfan = 0.0;
    this.airOutEA.setFlowRate( 0. );
    this.airOutEA.setTempDB( this.airInOA.getTempDB() );
    return;
}
dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcmta() );
if( dbOA < dCheck ){
    //外気乾球温度＜下限の時　下限値の特性で運転
    this.message.append( "(C)外気DB＜"+dCheck+"下限→下限値で特性計算");
    dbOA = dCheck;
}
dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
if( wbRA > dCheck ){
    //室内湿球温度＞上限の時　上限値の特性で運転
    this.message.append( "(C)室WB＞"+dCheck+"上限→上限値で特性計算");
    wbRA = dCheck;
}
dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
if( wbRA < dCheck ){
    //室内湿球温度＜下限の時　下限値の特性で運転
    this.message.append( "(C)室WB＜"+dCheck+"下限→下限値で特性計算");
    wbRA = dCheck;
}

//冷房側　処理熱量
rCode1 = this.pacDB.getKcmta( dbOA );//***
rCode2 = this.pacDB.getKcti( wbRA );
rCode3 = this.Lratemin;

this.Sc_Loadout = rCode1 * rCode2 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

//容量比
this.Rc = this.allTc_heat / this.Sc_Loadout;

this.CAPrateC = 1.0;
this.Tc_Loadout = this.allTc_heat;//室外機実処理熱量（冷却全熱）[W]//20110622nino
if( this.Rc > 1.0 ) {
    this.CAPrateC = 1.0/ this.Rc;
    this.Rc = 1.0;
    this.Tc_Loadout = this.Sc_Loadout;//室外機実処理熱量（冷却全熱）[W]//20110622nino
}


//消費電力
//低負荷領域の計算仕分け:Rh
if( this.Rc < this.in_run_stop ){
    switch( this.num_calcTypeLowerRangeLoad){
    case 0://  0_発停運転
        Rp = 0;
        break;

    case 1://  1_下限入力値固定
    case 2://  2_下限COP値固定
    case 3://  3_下限入力値と中間切片
        Rp = this.in_run_stop;
        break;

    default:
        Rp = 0;
    }
}else{
    Rp = this.Rc;
}

alph = this.pacDB.getAlphac( Rp );
beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc );
```

```
//       System.out.println("Rc="+Rc+" "+D_fmc+" "+D_pmc);

if( D_pmc * D_fmc < 0.001 ){
   beta = 1.0;
}

rCode0 = alph * beta;
rCode1 = this.pacDB.getKchpid( Rp );
rCode2 = this.pacDB.getKcmwta( dbOA );//***

//       System.out.println("WBra="+WBra);

rCode3 = this.pacDB.getKcwti( wbRA );

this.PPEcmp = rCode0 *rCode1 * rCode2 * rCode3 * this.out_PPEc_S;
//低負荷領域の計算仕分け:PPE
if( this.Rc < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.PPEcmp = 0.0;
      this.PPEfan = 0.0;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      this.PPEcmp *= ( this.Rc / Rp );
      this.PPEfan = this.out_PPEfan;
      break;

   case 3://  3_下限入力値と中間切片
      this.PPEcmp *= ( 0.5 + this.Rc / Rp / 2. );
      this.PPEfan = this.out_PPEfan;
      break;

   default:
   }
}else{
   //何もしない
}


//暖房側  処理熱量
rCode2 = this.pacDB.getKhti( this.DBra );
rCode3 = this.Lratemin;

this.Sh_Loadout = rCode2 * rCode3 * this.out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

//容量比
this.Rh = this.allTh_heat / this.Sh_Loadout;

this.CAPrateH = 1.0;
this.Th_Loadout = this.allTh_heat;//室外機実処理熱量（加熱全熱）[W]//20110622nino
if( this.Rh > 1.0 ) {
   this.CAPrateH = 1.0/ this.Rh;
   this.Rh = 1.0;
   this.Th_Loadout = this.Sh_Loadout;//室外機実処理熱量（加熱全熱）[W]//20110622nino
}


if( this.isFanOnOff ){//20141009 BESTEST
   this.PPEfan *= this.Rc;
}

//airOutEA
this.airOutEA.setFlowRate( this.fRate_airEAc );
this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPEcmp + this.PPEfan ) / this.fRate_airEAc +
this.airInOA.getTempDB() );
```

```
    if( this.Rc < 0.0001 ){
       this.CAPrateC = 0.0;
       this.CAPrateH = 0.0;
       this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
       this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
       this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
       this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
       this.PPEcmp = 0.0;
       this.PPEfan = 0.0;
       this.airOutEA.setFlowRate( 0. );
       this.airOutEA.setTempDB( this.airInOA.getTempDB() );

    }else if( this.Rc < this.in_run_stop ){
       switch( this.num_calcTypeLowerRangeLoad){
          case 0://  0_発停運転
             this.CAPrateC = 0.0;
             this.CAPrateH = 0.0;
             this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
             this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
             this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
             this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
             this.PPEcmp = 0.0;
             this.PPEfan = 0.0;
             this.airOutEA.setFlowRate( 0. );
             this.airOutEA.setTempDB( this.airInOA.getTempDB() );
             break;

          case 1://  1_下限入力値固定
             break;

          case 2://  2_下限COP値固定
             break;

          case 3://  3_下限入力値と中間切片
             break;

          default:
       }

    }

    //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

}

private void heatingOnly_cal(){
    if( this.isHighSH ){
       this.heatingOnly_cal_HighSH();
    }else{
       this.heatingOnly_cal_S();
    }
}

private void heatingOnly_cal_S(){
    double rCode0,rCode1,rCode2,rCode3;
    double D_fmh,D_pmh,D_out_Qh_S,D_out_PPEh_S,alph,beta;
    double Rp;

    D_fmh    = this.fmh;
    D_pmh    = this.pmh;


    D_out_Qh_S  = this.out_Qh_S;
    D_out_PPEh_S= this.out_PPEh_S;
    if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L > 0.0001) { //低温入力の場合
       D_fmh    = this.fmh;
        D_pmh      = this.pmh;
        D_out_Qh_S = this.out_Qh_L;
        D_out_PPEh_S= this.out_PPEh_L;
    }

    //上下限のチェック 20120821nino
```

```
        double wbOA = this.WBoa;
        double dbRA = this.DBra;

        double dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() );
        if( wbOA > dCheck ){
           //外気湿球温度＞上限の時　上限値の特性で運転
           this.message.append( "(C)外気WB＞"+dCheck+"上限→上限値で特性計算");
           wbOA = dCheck;
        }
        dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhta() );
        if( wbOA < dCheck ){
           //外気湿球温度＜下限の時　停止
           this.message.append( "(C)外気WB＜"+dCheck+"下限→停止");
           this.CAPrateC = 0.0;
           this.CAPrateH = 0.0;
           this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
           this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
           this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
           this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
           this.PPEcmp = 0.0;
           this.PPEfan = 0.0;
           this.airOutEA.setFlowRate( 0. );
           this.airOutEA.setTempDB( this.airInOA.getTempDB() );
           return;
        }
        dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
        if( dbRA > dCheck ){
           //室内乾球温度＞上限の時　上限値の特性で運転
           this.message.append( "(C)室DB＞"+dCheck+"上限→上限値で特性計算");
           dbRA = dCheck;
        }
        dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
        if( dbRA < dCheck ){
           //室内乾球温度＜下限の時　下限値の特性で運転
           this.message.append( "(C)室DB＜"+dCheck+"下限→下限値で特性計算");
           dbRA = dCheck;
        }


        //暖房　処理熱量
        rCode1 = this.pacDB.getKhta( wbOA );
        rCode2 = this.pacDB.getKhti( dbRA );
        rCode3 = this.Lratemin;
        //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode1+" "+rCode2+" "+rCode3);

        //       System.out.println("allTheat="+allTheat+" T_Loadout="+T_Loadout+" Rh="+Rc+" "+rCode1+" "+rCode2+"
"+rCode3);


        this.Sh_Loadout  = rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

        this.Rh = this.allTh_heat / this.Sh_Loadout;
        this.CAPrateC = 0.0;
        this.CAPrateH = 1.0;
        this.Th_Loadout = this.allTh_heat;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        if( this.Rh > 1.0 ) {
           this.CAPrateH = 1.0 / this.Rh;
           this.Rh = 1.0;
           this.Th_Loadout = this.Sh_Loadout;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        }
        //       System.out.println("Rh="+Rc);

        //暖房　消費電力
        //低負荷領域の計算仕分け:Rh
        if( this.Rh < this.in_run_stop ){
           switch( this.num_calcTypeLowerRangeLoad){
           case 0://  0_発停運転
              Rp = 0;
              break;

           case 1://  1_下限入力値固定
           case 2://  2_下限COP値固定
```

```
            case 3://  3_下限入力値と中間切片
               Rp = this.in_run_stop;
               break;

            default:
               Rp = 0;
            }
         }else{
            Rp = this.Rh;
         }

         //System.out.println( "Rh="+Rh);
         alph = this.pacDB.getAlphah( Rp );
         beta = this.pacDB.getBetah( Rp, D_fmh, D_pmh );

         if( D_pmh * D_fmh < 0.001 ){
            beta = 1.0;
         }

         rCode0 = alph * beta;
         rCode1 = this.pacDB.getKhhpid( Rp );
         rCode2 = this.pacDB.getKhwta( wb0A );


         rCode3 = this.pacDB.getKhwti( dbRA );

         this.PPEcmp = rCode0 *rCode1 * rCode2 * rCode3 * D_out_PPEh_S;
         //低負荷領域の計算仕分け:PPE
         if( this.Rh < this.in_run_stop && Rp > 0 ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
               this.PPEcmp = 0.0;
               this.PPEfan = 0.0;
               break;

            case 1://  1_下限入力値固定
               //何もしない
               break;

            case 2://  2_下限COP値固定
               this.PPEcmp *= ( this.Rh / Rp );
               this.PPEfan = this.out_PPEfan;
               break;

            case 3://  3_下限入力値と中間切片
               this.PPEcmp *= ( 0.5 + this.Rh / Rp / 2. );
               this.PPEfan = this.out_PPEfan;
               break;

            default:
            }
         }else{
            //何もしない
         }


         if( this.isFanOnOff ){//20141009 BESTEST
            this.PPEfan *= this.Rh;
         }

         //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);

         //airOutEA
         this.airOutEA.setFlowRate( this.fRate_airEAh );
         this.airOutEA.setTempDB( ( -this.Th_Loadout + this.PPEcmp + this.PPEfan ) / this.fRate_airEAh +
this.airInOA.getTempDB() );

         if( this.Rh < 0.0001 ){
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
```

```
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.PPEcmp = 0.0;
        this.PPEfan = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );

    }else if( this.Rh < this.in_run_stop ){
        switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                this.CAPrateC = 0.0;
                this.CAPrateH = 0.0;
                this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
                this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
                this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
                this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
                this.PPEcmp = 0.0;
                this.PPEfan = 0.0;
                this.airOutEA.setFlowRate( 0. );
                this.airOutEA.setTempDB( this.airInOA.getTempDB() );
                break;

            case 1://  1_下限入力値固定
                break;

            case 2://  2_下限COP値固定
                break;

            case 3://  3_下限入力値と中間切片
                break;

            default:
        }

    }
        //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
}

private void heatingOnly_cal_HighSH(){
    double rCode0,rCode1,rCode2,rCode3;
    double D_fmh,D_pmh,D_out_Qh_S,D_out_PPEh_S,alph,beta;
    double Rp;

    D_fmh    = this.fmh;
    D_pmh    = this.pmh;


    D_out_Qh_S  = this.out_Qh_S;
    D_out_PPEh_S= this.out_PPEh_S;
    if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L > 0.0001) { //低温入力の場合
        D_fmh    = this.fmh;
         D_pmh      = this.pmh;
         D_out_Qh_S = this.out_Qh_L;
         D_out_PPEh_S= this.out_PPEh_L;
    }

    //上下限のチェック 20120821nino
    double wbOA = this.WBoa;
    double dbRA = this.DBra;

    double dCheck = this.pacDB.getformulaRangeMaxRaw( this.pacDB.getFACKhtai() );
    if( wbOA > dCheck ){
        //外気湿球温度＞上限の時　上限値の特性で運転
        this.message.append( "(C)外気WB＞"+dCheck+"上限→上限値で特性計算");
        wbOA = dCheck;
    }
    dCheck = this.pacDB.getformulaRangeMinRaw( this.pacDB.getFACKhtai() );
    if( wbOA < dCheck ){
        //外気湿球温度＜下限の時　停止
        this.message.append( "(C)外気WB＜"+dCheck+"下限→停止");
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
```

```
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPEcmp = 0.0;
            this.PPEfan = 0.0;
            this.airOutEA.setFlowRate( 0. );
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );
            return;
        }
        dCheck = this.pacDB.getformulaRangeMaxCol( this.pacDB.getFACKhtai() );
        if( dbRA > dCheck ){
            //室内乾球温度＞上限の時　上限値の特性で運転
            this.message.append( "(C)室DB＞"+dCheck+"上限→上限値で特性計算");
            dbRA = dCheck;
        }
        dCheck = this.pacDB.getformulaRangeMinCol( this.pacDB.getFACKhtai() );
        if( dbRA < dCheck ){
            //室内乾球温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)室DB＜"+dCheck+"下限→下限値で特性計算");
            dbRA = dCheck;
        }


        //暖房　処理熱量
        rCode1 = this.pacDB.getKhtai( wbOA, dbRA );
        rCode3 = this.Lratemin;
        //        System.out.println("WBoa="+wbOA+" DBra="+dbRA+" rCode1="+rCode1+" rCode3="+rCode3);
        //        System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode1+" "+rCode2+" "+rCode3);

        //        System.out.println("allTheat="+allTheat+" T_Loadout="+T_Loadout+" Rh="+Rc+" "+rCode1+" "+rCode2+"
"+rCode3);


        this.Sh_Loadout  = rCode1 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

        this.Rh = this.allTh_heat / this.Sh_Loadout;
        this.CAPrateC = 0.0;
        this.CAPrateH = 1.0;
        this.Th_Loadout = this.allTh_heat;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        if( this.Rh > 1.0 ) {
            this.CAPrateH = 1.0 / this.Rh;
            this.Rh = 1.0;
            this.Th_Loadout = this.Sh_Loadout;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        }
        //        System.out.println("Rh="+Rc);

        //暖房　消費電力
        //低負荷領域の計算仕分け:Rh
        if( this.Rh < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                Rp = 0;
                break;

            case 1://  1_下限入力値固定
            case 2://  2_下限COP値固定
            case 3://  3_下限入力値と中間切片
                Rp = this.in_run_stop;
                break;

            default:
                Rp = 0;
            }
        }else{
            Rp = this.Rh;
        }

        //System.out.println( "Rh="+Rh);
        alph = this.pacDB.getAlphah( Rp );
        beta = this.pacDB.getBetah( Rp, D_fmh, D_pmh );
```

```
        if( D_pmh * D_fmh < 0.001 ){
           beta = 1.0;
        }

        rCode0 = alph * beta;
        rCode1 = this.pacDB.getKhhpid( Rp );
        rCode2 = this.pacDB.getKhwtai( wbOA, dbRA );

        this.PPEcmp = rCode0 *rCode1 * rCode2 * D_out_PPEh_S;
        //低負荷領域の計算仕分け:PPE
        if( this.Rh < this.in_run_stop && Rp > 0 ){
           switch( this.num_calcTypeLowerRangeLoad){
           case 0://  0_発停運転
              this.PPEcmp = 0.0;
              this.PPEfan = 0.0;
              break;

           case 1://  1_下限入力値固定
              //何もしない
              break;

           case 2://  2_下限COP値固定
              this.PPEcmp *= ( this.Rh / Rp );
              this.PPEfan = this.out_PPEfan;
              break;

           case 3://  3_下限入力値と中間切片
              this.PPEcmp *= ( 0.5 + this.Rh / Rp / 2. );
              this.PPEfan = this.out_PPEfan;
              break;

           default:
           }
        }else{
           //何もしない
        }


        if( this.isFanOnOff ){//20141009 BESTEST
           this.PPEfan *= this.Rh;
        }
        //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);

        //airOutEA
        this.airOutEA.setFlowRate( this.fRate_airEAh );
        this.airOutEA.setTempDB( ( -this.Th_Loadout + this.PPEcmp + this.PPEfan ) / this.fRate_airEAh +
this.airInOA.getTempDB() );

        if( this.Rh < 0.0001 ){
           this.CAPrateC = 0.0;
           this.CAPrateH = 0.0;
           this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
           this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
           this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
           this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
           this.PPEcmp = 0.0;
           this.PPEfan = 0.0;
           this.airOutEA.setFlowRate( 0. );
           this.airOutEA.setTempDB( this.airInOA.getTempDB() );

        }else if( this.Rh < this.in_run_stop ){
           switch( this.num_calcTypeLowerRangeLoad){
              case 0://  0_発停運転
                 this.CAPrateC = 0.0;
                 this.CAPrateH = 0.0;
                 this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
                 this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
                 this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
                 this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
                 this.PPEcmp = 0.0;
                 this.PPEfan = 0.0;
                 this.airOutEA.setFlowRate( 0. );
```

```
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );
            break;

         case 1://  1_下限入力値固定
            break;

         case 2://  2_下限COP値固定
            break;

         case 3://  3_下限入力値と中間切片
            break;

         default:
      }

   }
      //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
}


private void heating_and_cooling_cal(){
   if( this.isHighSH ){
      this.heating_and_cooling_cal_HighSH();
   }else{
      this.heating_and_cooling_cal_S();
   }
}

private void heating_and_cooling_cal_S(){
   double rCode0,rCode1,rCode2,rCode3;
   double D_fmh,D_pmh;
   double D_out_Qh_S, D_out_Qc_S;
   double D_out_PPEh_S,alph,beta;
   double deltaWB;
   double Rch;
   double Rp;

    D_fmh      = this.fmh;
    D_pmh      = this.pmh;
    D_out_Qh_S = this.out_Qh_S;
    D_out_Qc_S = this.out_Qc_S;
    D_out_PPEh_S= this.out_PPEh_S;

    if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L > 0.0001) { //低温入力の場合
      D_fmh    = this.fmh;
       D_pmh      = this.pmh;
       D_out_Qh_S = this.out_Qh_L;
       D_out_PPEh_S= this.out_PPEh_L;
    }

   //上下限のチェック 20120821nino
   double wbOA = this.WBoa;
   double dbRA = this.DBra;

   double dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() );
   if( wbOA > dCheck ){
      //外気湿球温度＞上限の時　上限値の特性で運転
      this.message.append( "(C)外気WB＞"+dCheck+"上限→上限値で特性計算");
      wbOA = dCheck;
   }
   dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhta() );
   if( wbOA < dCheck ){
      //外気湿球温度＜下限の時　停止
      this.message.append( "(C)外気WB＜"+dCheck+"下限→停止");
      this.CAPrateC = 0.0;
      this.CAPrateH = 0.0;
      this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
      this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
      this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
      this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
      this.PPEcmp = 0.0;
      this.PPEfan = 0.0;
```

```
            this.airOutEA.setFlowRate( 0. );
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );
            return;
        }
        dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
        if( dbRA > dCheck ){
            //室内乾球温度＞上限の時　上限値の特性で運転
            this.message.append( "(C)室DB＞"+dCheck+"上限→上限値で特性計算");
            dbRA = dCheck;
        }
        dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
        if( dbRA < dCheck ){
            //室内乾球温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)室DB＜"+dCheck+"下限→下限値で特性計算");
            dbRA = dCheck;
        }


         //暖房　処理熱量
         //定格暖房容量
         rCode1 = this.pacDB.getKhta( wbOA );
        rCode2 = this.pacDB.getKhti( dbRA );
        rCode3 = this.Lratemin;
        this.Sh_Loadout  = rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

        Rch = this.allTc_heat / this.Sh_Loadout;
        deltaWB = this.pacDB.getKdwb( Rch );

        //        System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode1+" "+rCode2+" "+rCode3);

        //        System.out.println("allTheat="+allTheat+" T_Loadout="+T_Loadout+" Rh="+Rc+" "+rCode1+" "+rCode2+"
"+rCode3);

        //補正暖房容量
         rCode1 = this.pacDB.getKhmta( wbOA + deltaWB );
        this.Sh_Loadout  = rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

        this.Rh = this.allTh_heat / this.Sh_Loadout;
        this.CAPrateH = 1.0;
        this.Th_Loadout = this.allTh_heat;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        if( this.Rh > 1.0 ) {
            this.CAPrateH = 1.0 / this.Rh;
            this.Rh = 1.0;
            this.Th_Loadout = this.Sh_Loadout;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        }
        //        System.out.println("Rh="+Rc);

        //暖房　消費電力
        //低負荷領域の計算仕分け:Rh
        if( this.Rh < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                Rp = 0;
                break;

            case 1://  1_下限入力値固定
            case 2://  2_下限COP値固定
            case 3://  3_下限入力値と中間切片
                Rp = this.in_run_stop;
                break;

            default:
                Rp = 0;
            }
        }else{
            Rp = this.Rh;
        }

        alph = this.pacDB.getAlphah( Rp );
        beta = this.pacDB.getBetah( Rp, D_fmh, D_pmh );

        if( D_pmh * D_fmh < 0.001 ){
```

```
            beta = 1.0;
        }

        rCode0 = alph * beta;
        rCode1 = this.pacDB.getKhhpid( Rp );
        rCode2 = this.pacDB.getKhmwta( wbOA + deltaWB );
        rCode3 = this.pacDB.getKhwti( dbRA );

        this.PPEcmp = rCode0 * rCode1 * rCode2 * rCode3 * D_out_PPEh_S;
        //低負荷領域の計算仕分け:PPE
        if( this.Rh < this.in_run_stop && Rp > 0 ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                this.PPEcmp = 0.0;
                this.PPEfan = 0.0;
                break;

            case 1://  1_下限入力値固定
                //何もしない
                break;

            case 2://  2_下限COP値固定
                this.PPEcmp *= ( this.Rh / Rp );
                this.PPEfan = this.out_PPEfan;
                break;

            case 3://  3_下限入力値と中間切片
                this.PPEcmp *= ( 0.5 + this.Rh / Rp / 2. );
                this.PPEfan = this.out_PPEfan;
                break;

            default:
            }
        }else{
            //何もしない
        }


        //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);

        //冷房  処理熱量
        rCode2 = this.pacDB.getKcti( this.WBra);
        rCode3 = this.Lratemin;

        this.Sc_Loadout  = rCode2 * rCode3 * D_out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]
        this.Rc = this.allTc_heat / this.Sc_Loadout;
        this.CAPrateC = 1.0;
        this.Tc_Loadout = this.allTc_heat;//室外機実処理熱量（冷却全熱）[W]//20110622nino
        if( this.Rc > 1.0 ) {
            this.CAPrateC = 1.0 / this.Rc;
            this.Rc = 1.0;
            this.Tc_Loadout = this.Sc_Loadout;//室外機実処理熱量（冷却全熱）[W]//20110622nino
        }


        if( this.isFanOnOff ){//20141009 BESTEST
            this.PPEfan *= this.Rh;
        }
        //airOutEA
        this.airOutEA.setFlowRate( this.fRate_airEAh );
        this.airOutEA.setTempDB( ( -this.Tc_Loadout + this.PPEcmp + this.PPEfan ) / this.fRate_airEAh +
this.airInOA.getTempDB() );


        if( this.Rh < 0.0001 ){
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPEcmp = 0.0;
```

```
        this.PPEfan = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );

    }else if( this.Rh < this.in_run_stop ){
        switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                this.CAPrateC = 0.0;
                this.CAPrateH = 0.0;
                this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
                this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
                this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
                this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
                this.PPEcmp = 0.0;
                this.PPEfan = 0.0;
                this.airOutEA.setFlowRate( 0. );
                this.airOutEA.setTempDB( this.airInOA.getTempDB() );
                break;

            case 1://  1_下限入力値固定
                break;

            case 2://  2_下限COP値固定
                break;

            case 3://  3_下限入力値と中間切片
                break;

            default:
        }

    }
        //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
}


private void heating_and_cooling_cal_HighSH(){//20181009
    double rCode0,rCode1,rCode2,rCode3;
    double D_fmh,D_pmh;
    double D_out_Qh_S, D_out_Qc_S;
    double D_out_PPEh_S,alph,beta;
    double deltaWB;
    double Rch;
    double Rp;

     D_fmh      = this.fmh;
     D_pmh      = this.pmh;
     D_out_Qh_S = this.out_Qh_S;
     D_out_Qc_S = this.out_Qc_S;
     D_out_PPEh_S= this.out_PPEh_S;

     if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L > 0.0001) { //低温入力の場合
       D_fmh    = this.fmh;
        D_pmh      = this.pmh;
        D_out_Qh_S = this.out_Qh_L;
        D_out_PPEh_S= this.out_PPEh_L;
     }

    //上下限のチェック 20120821nino
    double wbOA = this.WBoa;
    double dbRA = this.DBra;

    double dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() );
    if( wbOA > dCheck ){
        //外気湿球温度＞上限の時　上限値の特性で運転
        this.message.append( "(C)外気WB＞"+dCheck+"上限→上限値で特性計算");
        wbOA = dCheck;
    }
    dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhta() );
    if( wbOA < dCheck ){
        //外気湿球温度＜下限の時　停止
        this.message.append( "(C)外気WB＜"+dCheck+"下限→停止");
```

```
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPEcmp = 0.0;
            this.PPEfan = 0.0;
            this.airOutEA.setFlowRate( 0. );
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );
            return;
        }
        dCheck = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
        if( dbRA > dCheck ){
            //室内乾球温度＞上限の時　上限値の特性で運転
            this.message.append( "(C)室DB＞"+dCheck+"上限→上限値で特性計算");
            dbRA = dCheck;
        }
        dCheck = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
        if( dbRA < dCheck ){
            //室内乾球温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)室DB＜"+dCheck+"下限→下限値で特性計算");
            dbRA = dCheck;
        }


         //暖房　処理熱量
         //定格暖房容量
         rCode1 = this.pacDB.getKhta( wbOA );
        rCode2 = this.pacDB.getKhti( dbRA );
        rCode3 = this.Lratemin;
        this.Sh_Loadout  = rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

        Rch = this.allTc_heat / this.Sh_Loadout;
        deltaWB = this.pacDB.getKdwb( Rch );

        //        System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode1+" "+rCode2+" "+rCode3);

        //        System.out.println("allTheat="+allTheat+" T_Loadout="+T_Loadout+" Rh="+Rc+" "+rCode1+" "+rCode2+"
"+rCode3);

        //補正暖房容量
         rCode1 = this.pacDB.getKhmta( wbOA + deltaWB );
        this.Sh_Loadout  = rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

        this.Rh = this.allTh_heat / this.Sh_Loadout;
        this.CAPrateH = 1.0;
        this.Th_Loadout = this.allTh_heat;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        if( this.Rh > 1.0 ) {
            this.CAPrateH = 1.0 / this.Rh;
            this.Rh = 1.0;
            this.Th_Loadout = this.Sh_Loadout;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        }
        //        System.out.println("Rh="+Rc);

        //暖房　消費電力
        //低負荷領域の計算仕分け:Rh
        if( this.Rh < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                Rp = 0;
                break;

            case 1://  1_下限入力値固定
            case 2://  2_下限COP値固定
            case 3://  3_下限入力値と中間切片
                Rp = this.in_run_stop;
                break;

            default:
                Rp = 0;
            }
```

```
        }else{
           Rp = this.Rh;
        }

        alph = this.pacDB.getAlphah( Rp );
        beta = this.pacDB.getBetah( Rp, D_fmh, D_pmh );

        if( D_pmh * D_fmh < 0.001 ){
           beta = 1.0;
        }

        rCode0 = alph * beta;
        rCode1 = this.pacDB.getKhhpid( Rp );
        rCode2 = this.pacDB.getKhmwta( wbOA + deltaWB );
        rCode3 = this.pacDB.getKhwti( dbRA );

        this.PPEcmp = rCode0 * rCode1 * rCode2 * rCode3 * D_out_PPEh_S;
        //低負荷領域の計算仕分け:PPE
        if( this.Rh < this.in_run_stop && Rp > 0 ){
           switch( this.num_calcTypeLowerRangeLoad){
           case 0://  0_発停運転
              this.PPEcmp = 0.0;
              this.PPEfan = 0.0;
              break;

           case 1://  1_下限入力値固定
              //何もしない
              break;

           case 2://  2_下限COP値固定
              this.PPEcmp *= ( this.Rh / Rp );
              this.PPEfan = this.out_PPEfan;
              break;

           case 3://  3_下限入力値と中間切片
              this.PPEcmp *= ( 0.5 + this.Rh / Rp / 2. );
              this.PPEfan = this.out_PPEfan;
              break;

           default:
           }
        }else{
           //何もしない
        }


        //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);

        //冷房  処理熱量
        rCode2 = this.pacDB.getKcti( this.WBra);
        rCode3 = this.Lratemin;

        this.Sc_Loadout  = rCode2 * rCode3 * D_out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]
        this.Rc = this.allTc_heat / this.Sc_Loadout;
        this.CAPrateC = 1.0;
        this.Tc_Loadout = this.allTc_heat;//室外機実処理熱量（冷却全熱）[W]//20110622nino
        if( this.Rc > 1.0 ) {
           this.CAPrateC = 1.0 / this.Rc;
           this.Rc = 1.0;
           this.Tc_Loadout = this.Sc_Loadout;//室外機実処理熱量（冷却全熱）[W]//20110622nino
        }


        if( this.isFanOnOff ){//20141009 BESTEST
           this.PPEfan *= this.Rh;
        }
        //airOutEA
        this.airOutEA.setFlowRate( this.fRate_airEAh );
        this.airOutEA.setTempDB( ( -this.Tc_Loadout + this.PPEcmp + this.PPEfan ) / this.fRate_airEAh +
this.airInOA.getTempDB() );
```

```
        if( this.Rh < 0.0001 ){
           this.CAPrateC = 0.0;
           this.CAPrateH = 0.0;
           this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
           this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
           this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
           this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
           this.PPEcmp = 0.0;
           this.PPEfan = 0.0;
           this.airOutEA.setFlowRate( 0. );
           this.airOutEA.setTempDB( this.airInOA.getTempDB() );

        }else if( this.Rh < this.in_run_stop ){
           switch( this.num_calcTypeLowerRangeLoad){
              case 0://  0_発停運転
                 this.CAPrateC = 0.0;
                 this.CAPrateH = 0.0;
                 this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
                 this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
                 this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
                 this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
                 this.PPEcmp = 0.0;
                 this.PPEfan = 0.0;
                 this.airOutEA.setFlowRate( 0. );
                 this.airOutEA.setTempDB( this.airInOA.getTempDB() );
                 break;

              case 1://  1_下限入力値固定
                 break;

              case 2://  2_下限COP値固定
                 break;

              case 3://  3_下限入力値と中間切片
                 break;

              default:
           }

        }
           //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
     }


     private double WB_cal( double ddb, double xx ){
        double wwb, x1, de = 1;
        int kk = -1, j = 0;
        wwb = ddb;

        do{
           j++;
           x1 = Psychrometrics.FNXtw( ddb, wwb );
           if( Math.abs(x1-xx) < 0.000003 ) {
//            System.out.println(" j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
              return wwb;
           }
           if( x1 < xx ) {
              if( kk == -1 ){
                 de = de / 2.0;
              }
              wwb = wwb + de;
              kk = 1;
           }
              if(x1 >= xx ) {
              if( kk == 1 ){
                 de = de / 2.0;
              }
              wwb = wwb - de;
              kk = -1;
           }

        }while( j < 1000 );
```

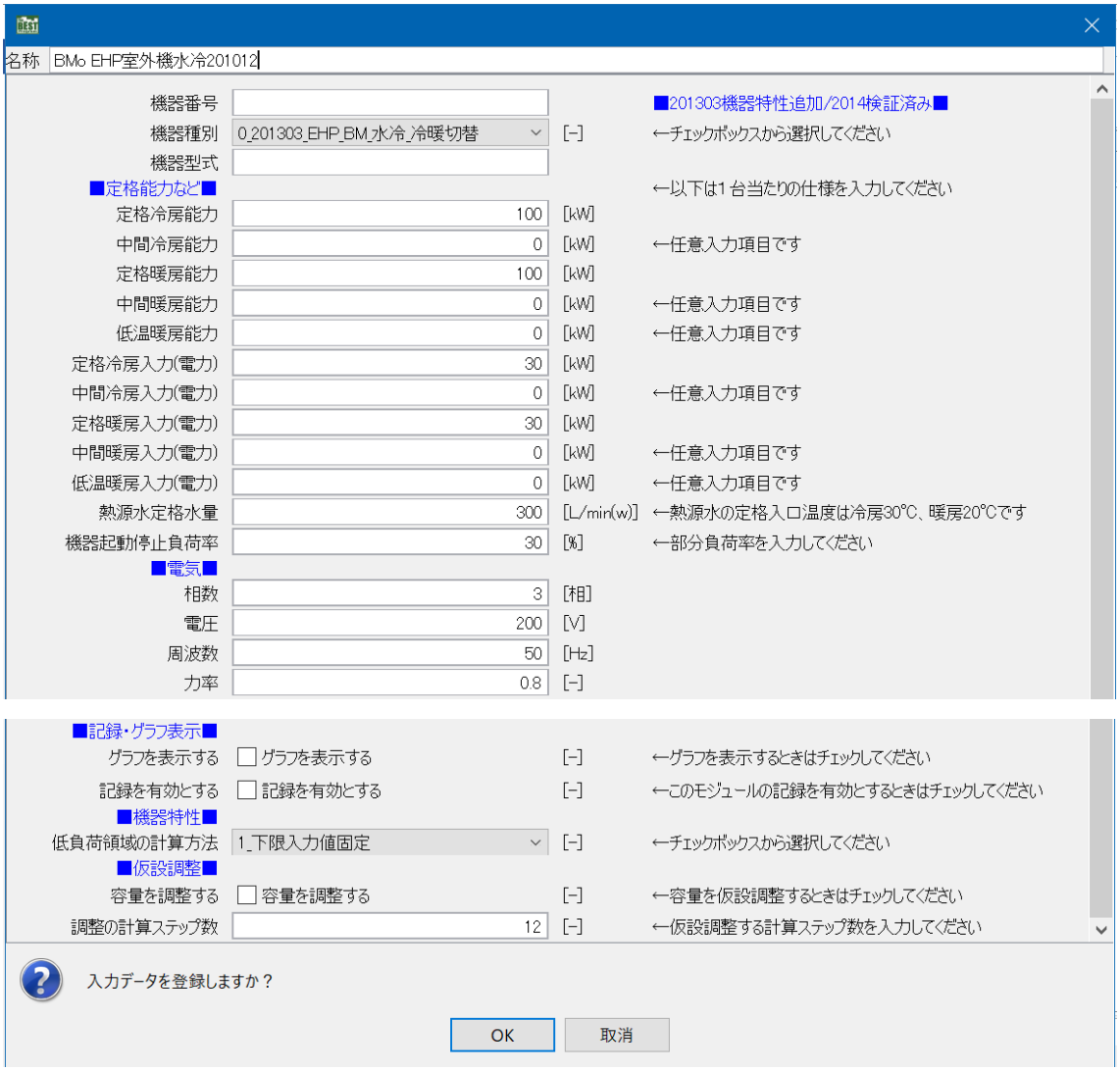
```
        //    System.out.println("*j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
        return wwb;

    }
}
```

「BMo EHP 室外機水冷 201012」（場所：設備 2015／個別分散 2015／）

モジュール名	BMo EHP 室外機水冷 201012
クラス	BuillMultiEHPOut_WaterCooledModule20101212

（1）入力画面

・スペック



（2）モジュールの概要

水冷タイプのビル用マルチの室外機モジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	－	[-]	－	－		
1	機器番号	String	m_no			－	－		
2	機器種別	String	m_ty	0_201303_EHP_BM_水冷_冷暖切替、1_201303_EHP_BM_水冷_冷暖同時、0_EHP_BM_水冷_冷暖切替20100627、1_EHP_BM_水冷_冷暖同時20100627	[-]	－	－		←チェックボックスから選択してください
3	機器型式	String	m_kt			－	－		
4	<html><font color=blue>■定格能力など■					－	－		←以下は１台当たりの仕様を入力してください
5	定格冷房能力	double	out_Qc_S	100	[kW]	－	0		
6	中間冷房能力	double	out_Qc_C	0	[kW]	－	0		←任意入力項目です
7	定格暖房能力	double	out_Qh_S	100	[kW]	－	0		
8	中間暖房能力	double	out_Qh_C	0	[kW]	－	0		←任意入力項目です
9	低温暖房能力	double	out_Qh_L	0	[kW]	－	0		←任意入力項目です
10	定格冷房入力(電力)	double	out_PPEc_S	30	[kW]	－	0		
11	中間冷房入力(電力)	double	out_PPEc_C	0	[kW]	－	0		←任意入力項目です
12	定格暖房入力(電力)	double	out_PPEh_S	30	[kW]	－	0		

13	中間暖房入力(電力)	double	out_PPEh_C	0	[kW]	―	0		←任意入力項目です
14	低温暖房入力(電力)	double	out_PPEh_L	0	[kW]	―	0		←任意入力項目です
15	熱源水定格水量	double	de_fRate_watInHS	300	[L/min(w)]	―	0		←熱源水の定格入口温度は冷房 30℃、暖房 20℃です
16	機器起動停止負荷率	double	in_run_stop	30	[%]	100	0		←部分負荷率を入力してください
17	■電気■					―	―		
18	相数	int	phase	3	[相]	―	1		
19	電圧	double	voltage	200	[V]	―	100		
20	周波数	double	frequency	50	[Hz]	60	50		
21	力率	double	powerFactor	0.8	[-]	1	0		
22	■記録・グラフ表示■					―	―		
23	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
24	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください
25	■機器特性■					―	―		
26	低負荷領域の計算方法	String	calcTypeLowerRangeLoad	0_発停運転、1_下限入力値固定、2_下限 COP 値固定、3_下限入力値と中間切片	[-]	―	―		←チェックボックスから選択してください
27	■仮設調整■					―	―		
28	容量を調整する	boolean	isAdjust2012	FALSE	[-]	―	―		←容量を仮設調整するときはチェックしてください
29	調整の計算ステップ数	int	numAdjustSteps	12	[-]	―	1		←仮設調整する計算ステップ数を入力してください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	熱源水入口	L0_watInHS	watInHS	-	状態	水	入口	
5	熱源水出口	L0_watOutHS	watOutHS	-	状態	水	出口	
6	冷媒入口	L0_valInLine	valInLine	-	値	値	入口	
7	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	室外機 Message#-#-	メッセージ	—	メッセージ
2	室内機冷却要求顕熱量合計#W#熱量	室内機冷却要求顕熱量合計	W	My
3	室内機加熱要求顕熱量合計#W#熱量	室内機加熱要求顕熱量合計	W	My
4	室内機冷却要求全熱量合計#W#熱量	室内機冷却要求全熱量合計	W	My
5	室内機加熱要求全熱量合計#W#熱量	室内機加熱要求全熱量合計	W	My
6	室内機冷却能力補正比率	室内機冷却能力補正比率	—	My
7	室内機加熱能力補正比率	室内機加熱能力補正比率	—	My
8	室外機消費電力#W#消費電力	室外機消費電力	W	エネルギー消費
9	室外機冷却可能熱量/最大#W#熱量	室外機冷却可能熱量/最大	W	My
10	室外機加熱可能熱量/最大#W#熱量	室外機加熱可能熱量/最大	W	My
11	室外機冷却実処理熱量/現状#W#熱量	室外機冷却実処理熱量/現状	W	負荷
12	室外機加熱実処理熱量/現状#W#熱量	室外機加熱実処理熱量/現状	W	負荷
13	処理全熱量合計負荷#W#-	処理全熱量合計負荷	W	My
14	室外機熱源水入口温度#℃#温度	室外機熱源水入口温度	℃	入口
15	室外機熱源水出口温度#℃#温度	室外機熱源水出口温度	℃	出口
16	室外機熱源水入口質量流量#g/s#質量流量	室外機熱源水入口質量流量	g/s	入口
17	室外機熱源水出口質量流量#g/s#質量流量	室外機熱源水出口質量流量	g/s	出口
18	室外機調整冷却能力#W#熱量	室外機調整冷却能力	W	調整
19	室外機調整加熱能力#W#熱量	室外機調整加熱能力	W	調整

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Map;
import java.util.Set;

import jp.or.ibec.best.DO.BM_EHPdata;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.essentials.utility.Psychrometrics;

/**
 * @author NINOMIYA/20101212
 * 水冷ビルマルチ：室外機クラス
 *
 * 水冷ビルマルチ/20101212nino
 * 20130329機器特性追加
 */
public class BuillMultiEHPOut_WaterCooledModule20101212 extends AbstractBestModule implements
IBestMetaModule  {

   /**
    * 論理アドレス
    */
   private final String moduleName="(BuillMultiEHPOut_WaterCooledModule20101212)";
   //private final String S_NODE_airInOA    = "LO_airInOA";//外気
   //private final String S_NODE_airOutEA   = "LO_airOutEA";//排気
   private final String S_NODE_watInHS    = "LO_watInHS";//熱源水入口
   private final String S_NODE_watOutHS   = "LO_watOutHS";//熱源水出口
   private final String S_NODE_valInLine  = "LO_valInLine";//単線
   private final String S_NODE_eleIn      = "LO_eleIn";//電力
   private final String C_NODE_swcIn = "L1_swcIn";//運転状態：off/on
   private final String C_NODE_modIn = "L1_modIn";//空調モード：停止/冷却/加熱
   private final String R_NODE = "L2_recOut";
   private Map<String,BM_EHPdata> rmlinemap=null;

   /**
    * 外部定義
    */
   private final String SPEC_name          = "名称";
   private final String SPEC_m_no          = "機器番号";
   private final String SPEC_m_ty          = "機器種別";
   private final String SPEC_m_kt          = "機器型式";
   private final String SPEC_out_Qc_S      = "定格冷房能力[W]";
   private final String SPEC_out_Qc_C      = "中間冷房能力[W]";
   private final String SPEC_out_PPEc_S    = "定格冷房入力(電力)[W]";
   private final String SPEC_out_PPEc_C    = "中間冷房入力(電力)[W]";
   private final String SPEC_out_Qh_S      = "定格暖房能力[W]";
   private final String SPEC_out_Qh_C      = "中間暖房能力[W]";
   private final String SPEC_out_Qh_L      = "低温暖房能力[W]";
   private final String SPEC_out_PPEh_S    = "定格暖房入力(電力)[W]";
   private final String SPEC_out_PPEh_C    = "中間暖房入力(電力)[W]";
   private final String SPEC_out_PPEh_L    = "低温暖房入力(電力)[W]";
/*
   private final String SPEC_out_CLEa_R    = "クランクケースヒータ(運転時)[W]";
   private final String SPEC_out_CLEa_S    = "クランクケースヒータ(停止時)[W]";
```

```
    private final String SPEC_out_TKEa_R    = "定格待機電力(運転時)[W]";
    private final String SPEC_out_TKEa_T    = "定格待機電力(待機時)[W]";
    private final String SPEC_out_TKEa_S    = "定格待機電力(停止時)[W]";
*/
    private final String SPEC_de_fRate_watInHS   = "熱源水定格水量[g/s]";

    private final String SPEC_in_run_stop  = "機器起動停止負荷率[-]";
    private final String SPEC_Phase       = "相数[-]";
    private final String SPEC_Voltage     = "電圧[V]";
    private final String SPEC_Frequency   = "周波数[Hz]";
    private final String SPEC_PowerFactor = "力率[-]";
    //
    private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
    private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

    private final String SPEC_CalcTypeLowerRangeLoad = "低負荷領域の計算方法";

    private final String SPEC_isAdjust2012   = "容量を調整する";
    private final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";

    private StringBuffer message= new StringBuffer();

// private BestAir airInOA  = null;
// private BestAir airOutEA = null;
    private BestWater watInHS  = null;
    private BestWater watOutHS = null;
    private BestElectricity eleIn = null;  //電力
    private Double CAPrateC =  new Double( 1 );//室内機能力補正比率[-]冷房側
    private Double CAPrateH =  new Double( 1 );//室内機能力補正比率[-]暖房側
    private BM_EHPdata RMdata = null;
    private PACDBManager pacDB=null;

    private int modIn  ;
    private int swcIn ;
    private int mState;  //停止、冷房、暖房フラグ

    private boolean isCOOLandHEATout = false;//冷暖同時機種の場合=true
    private boolean isWaterCooled    = false;//水冷の場合=true
    private int dbflg=0;
    private boolean is2018SinseiCHFree = false;//2018冷暖同時誘導基準申請計算モード


    /**
     * 変数 (外部定義に対応)
     */

    private String name              ;// "機器名称";
    private String m_no              ;//= "機器番号";
    private String m_ty              ;//= "機器種別";
    private String m_kt              ;//= "機器型式";
    private double out_Qc_S      ;//= "定格冷房能力";//kW
    private double out_Qc_C      ;//= "中間冷房能力";//kW
    private double out_PPEc_S    ;//= "定格冷房入力(電力)";//kW
    private double out_PPEc_C    ;//= "中間冷房入力(電力)";//kW
    private double out_Qh_S      ;//= "定格暖房能力";//kW
    private double out_Qh_C      ;//= "中間暖房能力";//kW
    private double out_Qh_L      ;//= "低温暖房能力";//kW
    private double out_PPEh_S    ;//= "定格暖房入力(電力)";//kW
    private double out_PPEh_C    ;//= "中間暖房入力(電力)";//kW
    private double out_PPEh_L    ;//= "低温暖房入力(電力)";//kW
/*
    private double out_CLEa_R=0.0    ;//= "クランクケースヒータ(運転時)";//W
    private double out_CLEa_S=0.0    ;//= "クランクケースヒータ(停止時)";//W
    private double out_TKEa_R =0.0   ;//= "定格待機電力(運転時)";//W
    private double out_TKEa_T =0.0   ;//= "定格待機電力(待機時)";//W
    private double out_TKEa_S =0.0  ;//= "定格待機電力(停止時)";//W
*/
    //private double fRate_airEA ;//風量[g/s]

    private double in_run_stop    ;//= "機器起動停止負荷率";//[%]
    private int phase;         //相数[-]
    private double voltage;    //電圧[V]
```

```
    private double frequency;  //周波数[Hz]
    private double powerFactor;   //力率[-]

    private double Lp;          //冷媒管長さ[m]
    private double Lh;          //室内機/室外機高低差[m]
    private boolean isGVisible = false;//このグラフを表示する=true
    private boolean isRecord   = false;//記録を有効とする=true

    private String[] rmnm=new String[50];

    /**
     * 出力（建物）
     */

    private final String RECORD_message = "室外機Message#-#-";
    private final String RECORD_allSc_heat = "室内機冷却要求顕熱量合計#W#熱量";
    private final String RECORD_allSh_heat = "室内機加熱要求顕熱量合計#W#熱量";
    private final String RECORD_allTc_heat = "室内機冷却要求全熱量合計#W#熱量";
    private final String RECORD_allTh_heat = "室内機加熱要求全熱量合計#W#熱量";
    private final String RECORD_CAPrateC = "室内機冷却能力補正比率";
    private final String RECORD_CAPrateH = "室内機加熱能力補正比率";
    private final String RECORD_ID4  = "室外機消費電力#W#消費電力";
    private final String RECORD_Sc_Loadout = "室外機冷却可能熱量/最大#W#熱量";
    private final String RECORD_Sh_Loadout = "室外機加熱可能熱量/最大#W#熱量";
    private final String RECORD_Tc_Loadout = "室外機冷却実処理熱量/現状#W#熱量";
    private final String RECORD_Th_Loadout = "室外機加熱実処理熱量/現状#W#熱量";
    private final String RECORD_qT         = "処理全熱量合計負荷#W#-";

    private final String RECORD_t_watInHS = "室外機熱源水入口温度#℃#温度";
    private final String RECORD_t_watOutHS = "室外機熱源水出口温度#℃#温度";
    private final String RECORD_fRate_watInHS = "室外機熱源水入口質量流量#g/s#質量流量";
    private final String RECORD_fRate_watOutHS = "室外機熱源水出口質量流量#g/s#質量流量";

    private final String RECORD_out_Qc_Adjust = "室外機調整冷却能力#W#熱量";
    private final String RECORD_out_Qh_Adjust = "室外機調整加熱能力#W#熱量";

    /**
     * その他変数
     */

    private double Rc;              //部分負荷率  冷房側
    private double Rh;              //部分負荷率  暖房側

// private double DBoa;          //外気乾球温度[℃]
// private double WBoa;          //外気湿球温度[℃]
// private double XGoa;          //外気絶対湿度[g/gD.A.]

    private double DBra;          //室内機吸込乾球温度[℃]
    private double WBra;          //室内機吸込湿球温度[℃]
    private double XGra;          //室内機吸込絶対湿度[g/gD.A.]

    private double t_watInHS;// = this.watInHS.getTemp();
    private double t_watOutHS;// = this.watOutHS.getTemp();
    private double fRate_watInHS;// = this.watInHS.getFlowRate();
    private double de_fRate_watInHS;// 定格熱源水の水量[g/s]

    private double Sc_Loadout;//室外機処理可能熱量（冷却全熱）[W]
    private double Sh_Loadout;//室外機処理可能熱量（加熱全熱）[W]
    private double Tc_Loadout;//室外機実処理熱量（冷却全熱）[W]
    private double Th_Loadout;//室外機実処理熱量（加熱全熱）[W]
    private double allSc_heat;//室内機処理熱量（冷却顕熱）[W]
    private double allSh_heat;//室内機処理熱量（加熱顕熱）[W]
    private double allTc_heat;//室内機処理熱量（冷却全熱）[W]
    private double allTh_heat;//室内機処理熱量（加熱全熱）[W]

    private double PPE;  //室外機消費電力[W]
    private double E_PPE;//室外機無効電力[]

    private int num;
    private String kiki_file;
    private String path="EHP";
    private String[] filenames= new String[1];
```

```
    private String equipmentName;

    private double DBra_max;
    private double WBra_min;
    private double Lratemin;
    private double fmc;
    private double pmc;
    private double fmh;
    private double pmh;

    //グラフ表示など
    private GraphJFrameBuilMultiOut_S20101212 gBMout = null;

    //低負荷領域の計算方法
    private String calcTypeLowerRangeLoad = null;//"低負荷領域の計算方法";
    private int num_calcTypeLowerRangeLoad;//"低負荷領域の計算方法";

    //仮設調整モード
    //仮設調整モード2012
    private boolean isAdjust2012  = false;//true="台数を調整する";//
    //private boolean isAdjustMode = true;//20110517nino 空衛学会OS論文検討用＝true
    private double out_Qc_Adjust;//= "調整冷却能力";
    private double out_Qh_Adjust;//= "調整加熱能力";
    private double fRate_watInHSc ;//冷却時水量[g/s]
    private double fRate_watInHSh ;//加熱時水量[g/s]

    private int numAdjustSteps;//"調整の計算ステップ数";
    private LinkedList<Double> rcList = null;//必要台数
    private LinkedList<Double> rhList = null;//必要台数
    private double max_rcAdjust = 1.;
    private double max_rhAdjust = 1.;
    private double ave_rcAdjust = 0;
    private double ave_rhAdjust = 0;
    private double out_rcAdjust = 1.;//= "調整率冷房";
    private double out_rhAdjust = 1.;//= "調整率暖房";
// private double in_run_stop_Num;//= "機器起動停止負荷率"台数補正;

    private double out_Qc_S_1      ;//= "定格冷房能力";//kW
    private double out_Qc_C_1      ;//= "中間冷房能力";//kW
    private double out_PPEc_S_1    ;//= "定格冷房入力(電力)";//kW
    private double out_PPEc_C_1    ;//= "中間冷房入力(電力)";//kW
    private double out_Qh_S_1      ;//= "定格暖房能力";//kW
    private double out_Qh_C_1      ;//= "中間暖房能力";//kW
    private double out_Qh_L_1      ;//= "低温暖房能力";//kW
    private double out_PPEh_S_1    ;//= "定格暖房入力(電力)";//kW
    private double out_PPEh_C_1    ;//= "中間暖房入力(電力)";//kW
    private double out_PPEh_L_1    ;//= "低温暖房入力(電力)";//kW


    @Override
    public void setProfile(BestSpecs spec) {
       //    外部定義項目取得
       if(spec == null) {
          return;
       }
       Map<String,String> map=spec.getSpec();
       if(map == null) {
          return;
       }

       // 機器名称
       this.name = spec.getSpecValue( this.SPEC_name, this.name );

       // 機器番号
       this.m_no = spec.getSpecValue( this.SPEC_m_no, "" );

       //  機器種別  0_標準型、1_寒冷地型、2_店舗用、3_設備用
       this.m_ty = spec.getSpecValue( this.SPEC_m_ty, "0_201303_EHP_BM_水冷_冷暖切替" );

       //"0_EHP_BM_水冷_冷暖切替20100627"      //20111027nino
       //this.kiki_file = "EHP_BM_WaterCooled_20100627";
       this.kiki_file = "EHP_BM_WaterCooled_201303";
```

```
        this.isCOOLandHEATout = false;
        this.isWaterCooled    = true;

        //
        if( this.m_ty.equals("0_201303_EHP_BM_水冷_冷暖切替")){
           this.kiki_file = "EHP_BM_WaterCooled_201303";
           this.isCOOLandHEATout = false;
        }
        if( this.m_ty.equals("1_201303_EHP_BM_水冷_冷暖同時")){
           this.kiki_file = "EHP_BM_WaterCooled_201303";
           this.isCOOLandHEATout = true;
        }
        if( this.m_ty.equals("1_201303_EHP_BM_水冷_冷暖同時2018申請")){//20181017
           this.kiki_file = "EHP_BM_WaterCooled_201303";
           this.isCOOLandHEATout = true;
           this.is2018SinseiCHFree = true;
        }

        //
        if( this.m_ty.equals("0_EHP_BM_水冷_冷暖切替20100627")){
           this.kiki_file = "EHP_BM_WaterCooled_20100627";
           this.isCOOLandHEATout = false;
        }
        if( this.m_ty.equals("1_EHP_BM_水冷_冷暖同時20100627")){
           this.kiki_file = "EHP_BM_WaterCooled_20100627";
           this.isCOOLandHEATout = true;
        }


        // 機器型式
        this.m_kt = spec.getSpecValue( this.SPEC_m_kt, "" );

        // 定格冷房能力
        this.out_Qc_S = spec.getSpecValue( this.SPEC_out_Qc_S, 0. );

        // 中間冷房能力
        this.out_Qc_C = spec.getSpecValue( this.SPEC_out_Qc_C, 0. );

        // 定格冷房入力(電力)
        this.out_PPEc_S = spec.getSpecValue( this.SPEC_out_PPEc_S, 0. );

        // 中間冷房入力(電力)
        this.out_PPEc_C = spec.getSpecValue( this.SPEC_out_PPEc_C, 0. );

        // 定格暖房能力
        this.out_Qh_S = spec.getSpecValue( this.SPEC_out_Qh_S, 0. );

        // 中間暖房能力
        this.out_Qh_C = spec.getSpecValue( this.SPEC_out_Qh_C, 0. );

        // 低温暖房能力
        this.out_Qh_L = spec.getSpecValue( this.SPEC_out_Qh_L, 0. );

        // 定格暖房入力(電力)
        this.out_PPEh_S = spec.getSpecValue( this.SPEC_out_PPEh_S, 0. );

        // 中間暖房入力(電力)
        this.out_PPEh_C = spec.getSpecValue( this.SPEC_out_PPEh_C, 0. );

        // 低温暖房入力(電力)
        this.out_PPEh_L = spec.getSpecValue( this.SPEC_out_PPEh_L, 0. );

    /*
        // クランクケースヒータ(運転時)
        this.out_CLEa_R = spec.getSpecValue( this.SPEC_out_CLEa_R, 0. );

        // クランクケースヒータ(停止時)
        this.out_CLEa_S = spec.getSpecValue( this.SPEC_out_CLEa_S, 0. );

        // 待機電力(運転時)
        this.out_TKEa_R = spec.getSpecValue( this.SPEC_out_TKEa_R, 0. );
```

```
        // 待機電力(待機時)
        this.out_TKEa_T = spec.getSpecValue( this.SPEC_out_TKEa_T, 0. );

        // 待機電力(停止時)
        this.out_TKEa_S = spec.getSpecValue( this.SPEC_out_TKEa_S, 0. );
*/
        //SPEC_de_fRate_watInHS   = "熱源水定格水量[g/s]";
        this.de_fRate_watInHS = spec.getSpecValue( this.SPEC_de_fRate_watInHS, this.out_Qc_S * 860 / 5/
3.6 );//20110505nino

        // 機器起動停止負荷率
        this.in_run_stop = spec.getSpecValue( this.SPEC_in_run_stop, 0.3 );

        // 相数
        this.phase = spec.getSpecValue( this.SPEC_Phase, 3 );

        // 電圧
        this.voltage = spec.getSpecValue( this.SPEC_Voltage, 200. );

        // 周波数
        this.frequency = spec.getSpecValue( this.SPEC_Frequency, 50. );

        // 力率
        this.powerFactor = spec.getSpecValue( this.SPEC_PowerFactor, 0.8 );

        //isGVisibleを取得
        this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

        //低負荷領域の計算方法
        this.calcTypeLowerRangeLoad = spec.getSpecValue( this.SPEC_CalcTypeLowerRangeLoad, "0_発停運転" );
        //calcTypeLowerRangeLoad
        //  0_発停運転
        //  1_下限入力値固定
        //  2_下限COP値固定
        //  3_下限入力値と中間切片
        if( this.calcTypeLowerRangeLoad.equals( "0_発停運転" )){
           this.num_calcTypeLowerRangeLoad = 0;
        }else if(this.calcTypeLowerRangeLoad.equals( "1_下限入力値固定" )){
           this.num_calcTypeLowerRangeLoad = 1;
        }else if(this.calcTypeLowerRangeLoad.equals( "2_下限COP値固定" )){
           this.num_calcTypeLowerRangeLoad = 2;
        }else if(this.calcTypeLowerRangeLoad.equals( "3_下限入力値と中間切片" )){
           this.num_calcTypeLowerRangeLoad = 3;
        }else{
           this.num_calcTypeLowerRangeLoad = 0;
        }

        if(out_Qc_S<0.001)      fmc= 0.0;
           else           fmc= out_Qc_C/out_Qc_S;
        if(out_PPEc_S<0.001) pmc= 0.0;
           else           pmc= out_PPEc_C/out_PPEc_S;

        if(out_Qh_S<0.001)      fmh= 0.0;
           else           fmh= out_Qh_C/out_Qh_S;
        if(out_PPEh_S<0.001) pmh= 0.0;
           else           pmh= out_PPEh_C/out_PPEh_S;

        if(pmc>0.9999) {pmc=0.0;fmc=0.0;}
        if(pmh>0.9999) {pmh=0.0;fmh=0.0;}


        //isAdjust2012 = "台数を調整する";//
        this.isAdjust2012 = spec.getSpecValue( this.SPEC_isAdjust2012, false );
        //自動調整する場合は０％まで運転とする。20200108
        if( this.isAdjust2012 ) {
           this.in_run_stop = 0;
        }

        //仮設調整
```

```
    this.fRate_watInHSc = this.de_fRate_watInHS;
    this.fRate_watInHSh = this.de_fRate_watInHS;

    // 調整の計算ステップ数
    this.numAdjustSteps = spec.getSpecValue( this.SPEC_NumAdjustSteps, 12 );

    this.rcList = new LinkedList<Double>();
    this.rhList = new LinkedList<Double>();
    for( int i=0; i<this.numAdjustSteps; i++ ){
       this.rcList.add( 0. );
       this.rhList.add( 0. );
    }

    this.out_Qc_S_1   = this.out_Qc_S;//= "定格冷房能力";//kW
    this.out_Qc_C_1   = this.out_Qc_C;//= "中間冷房能力";//kW
    this.out_PPEc_S_1 = this.out_PPEc_S;//= "定格冷房入力(電力)";//kW
    this.out_PPEc_C_1 = this.out_PPEc_C;//= "中間冷房入力(電力)";//kW
    this.out_Qh_S_1   = this.out_Qh_S;//= "定格暖房能力";//kW
    this.out_Qh_C_1   = this.out_Qh_C;//= "中間暖房能力";//kW
    this.out_Qh_L_1   = this.out_Qh_L;//= "低温暖房能力";//kW
    this.out_PPEh_S_1 = this.out_PPEh_S;//= "定格暖房入力(電力)";//kW
    this.out_PPEh_C_1 = this.out_PPEh_C;//= "中間暖房入力(電力)";//kW
    this.out_PPEh_L_1 = this.out_PPEh_L;//= "低温暖房入力(電力)";//kW

}


@Override
public void initialize(IBestStateMessage stateNodes,
      IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
      IBestDomainSchedule schedule) {
    //System.out.println(this.moduleName + " BuillMultiOutイニシャライズ");
    //状態ノードを受け取る
    super.sm = stateNodes;
    //制御ノードを受け取る
    super.cm = commandNodes;
    //記録ノードを受け取る
    super.rm = recordNodes;

    //eleIn
    this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

    //watInHS
    this.watInHS = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInHS );

    //watOutHS
    this.watOutHS = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutHS );

    //private final String S_NODE_valInLine  = "LO_valInLine";  //単線
    if( super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine )) != null ){

       //System.out.println( " BM_out S_NODE_valInLine登録済み" );

       this.rmlinemap
       =  (Map<String, BM_EHPdata>) super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine));

       if( this.rmlinemap.size() == 0 ){
          //
          //System.out.println( "rmlinemap のサイズが０ ");
          this.RMdata = new BM_EHPdata();
          this.RMdata.set_path( this.path );
          this.RMdata.set_kiki( this.kiki_file );
          this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
          this.RMdata.setWaterCooled(this.isWaterCooled );
          this.RMdata.setwatInHS( this.watInHS );//20120823nino
          this.rmlinemap.put( "Parent", this.RMdata );

          super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );

       }else{
          this.num = 0;
          Set<String> LineEntry = this.rmlinemap.keySet();
```

```
            for(Iterator<String> i = LineEntry.iterator();i.hasNext();){
               String key = (String) i.next();
               //          System.out.println(" name="+name+" key="+key);
               this.RMdata = this.rmlinemap.get(key);
               this.rmnm[this.num]=key;
               //          System.out.println("key="+key+" 0="+rmdata[0]+" 1="+rmdata[1]+" 2="+rmdata[2]+"
3="+rmdata[3]+" 4="+rmdata[4]+" 5="+rmdata[5]+" 6="+rmdata[6]);
               this.num++;
            }
            //
            //20150409
            double sum_in_Qc_S = 0;
            double sum_in_Qh_S = 0;
            double sum_in_dai = 0;
            //出力設定
            for(int i=0; i<this.num; i++){
               //System.out.println( "kiki_file Name = " +kiki_file );
               this.RMdata = this.rmlinemap.get( this.rmnm[i] );
               this.RMdata.set_kiki(kiki_file);
               this.RMdata.set_path(this.path);
               this.RMdata.setCOOLandHEAT( this.isCOOLandHEATout );
               this.RMdata.setWaterCooled(this.isWaterCooled );
               this.RMdata.setwatInHS( this.watInHS );//20120823nino
               rmlinemap.put(rmnm[i],this.RMdata);
               //20150409
               sum_in_Qc_S += this.RMdata.get_in_Qc_S();
               sum_in_Qh_S += this.RMdata.get_in_Qh_S();
               sum_in_dai += this.RMdata.get_in_dai();
            }
            //
            //System.out.println( "rmlinemap のサイズが "+this.num );
            this.RMdata = new BM_EHPdata();
            this.RMdata.set_path( this.path );
            this.RMdata.set_kiki( this.kiki_file );
            this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
            this.RMdata.setWaterCooled(this.isWaterCooled );
            this.RMdata.setwatInHS( this.watInHS );//20120823nino
            //20150409
            this.RMdata.set_in_Qc_S( sum_in_Qc_S );
            this.RMdata.set_in_Qh_S( sum_in_Qh_S );
            this.RMdata.set_in_dai( sum_in_dai );

            this.rmlinemap.put( "Parent", this.RMdata );

            //20150409
      /*    BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
            + "接続定格冷却容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qc_S + "[W](
                  +AirFormat.df_3( sum_in_Qc_S / this.out_Qc_S ) + "倍)" );
            BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
            + "接続定格加熱容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qh_S + "[W](
                  +AirFormat.df_3( sum_in_Qh_S / this.out_Qh_S ) + "倍)" );
            BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
            + "接続台数="+AirFormat.df_0( sum_in_dai )+"[台]" );
      */
            //接続容量、台数チェック
            this.RMdata.checkConnect( this.out_Qc_S, this.out_Qh_S, sum_in_Qc_S, sum_in_Qh_S, sum_in_dai,
                  this.moduleName, "initialize", this.name, this.isRecord, this.message );

            super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );
            this.dbflg = 1;
         }
      }else{

         //System.out.println( " BM_out S_NODE_valInLine未登録" );

         //System.out.println( this.moduleName + ">>Warning<< rmlinemap is null !!" );
         message.append( "(W)単線の接続なし→作成");
         this.rmlinemap = new HashMap<String,BM_EHPdata>();
         //
         this.RMdata = new BM_EHPdata();
         this.RMdata.set_path( this.path );
         this.RMdata.set_kiki( this.kiki_file );
```

```
            this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
            this.RMdata.setWaterCooled(this.isWaterCooled );
            this.RMdata.setwatInHS( this.watInHS );//20120823nino
            this.rmlinemap.put( "Parent", this.RMdata );

            super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );
        }

        //グラフ表示の準備
        if( this.isGVisible ){
            gBMout = new GraphJFrameBuilMultiOut_S20101212( this.name, 300,  out_Qc_S * 1.5, 0 );
        }



        filenames[0]=kiki_file;
        equipmentName=kiki_file;
        pacDB=new PACDBManager(path,filenames);
        pacDB.setEquipmentName(equipmentName);

    }




    /**
     * ビルマルチ計算及び結果出力/各室内機よりも先に動きます。
     */
    @Override
    public void outputs() {

        if (null == super.cm || null == super.sm )
            return;

        if( this.dbflg == 0 ){

            //System.out.println( "    BM_out  outputsで  室側データの設定" );

            //System.out.println( " Parent path="+ this.rmlinemap.get( "Parent" ).get_path() );

            if( super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine )) != null ){
                this.rmlinemap
                =  (Map<String, BM_EHPdata>) super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine));
                //
                this.num = 0;
                Set<String> LineEntry = this.rmlinemap.keySet();
                for(Iterator<String> i = LineEntry.iterator();i.hasNext();){
                    String key = (String) i.next();
                    //          System.out.println(" name="+name+" key="+key);
                    this.RMdata = this.rmlinemap.get(key);
                    this.rmnm[this.num]=key;
                    //          System.out.println("key="+key+" 0="+rmdata[0]+" 1="+rmdata[1]+" 2="+rmdata[2]+"
3="+rmdata[3]+" 4="+rmdata[4]+" 5="+rmdata[5]+" 6="+rmdata[6]);
                    this.num++;
                }
                //20150409
                double sum_in_Qc_S = 0;
                double sum_in_Qh_S = 0;
                double sum_in_dai = 0;
                //出力設定
                for(int i=0; i<this.num; i++){
                    //System.out.println( "kiki_file Name = " +kiki_file );
                    this.RMdata = this.rmlinemap.get( this.rmnm[i] );
                    this.RMdata.set_kiki(kiki_file);
                    this.RMdata.set_path(this.path);
                    this.RMdata.setCOOLandHEAT( this.isCOOLandHEATout );
                    this.RMdata.setWaterCooled(this.isWaterCooled );
                    this.RMdata.setwatInHS( this.watInHS );//20120823nino
                    rmlinemap.put(rmnm[i],this.RMdata);
                    //20150409
                    sum_in_Qc_S += this.RMdata.get_in_Qc_S();
```

```
            sum_in_Qh_S += this.RMdata.get_in_Qh_S();
            sum_in_dai += this.RMdata.get_in_dai();
        }
        //
        //20150409
        this.RMdata = this.rmlinemap.get( "Parent" );
        //20150409
        this.RMdata.set_in_Qc_S( sum_in_Qc_S );
        this.RMdata.set_in_Qh_S( sum_in_Qh_S );
        this.RMdata.set_in_dai( sum_in_dai );

        this.rmlinemap.put( "Parent", this.RMdata );

        //20150409
/*      BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
        + "接続定格冷却容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qc_S + "[W]("
              +AirFormat.df_3( sum_in_Qc_S / this.out_Qc_S ) + "倍)" );
        BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
        + "接続定格加熱容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qh_S + "[W]("
              +AirFormat.df_3( sum_in_Qh_S / this.out_Qh_S ) + "倍)" );
        BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
        + "接続台数="+AirFormat.df_0( sum_in_dai )+"[台]" );
*/      //
        //接続容量、台数チェック
        this.RMdata.checkConnect( this.out_Qc_S, this.out_Qh_S, sum_in_Qc_S, sum_in_Qh_S, sum_in_dai,
              this.moduleName, "outputs", this.name, this.isRecord, this.message );

    }
    this.dbflg = 1;

}

//message.setLength(0);
//ノード接続毎回読込み
//this.airInOA = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airInOA));
this.watInHS = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watInHS));
this.swcIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcIn) );
this.modIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modIn));

//停止中も熱源水は入口状態で出るものとする。
this.watOutHS.copyAllState( this.watInHS );//20130110


//初期化
this.Tc_Loadout = 0;
this.Th_Loadout = 0;

//mState=this.swc*this.mod;
//
if( Airswc.isOFF( this.swcIn ) ){
    //停止
    this.mState = 0;
}else if( this.watInHS.getFlowRate() == 0 && !this.isAdjust2012 ){
    //停止    //20110622nino
    this.mState = 0;
    this.message.append( "(W)熱源水流量=0→停止" );
}else if( Airswc.isON( this.swcIn )){
    //運転
    if( Airmod.isCOOL( this.modIn ) && Airmod.isHEAT( this.modIn ) ){
        //冷暖同時
        if( this.isCOOLandHEATout ){
            this.mState = 11;
        }else{
            //冷暖同時機種で無い場合は冷房運転とする
            this.mState = 1;
            this.message.append( "(W)冷暖同時機種ではないので冷房運転で計算" );
        }
    }else if( Airmod.isCOOL( this.modIn )){
        //冷房運転
        this.mState = 1;
    }else if( Airmod.isHEAT( this.modIn )){
        //暖房運転
```

```
            this.mState = 2;
        }else if( Airmod.isVENTILATE( this.modIn )){
            //換気モード
            this.mState = 3;
        }else{
            //停止
            this.mState = 0;
        }
    }


    /**
     * 入力
     */
    //外気乾球温度設定
    //this.DBoa = this.airInOA.getTempDB();

    //外気絶対湿度を設定
    //this.XGoa = this.airInOA.getHumi();

    //外気湿球温度を設定
    //this.WBoa = Psychrometrics.FNWbtx(this.DBoa, this.XGoa);
    //if( this.WBoa < -9000.0 ){
    // this.WBoa = this.WB_cal(this.DBoa, this.XGoa);
    //}

    //熱源水入口温度　流量
    this.t_watInHS = this.watInHS.getTemp();
    this.fRate_watInHS = this.watInHS.getFlowRate();

    this.rmlinemap
    = (Map<String,BM_EHPdata>) super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine ));

    this.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
    this.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
    this.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
    this.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
    this.DBra_max = -100.0;
    this.WBra_min = 1000.0;
    this.Lratemin = 100.0;

    double dmy1 = 100.0;

    //室内機データの読込
    for( int i=0; i < this.num; i++ ){
        this.RMdata = this.rmlinemap.get( this.rmnm[i] );
        dmy1 = this.RMdata.get_L_rate();
        if( this.Lratemin > dmy1 ){
            this.Lratemin=dmy1;
        }
        dmy1 = this.RMdata.get_R_wb();
        if( this.WBra_min > dmy1 && this.RMdata.get_R_Tc_load()>0 ){
            this.WBra_min=dmy1;
        }
        dmy1 = this.RMdata.get_R_db();
        if( this.DBra_max < dmy1 && this.RMdata.get_R_Th_load()>0  ){
            this.DBra_max=dmy1;
        }
        this.allSc_heat += this.RMdata.get_R_Sc_load();//室内機処理熱量（冷却顕熱）[W]
        this.allTc_heat += this.RMdata.get_R_Tc_load();//室内機処理熱量（冷却全熱）[W]
        //
        this.allSh_heat += this.RMdata.get_R_Sh_load();//室内機処理熱量（加熱顕熱）[W]
        this.allTh_heat += this.RMdata.get_R_Th_load();//室内機処理熱量（加熱全熱）[W]

        //System.out.println( "i="+i+"  this.RMdata.get_R_Tc_load()" + this.RMdata.get_R_Tc_load() +
"this.RMdata.get_R_Th_load()" + this.RMdata.get_R_Th_load());
    }

    this.WBra = this.WBra_min;
    this.DBra = this.DBra_max;
```

```
      //*************************----------------
      if( this.isAdjust2012 ){
         double rc;
         double c_OutCapa;
         double rh;
         double h_OutCapa;


         //処理可能熱量　仮計算
         if( this.allTc_heat==0 && this.allTh_heat==0 ){
            //冷却・加熱要求なし
            c_OutCapa = this.out_Qc_S_1;
            h_OutCapa = this.out_Qh_S_1;
         }else if( this.allTh_heat == 0 ){
            //冷却要求のみ coolingOnly_cal()
            this.fRate_watInHS = this.fRate_watInHSc;//固定流量で調整する
            c_OutCapa = this.pacDB.getKctw( this.t_watInHS ) * this.pacDB.getKcf( this.fRate_watInHS /
this.fRate_watInHSc ) * this.pacDB.getKcti( this.WBra )* this.Lratemin * this.out_Qc_S_1;//仮計算　室外機処理可能熱量（冷
却全熱）[W]
            h_OutCapa = this.out_Qh_S_1;
         }else if( this.allTc_heat == 0 ){
            //加熱要求のみ heatingOnly_cal()
            this.fRate_watInHS = this.fRate_watInHSh;//固定流量で調整する
            c_OutCapa = this.out_Qc_S_1;
            h_OutCapa = this.pacDB.getKhtw( this.t_watInHS ) * this.pacDB.getKhf( this.fRate_watInHS /
this.fRate_watInHSh ) * this.pacDB.getKhti( this.DBra )* this.Lratemin * this.out_Qh_S_1;//仮計算　室外機処理可能熱量（加
熱全熱）[W]
         }else if( this.allTc_heat >= this.allTh_heat ){
            //冷却主体運転 cooling_and_heating_cal()
            this.fRate_watInHS = this.fRate_watInHSc;//固定流量で調整する
            c_OutCapa = this.pacDB.getKcmtw( this.t_watInHS ) * this.pacDB.getKcmf( this.fRate_watInHS /
this.fRate_watInHSc ) * this.pacDB.getKcti( this.WBra )* this.Lratemin * this.out_Qc_S_1;//仮計算　室外機処理可能熱量（冷
却全熱）[W]
            h_OutCapa = this.pacDB.getKhti( this.DBra ) * this.Lratemin * this.out_Qh_S_1;//室外機処理可能熱量（加熱全
熱）[W]
         }else{
            //加熱主体運転 heating_and_cooling_cal()
            this.fRate_watInHS = this.fRate_watInHSh;//固定流量で調整する
            c_OutCapa = this.pacDB.getKcti( this.WBra) * this.Lratemin * out_Qc_S_1;//室外機処理可能熱量（冷却全熱）[W]
            h_OutCapa = this.pacDB.getKhmtw( this.t_watInHS ) * this.pacDB.getKhmf( this.fRate_watInHS /
this.fRate_watInHSh ) * this.pacDB.getKhti( this.DBra )* this.Lratemin * this.out_Qh_S_1;//仮計算　室外機処理可能熱量（加
熱全熱）[W]
         }

         //冷却能力調整
         if( c_OutCapa == 0 ){
            //rc = this.allTc_heat / this.out_Qc_S;
            rc = this.allTc_heat / this.out_Qc_S_1;
         }else{
            rc = this.allTc_heat / c_OutCapa;
         }

         if( rc > 1.1 ){
         // rc = 1.1;
         }

         //加熱能力調整
         if( h_OutCapa == 0 ){
            //rh = this.allTh_heat / this.out_Qh_S;
            rh = this.allTh_heat / this.out_Qh_S_1;
         }else{
            rh = this.allTh_heat / h_OutCapa;
         }

         if( rh > 1.1 ){
         // rh = 1.1;
         }

         //*************************----------------
         if( true ){
            //移動平均の最大値で調整していく
            this.rcList.removeLast();
```

```
            this.rcList.addFirst( rc );
            //
            double sum_rc = 0;
            for( int i=0; i<this.numAdjustSteps; i++ ){
               sum_rc += this.rcList.get( i );
            }
            this.ave_rcAdjust = sum_rc / this.numAdjustSteps;
            //
            if( sum_rc > this.max_rcAdjust * this.numAdjustSteps ){
               this.max_rcAdjust = sum_rc / this.numAdjustSteps;

               this.out_rcAdjust = this.max_rcAdjust;
               //this.in_run_stop_Num = this.in_run_stop / this.out_rcAdjust;

               this.out_Qc_S   = this.out_Qc_S_1    * this.out_rcAdjust;
               this.out_Qc_C   = this.out_Qc_C_1    * this.out_rcAdjust;
               this.out_PPEc_S = this.out_PPEc_S_1  * this.out_rcAdjust;
               this.out_PPEc_C = this.out_PPEc_C_1  * this.out_rcAdjust;

               this.fRate_watInHSc = this.de_fRate_watInHS * this.out_rcAdjust;
            }
         }


         if( true ){
            //移動平均の最大値で調整していく
            this.rhList.removeLast();
            this.rhList.addFirst( rh );
            //
            double sum_rh = 0;
            for( int i=0; i<this.numAdjustSteps; i++ ){
               sum_rh += this.rhList.get( i );
            }
            this.ave_rhAdjust = sum_rh / this.numAdjustSteps;
            //
            if( sum_rh > this.max_rhAdjust * this.numAdjustSteps ){
               this.max_rhAdjust = sum_rh / this.numAdjustSteps;

               this.out_rhAdjust = this.max_rhAdjust;
               //this.in_run_stop_Num = this.in_run_stop / this.out_rcAdjust;

               this.out_Qh_S   = this.out_Qh_S_1   * this.out_rhAdjust;
               this.out_Qh_C   = this.out_Qh_C_1   * this.out_rhAdjust;
               this.out_Qh_L   = this.out_Qh_L_1   * this.out_rhAdjust;
               this.out_PPEh_S = this.out_PPEh_S_1 * this.out_rhAdjust;
               this.out_PPEh_C = this.out_PPEh_C_1 * this.out_rhAdjust;
               this.out_PPEh_L = this.out_PPEh_L_1 * this.out_rhAdjust;

               this.fRate_watInHSh = this.de_fRate_watInHS * this.out_rhAdjust;
            }
         }
         //*************************----------------


         //調整補正
/*       if( rc > 1.0 ){
            this.out_Qc_S *= rc;
            this.out_Qc_C *= rc;
            this.out_PPEc_S *= rc;
            this.out_PPEc_C *= rc;
            //
            this.fRate_airEAc *= rc;
         }
*/
         //調整補正
/*       if( rh > 1.0 ){
            this.out_Qh_S *= rh;
            this.out_Qh_C *= rh;
            this.out_Qh_L *= rh;
            this.out_PPEh_S *= rh;
            this.out_PPEh_C *= rh;
            this.out_PPEh_L *= rh;
```

```
            //
            this.fRate_airEAh *= rh;
        }
*/
        this.out_Qc_Adjust = this.out_Qc_S;
        this.out_Qh_Adjust = this.out_Qh_S;

        //System.out.println( "BM out  out_Qc_Adjust=" +out_Qc_Adjust+"  out_Qh_Adjust="+out_Qh_Adjust);
    }


    //*************************----------------


    switch( this.mState ){
    case 3://20120313nino
        //換気
        //室外機は停止とする
    case 0:
        //停止
        this.message.append("(C)停止");
        this.calc_Stop();
        break;
    case 1:
        //冷房
        if( this.out_Qc_S_1 > 0 ){
            this.message.append("(C)冷房運転");
            this.coolingOnly_cal();
        }else{
            this.message.append("(C)冷能力=0停止");//20130625
            this.calc_Stop();
        }

        break;
    case 2:
        //暖房
        if( this.out_Qh_S_1 > 0 ){
            this.message.append("(C)暖房運転");
            this.heatingOnly_cal();
        }else{
            this.message.append("(C)暖能力=0停止");//20130625
            this.calc_Stop();
        }

        break;
    case 11:
        //冷暖同時
        this.message.append("(C)冷暖房運転");
        if( this.allTh_heat == 0 && this.out_Qc_S_1 > 0 ){
            //冷房運転
            this.coolingOnly_cal();

        }else if( this.allTc_heat == 0 && this.out_Qh_S_1 > 0){
            //暖房運転
            this.heatingOnly_cal();

        }else if( this.allTc_heat > this.allTh_heat && this.out_Qc_S_1 > 0){
            //冷房主体運転
            //20181017
            if( this.is2018SinseiCHFree ) {
                this.coolingOnly_cal();
            }else {
                this.cooling_and_heating_cal();
            }

        }else if( this.allTh_heat > this.allTc_heat && this.out_Qh_S_1 > 0){
            //暖房主体運転
            //20181017
            if( this.is2018SinseiCHFree ) {
                this.heatingOnly_cal();
            }else {
```

```
            this.heating_and_cooling_cal();
         }
      }else{
         this.message.append("(C)能力=0停止");//20130625
         this.calc_Stop();
      }

      break;
   default:
      this.message.append("(C)運転モード？停止");//20130625
      this.calc_Stop();
      //System.out.println(this.moduleName + ">>Error<< onOff*modeが範囲外");
   }

   this.E_PPE = this.PPE * Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 );
   this.eleIn.setActivePower( this.PPE );
   this.eleIn.setReactivePower( this.E_PPE );
   this.eleIn.setPhase( this.phase );
   this.eleIn.setVoltage( this.voltage );
   this.eleIn.setFrequency( this.frequency );

   //出力設定
   for(int i=0;i<num;i++){
      this.RMdata = this.rmlinemap.get( this.rmnm[i] );
      this.RMdata.set_CAPrateC( this.CAPrateC );
      this.RMdata.set_CAPrateH( this.CAPrateH );
      this.rmlinemap.put( this.rmnm[i], this.RMdata );
   }
   super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );

   super.sm.setState( super.getConnectionNode( this.S_NODE_watOutHS ), this.watOutHS );

   this.t_watOutHS = this.watOutHS.getTemp();

   //グラフデータ追加
   if( this.isGVisible ){
      this.gBMout.addData( BestTimeManager.getDateWeatherTime(),
      this.Sc_Loadout,//室外機処理可能熱量（冷却）[W]
      this.Sh_Loadout,//室外機処理可能熱量（加熱）[W]
      this.allTc_heat,//室内機処理要求熱量（冷却）[W]
      this.allTh_heat,//室内機処理要求熱量（加熱）[W]
      this.PPE,
      this.t_watInHS,
      this.t_watOutHS,
      this.fRate_watInHS,
      this.XGra,
      this.CAPrateC.doubleValue() );
   }

   //記録ノード
   if( this.isRecord && super.rm != null ){
      this.record();
   }

   this.message.setLength(0);
}

private void calc_Stop(){
   this.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
   this.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
   this.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
   this.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
   this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
   this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
   this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
   this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
   this.PPE = 0.0;
   this.CAPrateC = 0.0;
   this.CAPrateH = 0.0;

}
```

```
    /**
     * 記録
     */
    private void record(){
        //記録ノード
        if( super.rm != null ){
        if( CheckPrintModule.isPrintMessage ){
            //message
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_message, this.name, message.toString() );
        }

        if( CheckPrintModule.isPrintEnergy ){
            //記録ノードに室外機消費電力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID4, this.name, AirFormat.df_2(this.PPE));
        }

        if( CheckPrintModule.isPrintLoad ){
            //記録ノードに室外機実処理熱量（全熱）を設定
            if( this.Tc_Loadout > 0 ){
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT, this.name, AirFormat.df_2(-
this.Tc_Loadout));//室外機冷却実処理熱量/現状[W]
            }else{
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT, this.name,
AirFormat.df_2(this.Th_Loadout));//室外機加熱実処理熱量/現状[W]
            }
        }

        if( CheckPrintModule.isPrintStateOut ){
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_t_watOutHS, this.name,
AirFormat.df_2(this.t_watOutHS));//室外機熱源水出口温度#℃#温度
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_fRate_watOutHS, this.name,
AirFormat.df_2(this.watOutHS.getFlowRate()));//室外機熱源水出口質量流量#g/s#質量流量
        }

        if( CheckPrintModule.isPrintStateMy ){
            //記録ノードに室内機加熱能力補正比率を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_CAPrateC, this.name,
AirFormat.df_2(this.CAPrateC));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_CAPrateH, this.name,
AirFormat.df_2(this.CAPrateH));
            //記録ノードに室内機要求熱量（顕熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_allSc_heat, this.name,
AirFormat.df_2(this.allSc_heat));//室内機冷却要求顕熱量合計[W]
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_allSh_heat, this.name,
AirFormat.df_2(this.allSh_heat));//室内機加熱要求顕熱量合計[W]
            //記録ノードに室内機要求熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_allTc_heat, this.name,
AirFormat.df_2(this.allTc_heat));//室内機冷却要求全熱量合計[W]
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_allTh_heat, this.name,
AirFormat.df_2(this.allTh_heat));//室内機処理要求全熱量合計[W]
            //記録ノードに室外機処理可能熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_Sc_Loadout, this.name,
AirFormat.df_2(this.Sc_Loadout));//室外機冷却可能熱量/最大[W]
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_Sh_Loadout, this.name,
AirFormat.df_2(this.Sh_Loadout));//室外機加熱可能熱量/最大[W]
            //記録ノードに室外機実処理熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_Tc_Loadout, this.name,
AirFormat.df_2(this.Tc_Loadout));//室外機冷却実処理熱量/現状[W]
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_Th_Loadout, this.name,
AirFormat.df_2(this.Th_Loadout));//室外機加熱実処理熱量/現状[W]
            //
        }

        if( CheckPrintModule.isPrintStateIn ){
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_t_watInHS, this.name,
AirFormat.df_2(this.t_watInHS));//室外機熱源水入口温度#℃#温度
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_fRate_watInHS, this.name,
AirFormat.df_2(this.fRate_watInHS));//室外機熱源水入口質量流量#g/s#質量流量
        }

        if( CheckPrintModule.isPrintAdjust ){
```

```
            if( this.isAdjust2012 ){
               //記録ノードに室外機調整能力を設定
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qc_Adjust, this.name,
this.out_Qc_Adjust );//室外機調整能力[W]
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qh_Adjust, this.name,
this.out_Qh_Adjust );//室外機調整能力[W]
               //記録ノードに室外機調整能力を設定
               //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rcAdjust", this.name, this.out_rcAdjust );//
室外機調整率[-]
               //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rhAdjust", this.name, this.out_rhAdjust );//
室外機調整率[-]
            }
         }

         }
      }

      /**
       * ビルマルチ計算/各室内機の集計後
       */

      @Override
      public void update() {
         double rCode0;
         double rCode1;

         //能力補正係数のうち室外機側分を求める　次のステップで室内機側の能力設定に使用する
         //CoolingOnly
         rCode0 = this.pacDB.getKctw( this.t_watInHS );//水温補正
         rCode1 = this.pacDB.getKcf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
         for( int i=0; i<this.num; i++ ){//20110623nino
            this.RMdata = this.rmlinemap.get( this.rmnm[i] );
            this.RMdata.set_out_c_CAPrevision( rCode0 * rCode1 );
         }

         //Cooling and heating
         //rCode0 = this.pacDB.getKcmtw( this.t_watInHS );//水温補正
         //rCode1 = this.pacDB.getKcmf( this.fRate_watInHS / this.de_fRate_watInHS );//水量補正

         //Heating_only
         rCode0 = this.pacDB.getKhtw( this.t_watInHS );//水温補正
         rCode1 = this.pacDB.getKhf( this.fRate_watInHS / this.fRate_watInHSh );//水量補正
         for( int i=0; i<this.num; i++ ){//20110623nino
            this.RMdata = this.rmlinemap.get( this.rmnm[i] );
            this.RMdata.set_out_h_CAPrevision( rCode0 * rCode1 );
         }

         //Heating and cooling
          //rCode0 = this.pacDB.getKhmtw( this.t_watInHS );//水温補正
          //rCode1 = this.pacDB.getKhmf( this.de_fRate_watInHS / this.de_fRate_watInHS );//水量補正

      }

      public Object viewInternal(TestCommand cmd) {

         ArrayList<Object> result = new ArrayList<Object>();
         result.add(super.sm);
         result.add(super.cm);
         result.add(super.rm);
         //外部定義
         result.add(this.name);

         result.add(this.out_Qc_S);
         result.add(this.out_Qc_C);
         result.add(this.out_PPEc_S);
         result.add(this.out_PPEc_C);
         result.add(this.out_Qh_S);
         result.add(this.out_Qh_C);
         result.add(this.out_Qh_L);
         result.add(this.out_PPEh_S);
         result.add(this.out_PPEh_C);
         result.add(this.out_PPEh_L);
```

```
        result.add(this.Lp);
        result.add(this.Lh);

        //result.add(this.DBoa);
        //result.add(this.XGoa);
        result.add(this.DBra);
        result.add(this.XGra);

        return result;
    }


    private void coolingOnly_cal(){
        double rCode0,rCode1,rCode2,rCode3,rCode4;
        double D_fmc,D_pmc,alph,beta;
        double Rp;

        D_fmc    = this.fmc;
        D_pmc    = this.pmc;

        //上下限のチェック 20120821nino
        double twatIn = this.t_watInHS;
        double wbRA = this.WBra;

        if( twatIn > this.pacDB.getformulaRangeMax( this.pacDB.getFACKctw() ) ){
            //熱源水入口温度＞上限の時停止
            this.message.append( "(C)熱源水入口温度"+twatIn+"＞"+this.pacDB.getformulaRangeMax( this.pacDB.getFACKctw() )+"
上限→停止");
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPE = 0.0;
            //this.airOutEA.setFlowRate( 0. );
            //this.airOutEA.setTempDB( this.airInOA.getTempDB() );
            //this.watOutHS.setFlowRate( 0.);//20130110
            //this.watOutHS.setTemp( this.watInHS.getTemp() );//20130110
            this.watOutHS.copyAllState( this.watInHS );//20130110
            return;
        }
        if( twatIn < this.pacDB.getformulaRangeMin( this.pacDB.getFACKctw() ) ){
            //熱源水入口温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)熱源水入口温度"+twatIn+"＜"+this.pacDB.getformulaRangeMin( this.pacDB.getFACKctw() )+"
下限→下限値で特性計算");
            twatIn = this.pacDB.getformulaRangeMin( this.pacDB.getFACKctw() );
        }
        if( wbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() ) ){
            //室内湿球温度＞上限の時　上限値の特性で運転
            this.message.append( "(C)室WB＞上限→上限値で特性計算");
            wbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
        }
        if( wbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() ) ){
            //室内湿球温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)室WB＜下限→下限値で特性計算");
            wbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
        }


        //能力
        rCode0 = this.pacDB.getKctw( twatIn );//水温補正
        rCode1 = this.pacDB.getKcf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
        rCode2 = this.pacDB.getKcti( wbRA );//室温補正
        rCode3 = this.Lratemin;//配管補正

        this.Sc_Loadout = rCode0 * rCode1 * rCode2 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

        // System.out.println("Sc_Loadout="+Sc_Loadout+" Lratemin"+Lratemin+" / allTc_heat="+this.allTc_heat );

        this.Rc = this.allTc_heat/this.Sc_Loadout;
```

```
this.CAPrateC = 1.0;
this.CAPrateH = 0.0;
this.Tc_Loadout = this.allTc_heat;//室外機実処理熱量（冷却全熱）[W]//20110622nino
if( this.Rc > 1.0 ) {
   this.CAPrateC = 1.0/ this.Rc;
   this.Rc = 1.0;
   this.Tc_Loadout = this.Sc_Loadout;//室外機実処理熱量（冷却全熱）[W]//20110622nino
}

//低負荷領域の計算仕分け:Rh
if( this.Rc < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = this.Rc;
}


alph = this.pacDB.getAlphac( Rp );

// System.out.println("Rc="+Rc+" "+D_fmc+" "+D_pmc);

beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc );

if( D_pmc * D_fmc < 0.001 ){
   beta = 1.0;
}

//入力
rCode0 = alph * beta;
rCode1 = this.pacDB.getKchpid( Rp );//代表入力補正
rCode2 = this.pacDB.getKcwtw( twatIn );//水温補正
rCode3 = this.pacDB.getKcwf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正

//       System.out.println("WBra="+WBra);

rCode4 = this.pacDB.getKcwti( wbRA );//室温補正

this.PPE = rCode0 *rCode1 * rCode2 * rCode3 * rCode4 * this.out_PPEc_S;
//低負荷領域の計算仕分け:PPE
if( this.Rc < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.PPE = 0.0;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      this.PPE *= ( this.Rc / Rp );
      break;

   case 3://  3_下限入力値と中間切片
      this.PPE *= ( 0.5 + this.Rc / Rp / 2. );
      break;

   default:
   }
```

```
    }else{
       //何もしない
    }


    //airOutEA
    //this.airOutEA.setFlowRate( this.fRate_airEA );
    //this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPE ) / this.fRate_airEA * 1000. + this.airInOA.getTempDB() );
    //watOutHS
    this.watOutHS.setFlowRate( this.fRate_watInHS );
    this.watOutHS.setTemp( ( this.Tc_Loadout + this.PPE ) / this.fRate_watInHS / 4.18605 + this.t_watInHS );

    if( this.Rc < 0.0001){
       this.CAPrateC = 0.0;
       this.CAPrateH = 0.0;
       this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
       this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
       this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
       this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
       this.PPE = 0.0;
       this.watOutHS.copyAllState( this.watInHS );//20130110
    }else if( this.Rc < this.in_run_stop ){
       switch( this.num_calcTypeLowerRangeLoad){
       case 0://  0_発停運転
          this.CAPrateC = 0.0;
          this.CAPrateH = 0.0;
          this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
          this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
          this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
          this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
          this.PPE = 0.0;
          this.watOutHS.copyAllState( this.watInHS );//20130110
          break;

       case 1://  1_下限入力値固定
          break;

       case 2://  2_下限COP値固定
          break;

       case 3://  3_下限入力値と中間切片
          break;

       default:
       }
    }



    //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

 }


 private void cooling_and_heating_cal(){
    double rCode0,rCode1,rCode2,rCode3,rCode4;
    double D_fmc,D_pmc,alph,beta;
    double Rp;

    D_fmc    = this.fmc;
    D_pmc    = this.pmc;

    //上下限のチェック 20120821nino
    double twatIn = this.t_watInHS;
    double wbRA = this.WBra;

    if( twatIn > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcmtw() ) ){
       //熱源水入口温度＞上限の時停止
       this.message.append( "(C)熱源水入口温度"+twatIn+"＞"+this.pacDB.getformulaRangeMax( this.pacDB.getFACKcmtw() )+"
上限→停止");
       this.CAPrateC = 0.0;
       this.CAPrateH = 0.0;
```

```
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPE = 0.0;
            this.watOutHS.copyAllState( this.watInHS );//20130110
            return;
        }
        if( twatIn < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcmtw() ) ){
            //熱源水入口温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)熱源水入口温度"+twatIn+"＜"+ this.pacDB.getformulaRangeMin( this.pacDB.getFACKcmtw())+"
下限→下限値で特性計算");
            twatIn = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcmtw() );
        }
        if( wbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() ) ){
            //室内湿球温度＞上限の時　上限値の特性で運転
            this.message.append( "(C)室WB＞上限→上限値で特性計算");
            wbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
        }
        if( wbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() ) ){
            //室内湿球温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)室WB＜下限→下限値で特性計算");
            wbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
        }


        //冷房側　処理熱量
        rCode0 = this.pacDB.getKcmtw( twatIn );//水温補正
        rCode1 = this.pacDB.getKcmf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
        rCode2 = this.pacDB.getKcti( wbRA );//室温補正
        rCode3 = this.Lratemin;//配管特性

        this.Sc_Loadout = rCode0 * rCode1 * rCode2 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

        //容量比
        this.Rc = this.allTc_heat / this.Sc_Loadout;

        this.CAPrateC = 1.0;
        this.Tc_Loadout = this.allTc_heat;//室外機実処理熱量（冷却全熱）[W]//20110622nino
        if( this.Rc > 1.0 ) {
            this.CAPrateC = 1.0/ this.Rc;
            this.Rc = 1.0;
            this.Tc_Loadout = this.Sc_Loadout;//室外機実処理熱量（冷却全熱）[W]//20110622nino
        }

        //低負荷領域の計算仕分け:Rh
        if( this.Rc < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                Rp = 0;
                break;

            case 1://  1_下限入力値固定
            case 2://  2_下限COP値固定
            case 3://  3_下限入力値と中間切片
                Rp = this.in_run_stop;
                break;

            default:
                Rp = 0;
            }
        }else{
            Rp = this.Rc;
        }


        //消費電力
        alph = this.pacDB.getAlphac( Rp );

        //        System.out.println("Rc="+Rc+" "+D_fmc+" "+D_pmc);
        beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc );
```

```
if( D_pmc * D_fmc < 0.001 ){
   beta = 1.0;
}

//入力
rCode0 = alph * beta;
rCode1 = this.pacDB.getKchpid( Rp );
rCode2 = this.pacDB.getKcmwtw( twatIn );//水温補正
rCode3 = this.pacDB.getKcmwf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正

//       System.out.println("WBra="+WBra);

rCode4 = this.pacDB.getKcwti( wbRA );//室温補正

this.PPE = rCode0 *rCode1 * rCode2 * rCode3 * rCode4 * this.out_PPEc_S;
//低負荷領域の計算仕分け:PPE
if( this.Rc < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.PPE = 0.0;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      this.PPE *= ( this.Rc / Rp );
      break;

   case 3://  3_下限入力値と中間切片
      this.PPE *= ( 0.5 + this.Rc / Rp / 2. );
      break;

   default:
   }
}else{
   //何もしない
}

//暖房側  処理熱量
rCode2 = this.pacDB.getKhti( this.DBra );//室温補正
rCode3 = this.Lratemin;//配管補正

this.Sh_Loadout = rCode2 * rCode3 * this.out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

//容量比
this.Rh = this.allTh_heat / this.Sh_Loadout;

this.CAPrateH = 1.0;
this.Th_Loadout = this.allTh_heat;//室外機実処理熱量（加熱全熱）[W]  //20110622nino
if( this.Rh > 1.0 ) {
   this.CAPrateH = 1.0/ this.Rh;
   this.Rh = 1.0;
   this.Th_Loadout = this.Sh_Loadout;//室外機実処理熱量（加熱全熱）[W]
}

//airOutEA
//this.airOutEA.setFlowRate( this.fRate_airEA );
//this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPE ) / this.fRate_airEA * 1000. + this.airInOA.getTempDB() );
//watOutHS
this.watOutHS.setFlowRate( this.fRate_watInHS );
this.watOutHS.setTemp( ( this.Tc_Loadout + this.PPE ) / this.fRate_watInHS / 4.18605 + this.t_watInHS );


if( this.Rc < 0.0001 ){
   this.CAPrateC = 0.0;
   this.CAPrateH = 0.0;
   this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
   this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
   this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
   this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
```

```
            this.PPE = 0.0;
            this.watOutHS.copyAllState( this.watInHS );//20130110

        }else if( this.Rc < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                this.CAPrateC = 0.0;
                this.CAPrateH = 0.0;
                this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
                this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
                this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
                this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
                this.PPE = 0.0;
                this.watOutHS.copyAllState( this.watInHS );//20130110
                break;

            case 1://  1_下限入力値固定
                break;

            case 2://  2_下限COP値固定
                break;

            case 3://  3_下限入力値と中間切片
                break;

            default:
            }
        }

        //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

    }


    private void heatingOnly_cal(){
        double rCode0,rCode1,rCode2,rCode3, rCode4;
        double D_fmh,D_pmh,D_out_Qh_S,D_out_PPEh_S,alph,beta;
        double Rp;

        D_fmh    = this.fmh;
        D_pmh    = this.pmh;

        D_out_Qh_S  = this.out_Qh_S;
        D_out_PPEh_S= this.out_PPEh_S;

        //if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L > 0.0001) { //低温入力の場合
        //    D_fmh    = this.fmh;
        //    D_pmh    = this.pmh;
        //    D_out_Qh_S  = this.out_Qh_L;
        //    D_out_PPEh_S= this.out_PPEh_L;
        //}


        //上下限のチェック 20120821nino
        double twatIn = this.t_watInHS;
        double dbRA = this.DBra;

        if( twatIn > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhtw() ) ){
            //熱源水入口温度＞上限の時　上限値の特性で運転
            this.message.append( "(C)熱源水入口温度"+twatIn+"＞"+this.pacDB.getformulaRangeMax( this.pacDB.getFACKhtw() )+"
上限→上限値で特性計算");
            twatIn = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhtw() );
        }
        if( twatIn < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhtw() ) ){
            //熱源水入口温度＜下限の時　停止
            this.message.append( "(C)熱源水入口温度"+twatIn+"＜"+this.pacDB.getformulaRangeMin( this.pacDB.getFACKhtw() )+"
下限→停止");
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
```

```
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPE = 0.0;
            this.watOutHS.copyAllState( this.watInHS );//20130110
            return;
        }
        if( dbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() ) ){
            //室内乾球温度＞上限の時　上限値の特性で運転
            this.message.append( "(C)室DB＞上限→上限値で特性計算");
            dbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
        }
        if( dbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() ) ){
            //室内乾球温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)室DB＜下限→下限値で特性計算");
            dbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
        }

        //暖房　処理熱量
        rCode0 = this.pacDB.getKhtw( twatIn );//水温補正
        rCode1 = this.pacDB.getKhf( this.fRate_watInHS / this.fRate_watInHSh );//水量補正
        rCode2 = this.pacDB.getKhti( dbRA );//室温補正
        rCode3 = this.Lratemin;//配管補正
        //        System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode1+" "+rCode2+" "+rCode3);

        //        System.out.println("allTheat="+allTheat+" T_Loadout="+T_Loadout+" Rh="+Rc+" "+rCode1+" "+rCode2+"
"+rCode3);


        this.Sh_Loadout  = rCode0 * rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

        this.Rh = this.allTh_heat / this.Sh_Loadout;

        this.CAPrateC = 0.0;
        this.CAPrateH = 1.0;
        this.Th_Loadout = this.allTh_heat;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        if( this.Rh > 1.0 ) {
            this.CAPrateH = 1.0 / this.Rh;
            this.Rh = 1.0;
            this.Th_Loadout = this.Sh_Loadout;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        }
        //        System.out.println("Rh="+Rh);

        //低負荷領域の計算仕分け:Rh
        if( this.Rh < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                Rp = 0;
                break;

            case 1://  1_下限入力値固定
            case 2://  2_下限COP値固定
            case 3://  3_下限入力値と中間切片
                Rp = this.in_run_stop;
                break;

            default:
                Rp = 0;
            }
        }else{
            Rp = this.Rh;
        }

        //暖房　消費電力
        alph = this.pacDB.getAlphah( Rp);
        beta = this.pacDB.getBetah( Rp, D_fmh, D_pmh );

        if( D_pmh * D_fmh < 0.001 ){
            beta = 1.0;
        }

        rCode0 = alph * beta;
        rCode1 = this.pacDB.getKhhpid( Rp );
        rCode2 = this.pacDB.getKhwtw( twatIn );//水温補正
```

```
rCode3 = this.pacDB.getKhwf( this.fRate_watInHS / this.fRate_watInHSh );//水量補正

rCode4 = this.pacDB.getKhwti( dbRA );//室温補正

this.PPE = rCode0 *rCode1 * rCode2 * rCode3 * rCode4 * D_out_PPEh_S;
//低負荷領域の計算仕分け:PPE
if( this.Rh < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.PPE = 0.0;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      this.PPE *= ( this.Rh / Rp );
      break;

   case 3://  3_下限入力値と中間切片
      this.PPE *= ( 0.5 + this.Rh / Rp / 2. );
      break;

   default:
   }
}else{
   //何もしない
}


//       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);

//airOutEA
//this.airOutEA.setFlowRate( this.fRate_airEA );
//this.airOutEA.setTempDB( ( -this.Th_Loadout + this.PPE ) / this.fRate_airEA * 1000. + this.airInOA.getTempDB() );
//watOutHS
this.watOutHS.setFlowRate( this.fRate_watInHS );
this.watOutHS.setTemp( ( -this.Th_Loadout + this.PPE ) / this.fRate_watInHS / 4.18605 + this.t_watInHS );

if( this.Rh < 0.0001 ){
   this.CAPrateC = 0.0;
   this.CAPrateH = 0.0;
   this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
   this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
   this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
   this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
   this.PPE = 0.0;
   this.watOutHS.copyAllState( this.watInHS );//20130110

}else if( this.Rh < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.CAPrateC = 0.0;
      this.CAPrateH = 0.0;
      this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
      this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
      this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
      this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
      this.PPE = 0.0;
      this.watOutHS.copyAllState( this.watInHS );//20130110
      break;

   case 1://  1_下限入力値固定
      break;

   case 2://  2_下限COP値固定
      break;

   case 3://  3_下限入力値と中間切片
      break;
```

```
        default:
        }
    }

        //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
}

private void heating_and_cooling_cal(){
    double rCode0,rCode1,rCode2,rCode3,rCode4;
    double D_fmh,D_pmh;
    double D_out_Qh_S, D_out_Qc_S;
    double D_out_PPEh_S,alph,beta;
    double deltaWT;
    double Rch;
    double Rp;

     D_fmh      = this.fmh;
     D_pmh      = this.pmh;
     D_out_Qh_S = this.out_Qh_S;
     D_out_Qc_S = this.out_Qc_S;
     D_out_PPEh_S= this.out_PPEh_S;

     //if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L > 0.0001) { //低温入力の場合
     //   D_fmh    = this.fmh;
     //   D_pmh    = this.pmh;
     //   D_out_Qh_S  = this.out_Qh_L;
     //   D_out_PPEh_S= this.out_PPEh_L;
     //}

    //上下限のチェック 20120821nino
    double twatIn = this.t_watInHS;
    double dbRA = this.DBra;

    if( twatIn > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhmtw() ) ){
        //熱源水入口温度＞上限の時　上限値の特性で運転
        this.message.append( "(C)熱源水入口温度＞上限→上限値で特性計算");
        twatIn = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhmtw() );
    }
    if( twatIn < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhmtw() ) ){
        //熱源水入口温度＜下限の時　停止
        this.message.append( "(C)熱源水入口温度＜下限→停止");
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.PPE = 0.0;
        this.watOutHS.copyAllState( this.watInHS );//20130110
        return;
    }
    if( dbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() ) ){
        //室内乾球温度＞上限の時　上限値の特性で運転
        this.message.append( "(C)室DB＞上限→上限値で特性計算");
        dbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
    }
    if( dbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() ) ){
        //室内乾球温度＜下限の時　下限値の特性で運転
        this.message.append( "(C)室DB＜下限→下限値で特性計算");
        dbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
    }


     //暖房　処理熱量
     //定格暖房容量
     rCode0 = this.pacDB.getKhmtw( twatIn );//水温補正
     rCode1 = this.pacDB.getKhmf( this.fRate_watInHS / this.fRate_watInHSh );//水量補正
    rCode2 = this.pacDB.getKhti( dbRA );//室温補正
    rCode3 = this.Lratemin;//配管補正
    this.Sh_Loadout  = rCode0 * rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

    Rch = this.allTc_heat / this.Sh_Loadout;
```

```
        deltaWT = this.pacDB.getKdwt( Rch );

        //        System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode1+" "+rCode2+" "+rCode3);

        //        System.out.println("allTheat="+allTheat+" T_Loadout="+T_Loadout+" Rh="+Rc+" "+rCode1+" "+rCode2+"
"+rCode3);
        //補正暖房容量
         rCode0 = this.pacDB.getKhmtw( twatIn + deltaWT );

         this.Sh_Loadout  = rCode0 * rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

        this.Rh = this.allTh_heat / this.Sh_Loadout;

        this.CAPrateH = 1.0;
        this.Th_Loadout = this.allTh_heat;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        if( this.Rh > 1.0 ) {
           this.CAPrateH = 1.0 / this.Rh;
           this.Rh = 1.0;
           this.Th_Loadout = this.Sh_Loadout;//室外機実処理熱量（加熱全熱）[W]//20110622nino
        }
        //        System.out.println("Rh="+Rc);
        //低負荷領域の計算仕分け:Rh
        if( this.Rh < this.in_run_stop ){
           switch( this.num_calcTypeLowerRangeLoad){
           case 0://  0_発停運転
              Rp = 0;
              break;

           case 1://  1_下限入力値固定
           case 2://  2_下限COP値固定
           case 3://  3_下限入力値と中間切片
              Rp = this.in_run_stop;
              break;

           default:
              Rp = 0;
           }
        }else{
           Rp = this.Rh;
        }


        //暖房　消費電力
        alph = this.pacDB.getAlphah( Rp);
        beta = this.pacDB.getBetah( Rp, D_fmh, D_pmh );

        if( D_pmh * D_fmh < 0.001 ){
           beta = 1.0;
        }

        //入力
        rCode0 = alph * beta;
        rCode1 = this.pacDB.getKhhpid( Rp );
        rCode2 = this.pacDB.getKhmwtw( twatIn + deltaWT );//水温補正
        rCode3 = this.pacDB.getKhmwf( this.fRate_watInHS / this.fRate_watInHSh );//水量補正
        rCode4 = this.pacDB.getKhwti( dbRA );//室温補正

        this.PPE = rCode0 *rCode1 * rCode2 * rCode3 * rCode4 * D_out_PPEh_S;
        //低負荷領域の計算仕分け:PPE
        if( this.Rh < this.in_run_stop && Rp > 0 ){
           switch( this.num_calcTypeLowerRangeLoad){
           case 0://  0_発停運転
              this.PPE = 0.0;
              break;

           case 1://  1_下限入力値固定
              //何もしない
              break;

           case 2://  2_下限COP値固定
              this.PPE *= ( this.Rh / Rp );
              break;
```

```
   case 3://  3_下限入力値と中間切片
      this.PPE *= ( 0.5 + this.Rh / Rp / 2. );
      break;

   default:
   }
}else{
   //何もしない
}


//       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);

//冷房  処理熱量
rCode2 = this.pacDB.getKcti( this.WBra);//室温補正
rCode3 = this.Lratemin;//配管補正

this.Sc_Loadout  = rCode2 * rCode3 * D_out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

this.Rc = this.allTc_heat / this.Sc_Loadout;

this.CAPrateC = 1.0;
this.Tc_Loadout = this.allTc_heat;//室外機実処理熱量（冷却全熱）[W]//20110622nino
if( this.Rc > 1.0 ) {
   this.CAPrateC = 1.0 / this.Rc;
   this.Rc = 1.0;
   this.Tc_Loadout = this.Sc_Loadout;//室外機実処理熱量（冷却全熱）[W]//20110622nino
}


//airOutEA
//this.airOutEA.setFlowRate( this.fRate_airEA );
//this.airOutEA.setTempDB( ( -this.Tc_Loadout + this.PPE ) / this.fRate_airEA * 1000. + this.airInOA.getTempDB() );
//watOutHS
this.watOutHS.setFlowRate( this.fRate_watInHS );
this.watOutHS.setTemp( ( -this.Th_Loadout + this.PPE ) / this.fRate_watInHS / 4.18605 + this.t_watInHS );


if( this.Rh < 0.0001 ){
   this.CAPrateC = 0.0;
   this.CAPrateH = 0.0;
   this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
   this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
   this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
   this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
   this.PPE = 0.0;
   this.watOutHS.copyAllState( this.watInHS );//20130110

}else if( this.Rh < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.CAPrateC = 0.0;
      this.CAPrateH = 0.0;
      this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
      this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
      this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
      this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
      this.PPE = 0.0;
      this.watOutHS.copyAllState( this.watInHS );//20130110
      break;

   case 1://  1_下限入力値固定
      break;

   case 2://  2_下限COP値固定
      break;

   case 3://  3_下限入力値と中間切片
      break;

   default:
```

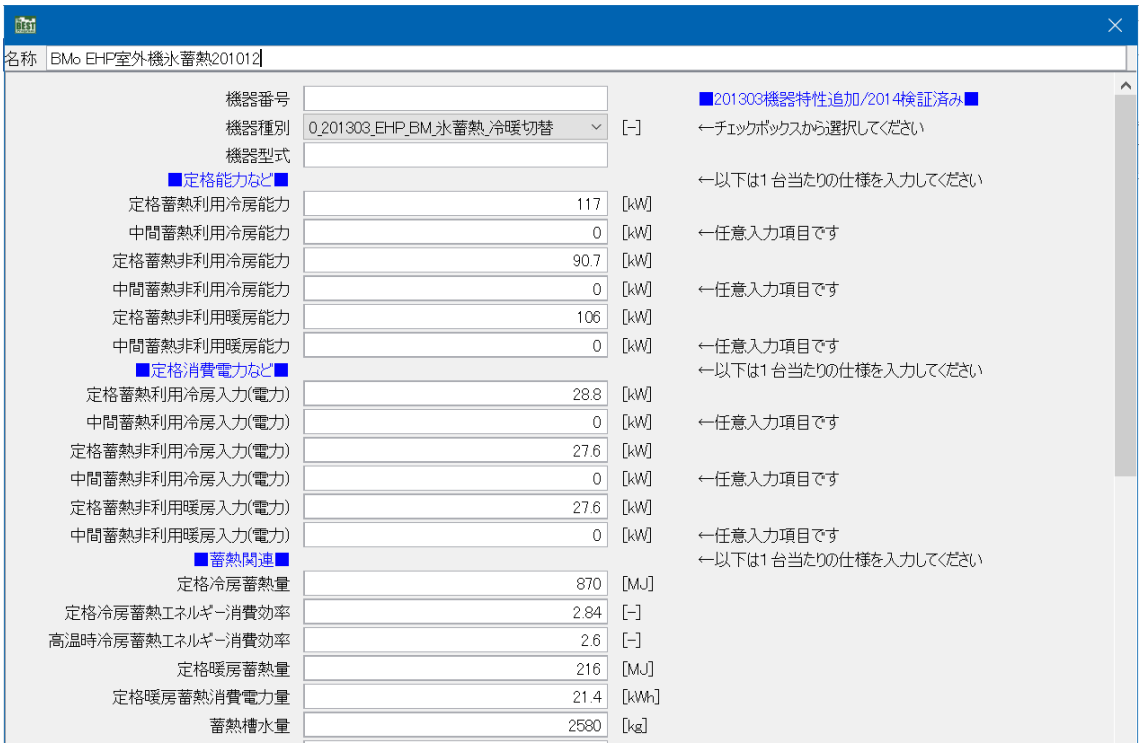
```
            }
        }

            //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
    }


    private double WB_cal( double ddb, double xx ){
        double wwb, x1, de = 1;
        int j, kk = -1, j = 0;
        wwb = ddb;

        do{
            j++;
            x1 = Psychrometrics.FNXtw( ddb, wwb );
            if( Math.abs(x1-xx) < 0.000003 ) {
//          System.out.println(" j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
            return wwb;
            }
            if( x1 < xx ) {
                if( kk == -1 ){
                    de = de / 2.0;
                }
                wwb = wwb + de;
                kk = 1;
            }
                if(x1 >= xx ) {
                if( kk == 1 ){
                    de = de / 2.0;
                }
                wwb = wwb - de;
                kk = -1;
            }

        }while( j < 1000 );
        //    System.out.println("*j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
        return wwb;

    }
}
```

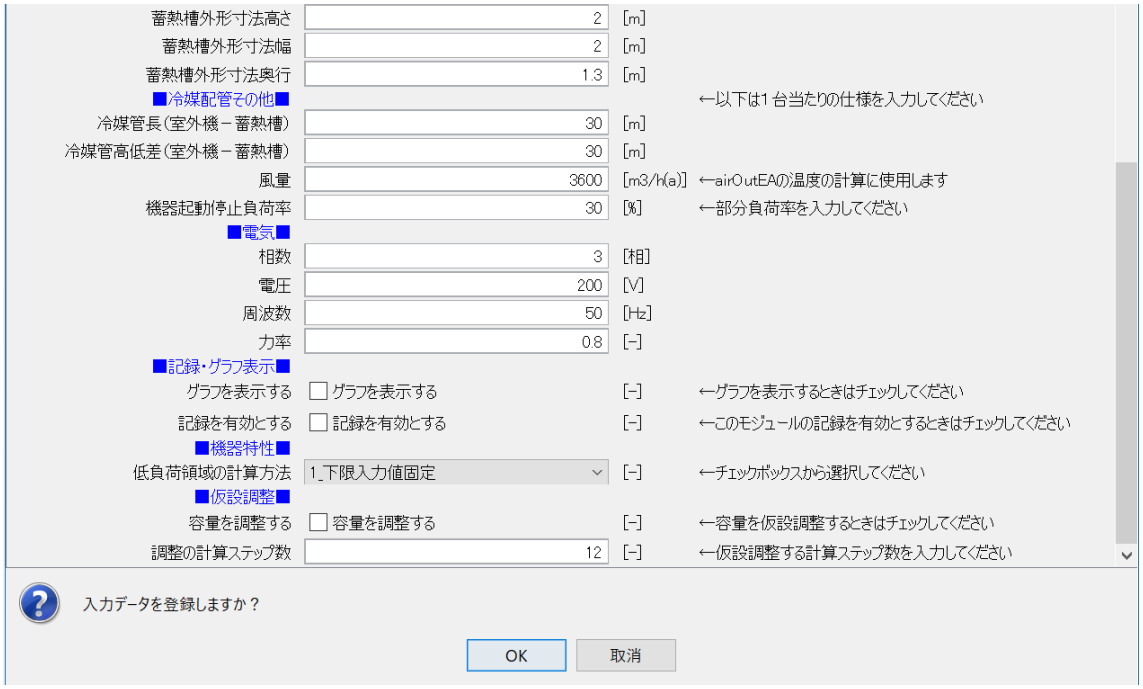
「BMo EHP 室外機氷蓄熱 201012」（場所：設備 2015／個別分散 2015／）

モジュール名	BMo EHP 室外機氷蓄熱 201012
クラス	BuillMultiEHPOut_IceModule20101212

（1）入力画面

・スペック





（2）モジュールの概要

氷蓄熱タイプのビル用マルチ室外機モジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

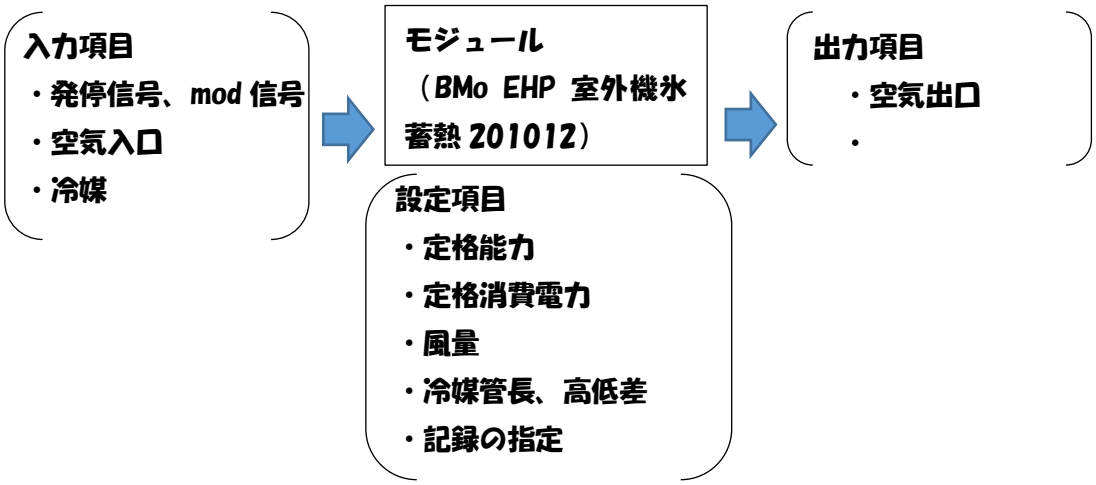


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	機器番号	String	m_no			―	―		
2	機器種別	String	m_ty	0_201303_EHP_BM_氷蓄熱_冷暖切替、0_EHP_BM_氷蓄熱_冷暖切替 200909	[-]	―	―		←チェックボックスから選択してください
3	機器型式	String	m_kt			―	―		
4	■定格能力など■					―	―		←以下は１台当たりの仕様を入力してください
5	定格蓄熱利用冷房能力	double	out_QSc_S	117	[kW]	―	0		
6	中間蓄熱利用冷房能力	double	out_QSc_C	0	[kW]	―	―		←任意入力項目です
7	*定格蓄熱利用暖房能力	double		0	[kW]	―	0		
8	*中間蓄熱利用暖房能力	double		0	[kW]	―	―		←任意入力項目です
9	定格蓄熱非利用冷房能力	double	out_Qc_S	90.7	[kW]	―	0		
10	中間蓄熱非利用冷房能力	double	out_Qc_C	0	[kW]	―	―		←任意入力項目です
11	定格蓄熱非利用暖房能力	double	out_Qh_S	106	[kW]	―	0		
12	中間蓄熱非利用暖房能力	double	out_Qh_C	0	[kW]	―	―		←任意入力項目です
13	*低温暖房能力	double		0	[kW]	―	―		←任意入力項目です
14	■定格消費電力など■					―	―		←以下は１台当たりの仕様を入力してください
15	定格蓄熱利用冷房入力(電力)	double	out_PPESc_S	28.8	[kW]	―	0		
16	中間蓄熱利用冷房入力(電力)	double	out_PPESc_C	0	[kW]	―	―		←任意入力項目です
17	*定格蓄熱利用暖房入力	double		0	[kW]	―	0		

	(電力)								
18	*中間蓄熱利用暖房入力(電力)	double		0	[kW]	―	―		←任意入力項目です
19	定格蓄熱非利用冷房入力(電力)	double	out_PPEc_S	27.6	[kW]	―	0		
20	中間蓄熱非利用冷房入力(電力)	double	out_PPEc_C	0	[kW]	―	―		←任意入力項目です
21	定格蓄熱非利用暖房入力(電力)	double	out_PPEh_S	27.6	[kW]	―	0		
22	中間蓄熱非利用暖房入力(電力)	double	out_PPEh_C	0	[kW]	―	―		←任意入力項目です
23	*低温暖房入力(電力)	double		0	[kW]	―	―		←任意入力項目です
24	■蓄熱関連■					―	―		←以下は１台当たりの仕様を入力してください
25	定格冷房蓄熱量	double	out_STQQc_S	870	[MJ]	―	0		
26	定格冷房蓄熱エネルギー消費効率	double	out_COP_STc_S	2.84	[-]	―	0		
27	高温時冷房蓄熱エネルギー消費効率	double	out_COP_STc_H	2.6	[-]	―	0		
28	定格暖房蓄熱量	double	out_STQQh_S	216	[MJ]	―	0		
29	定格暖房蓄熱消費電力量	double	out_PEESTh_S	21.4	[kWh]	―	0		
30	蓄熱槽水量	double	out_ST_M_W	2580	[kg]	―	0		
31	蓄熱槽外形寸法高さ	double	out_ST_Lh	2	[m]	―	0		
32	蓄熱槽外形寸法幅	double	out_ST_LW	2	[m]	―	0		
33	蓄熱槽外形寸法奥行	double	out_ST_LD	1.3	[m]	―	0		
34	■冷媒配管その他■					―	―		←以下は１台当たりの仕様を入力してください
35	冷媒管長（室外機ー蓄熱槽）	double	out_Lp	30	[m]	―	0		

36	冷媒管高低差 （室外機ー蓄熱槽）	double	out_Lh	30	[m]	ー	0		
37	風量	double	fRate_airEA	3600	[m3/h(a)]	ー	0		←airOutEA の温度の計算に使用します
38	機器起動停止負荷率	double	in_run_stop	30	[%]	100	0		←部分負荷率を入力してください
39	■電気■					ー	ー		
40	相数	int	phase	3	[相]	ー	1		
41	電圧	double	voltage	200	[V]	ー	100		
42	周波数	double	frequency	50	[Hz]	60	50		
43	力率	double	powerFactor	0.8	[-]	1	0		
44	■記録・グラフ表示■					ー	ー		
45	グラフを表示する	boolean	isGVisible	FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
46	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください
47	■機器特性■					ー	ー		
48	低負荷領域の計算方法	String	calcTypeLowerRangeLoad	0_発停運転、1_下限入力値固定、2_下限COP 値固定、3_下限入力値と中間切片	[-]	ー	ー		←チェックボックスから選択してください
49	■仮設調整■					ー	ー		
50	容量を調整する	boolean	isAdjust2012	FALSE	[-]	ー	ー		←容量を仮設調整するときはチェックしてください
51	調整の計算ステップ数	int	numAdjustSteps	12	[-]	ー	1		←仮設調整する計算ステップ数を入力してください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	外気入口	L0_airInOA	airInOA	-	状態	空気	入口	
5	排気出口	L0_airOutEA	airOutEA	-	状態	空気	出口	
6	冷媒入口	L0_valInLine	valInLine	-	値	値	入口	
7	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	室外機 Message#-#-	メッセージ	—	メッセージ
2	室内機冷却要求顕熱量合計#W#熱量	室内機冷却要求顕熱量合計	W	My
3	室内機加熱要求顕熱量合計#W#熱量	室内機加熱要求顕熱量合計	W	My
4	室内機冷却要求全熱量合計#W#熱量	室内機冷却要求全熱量合計	W	My
5	室内機加熱要求全熱量合計#W#熱量	室内機加熱要求全熱量合計	W	My
6	室内機冷却能力補正比率#-#-	室内機冷却能力補正比率	—	My
7	室内機加熱能力補正比率#-#-	室内機加熱能力補正比率-	—	My
8	室外機消費電力#W#消費電力	室外機消費電力	W	エネルギー消費
9	室外機冷却可能熱量/最大#W#熱量	室外機冷却可能熱量/最大	W	—
10	室外機加熱可能熱量/最大#W#熱量	室外機加熱可能熱量/最大	W	—
11	室外機冷却実処理熱量/現状#W#熱量	室外機冷却実処理熱量/現状	W	My
12	室外機加熱実処理熱量/現状#W#熱量	室外機加熱実処理熱量/現状	W	My
13	室外機冷却蓄熱#W#熱量	室外機冷却蓄熱	W	My
14	室外機加熱蓄熱#W#熱量	室外機加熱蓄熱	W	My
15	室外機蓄熱量#J#熱量	室外機蓄熱量	J	My
16	処理全熱量合計負荷#W#-	処理全熱量合計負荷	W	負荷
17	室外機外気入口 DB#℃#-	室外機外気入口 DB	℃	入口
18	室外機外気 WB#℃#-	室外機外気 WB	℃	入口
19	＊室外機外気流量#g/s#-	室外機外気流量	g/s	
20	室外機調整冷却能力#W#熱量	室外機調整冷却能力	W	調整
21	室外機調整加熱能力#W#熱量	室外機調整加熱能力	W	調整

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Map;
import java.util.Set;

import jp.or.ibec.best.DO.BM_EHPdata;
import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.essentials.utility.Psychrometrics;

/**
 * @author SUGANAGA 2008/04/07 /20080428 NINOMIYA/20101212
 * ビルマルチ：室外機クラス
 *
 * 氷蓄熱/20101212nino
 * 201303機器特性追加
 */
public class BuillMultiEHPOut_IceModule20101212 extends AbstractBestModule implements
IBestMetaModule  {

   /**
    * 論理アドレス
    */
   private final String moduleName="(BuillMultiEHPOut_IceModule20101212)";
   private final String S_NODE_airInOA    = "LO_airInOA";//外気
   private final String S_NODE_airOutEA   = "LO_airOutEA";//排気
   private final String S_NODE_valInLine  = "LO_valInLine";//単線
   private final String S_NODE_eleIn      = "LO_eleIn";//電力
   private final String C_NODE_swcIn = "L1_swcIn";//運転状態：off/on
   private final String C_NODE_modIn = "L1_modIn";//空調モード：停止/冷却/加熱
   private final String R_NODE = "L2_recOut";
   private Map<String,BM_EHPdata> rmlinemap=null;

   /**
    * 外部定義
    */
   private final String SPEC_name          = "名称";
   private final String SPEC_m_no          = "機器番号";
   private final String SPEC_m_ty          = "機器種別";
   private final String SPEC_m_kt          = "機器型式";
   //
   private final String SPEC_out_QSc_S      = "定格蓄熱利用冷房能力[W]";
   private final String SPEC_out_QSc_C      = "中間蓄熱利用冷房能力[W]";
   private final String SPEC_out_PPESc_S    = "定格蓄熱利用冷房入力(電力)[W]";
   private final String SPEC_out_PPESc_C    = "中間蓄熱利用冷房入力(電力)[W]";
   //
   private final String SPEC_out_Qc_S      = "定格蓄熱非利用冷房能力[W]";
   private final String SPEC_out_Qc_C      = "中間蓄熱非利用冷房能力[W]";
   private final String SPEC_out_PPEc_S    = "定格蓄熱非利用冷房入力(電力)[W]";
   private final String SPEC_out_PPEc_C    = "中間蓄熱非利用冷房入力(電力)[W]";
   //
   private final String SPEC_out_Qh_S      = "定格蓄熱非利用暖房能力[W]";
   private final String SPEC_out_Qh_C      = "中間蓄熱非利用暖房能力[W]";
   //private final String SPEC_out_Qh_L      = "低温蓄熱非利用暖房能力[W]";
   private final String SPEC_out_PPEh_S    = "定格蓄熱非利用暖房入力(電力)[W]";
```

```
    private final String SPEC_out_PPEh_C    = "中間蓄熱非利用暖房入力(電力)[W]";
    //private final String SPEC_out_PPEh_L    = "低温蓄熱非利用暖房入力(電力)[W]";
/*
    private final String SPEC_out_CLEa_R    = "クランクケースヒータ(運転時)[W]";
    private final String SPEC_out_CLEa_S    = "クランクケースヒータ(停止時)[W]";
    private final String SPEC_out_TKEa_R    = "定格待機電力(運転時)[W]";
    private final String SPEC_out_TKEa_T    = "定格待機電力(待機時)[W]";
    private final String SPEC_out_TKEa_S    = "定格待機電力(停止時)[W]";
*/
    private final String SPEC_out_STQc_S   = "定格冷房蓄熱量[J]";
    private final String SPEC_out_COP_STc_S   = "定格冷房蓄熱エネルギー消費効率[-]";
    private final String SPEC_out_COP_STc_H   = "高温時冷房蓄熱エネルギー消費効率[-]";
    private final String SPEC_out_STQh_S   = "定格暖房蓄熱量[J]";
    private final String SPEC_out_PEESTh_S = "定格暖房蓄熱消費電力量[Ws]";

    private final String SPEC_out_ST_M_W   = "蓄熱槽水量[g]";
    private final String SPEC_out_ST_Lh    = "蓄熱槽外形寸法高さ[m]";
    private final String SPEC_out_ST_LW    = "蓄熱槽外形寸法幅  [m]";
    private final String SPEC_out_ST_LD    = "蓄熱槽外形寸法奥行[m]";

    private final String SPEC_out_Lp       = "冷媒管長 (室外機ー蓄熱槽) [m]";
    private final String SPEC_out_Lh       = "冷媒管高低差 (室外機ー蓄熱槽) [m]";

    private final String SPEC_fRate_airEA   = "風量[g/s]";

    private final String SPEC_in_run_stop  = "機器起動停止負荷率[-]";
    private final String SPEC_Phase       = "相数[-]";
    private final String SPEC_Voltage     = "電圧[V]";
    private final String SPEC_Frequency   = "周波数[Hz]";
    private final String SPEC_PowerFactor = "力率[-]";
    //
    private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
    private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

    private final String SPEC_CalcTypeLowerRangeLoad = "低負荷領域の計算方法";

    private final String SPEC_isAdjust2012   = "容量を調整する";
    private final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";

    private StringBuffer message= new StringBuffer();

    private BestAir airInOA  = null;
    private BestAir airOutEA = null;
    private BestElectricity eleIn = null;  //電力
    private Double CAPrateC = new Double( 1 );//室内機能力補正比率[-]冷房側
    private Double CAPrateH = new Double( 1 );//室内機能力補正比率[-]暖房側
    private BM_EHPdata RMdata = null;   //電力
    private PACDBManager pacDB=null;

    private int modIn  ;
    private int swcIn ;
    private int mState;  //停止、冷房、暖房フラグ

    private boolean isCOOLandHEATout = false;//冷暖同時機種の場合=true
    private int dbflg=0;


    /**
     * 変数 (外部定義に対応)
     */

    private String name             ;// "機器名称";
    private String m_no             ;//= "機器番号";
    private String m_ty             ;//= "機器種別";
    private String m_kt             ;//= "機器型式";
    //
    private double out_QSc_S     ;//= "定格蓄熱利用冷房能力";//kW
    private double out_QSc_C     ;//= "中間蓄熱利用冷房能力";//kW
    private double out_PPESc_S   ;//= "定格蓄熱利用冷房入力(電力)";//kW
    private double out_PPESc_C   ;//= "中間蓄熱利用冷房入力(電力)";//kW
    //
    private double out_Qc_S      ;//= "定格蓄熱非利用冷房能力";//kW
```

```
    private double out_Qc_C      ;//= "中間蓄熱非利用冷房能力";//kW
    private double out_PPEc_S    ;//= "定格蓄熱非利用冷房入力(電力)";//kW
    private double out_PPEc_C    ;//= "中間蓄熱非利用冷房入力(電力)";//kW
    //
    private double out_Qh_S      ;//= "定格蓄熱非利用暖房能力";//kW
    private double out_Qh_C      ;//= "中間蓄熱非利用暖房能力";//kW
    //private double out_Qh_L      ;//= "低温蓄熱非利用暖房能力";//kW
    private double out_PPEh_S    ;//= "定格蓄熱非利用暖房入力(電力)";//kW
    private double out_PPEh_C    ;//= "中間蓄熱非利用暖房入力(電力)";//kW
    //private double out_PPEh_L    ;//= "低温蓄熱非利用暖房入力(電力)";//kW
/*
    private double out_CLEa_R=0.0    ;//= "クランクケースヒータ(運転時)";//W
    private double out_CLEa_S=0.0    ;//= "クランクケースヒータ(停止時)";//W
    private double out_TKEa_R =0.0   ;//= "定格待機電力(運転時)";//W
    private double out_TKEa_T =0.0   ;//= "定格待機電力(待機時)";//W
    private double out_TKEa_S =0.0  ;//= "定格待機電力(停止時)";//W
*/
    private double out_STQQc_S    ;//= "定格冷房蓄熱量[J]";
    private double out_COP_STc_S  ;//= "定格冷房蓄熱エネルギー消費効率[-]";
    private double out_COP_STc_H  ;//= "高温時冷房蓄熱エネルギー消費効率[-]";
    private double out_STQQh_S    ;//= "定格暖房蓄熱量[J]";
    private double out_PEESTh_S   ;//= "定格暖房蓄熱消費電力量[Ws]";

    private double out_ST_M_W     ;//= "蓄熱槽水量[g]";
    private double out_ST_Lh      ;//= "蓄熱槽外形寸法高さ[m]";
    private double out_ST_LW      ;//= "蓄熱槽外形寸法幅  [m]";
    private double out_ST_LD      ;//= "蓄熱槽外形寸法奥行[m]";

    private double out_Lp         ;//= "冷媒管長 (室外機ー蓄熱槽) [m]";
    private double out_Lh         ;//= "冷媒管高低差 (室外機ー蓄熱槽) [m]";

    private double fRate_airEA ;//風量[g/s]

    private double in_run_stop    ;//= "機器起動停止負荷率";//[%]
    private int phase;         //相数[-]
    private double voltage;    //電圧[V]
    private double frequency;  //周波数[Hz]
    private double powerFactor;   //力率[-]

    private double Lp;         //冷媒管長さ[m]
    private double Lh;         //室内機/室外機高低差[m]
    private boolean isGVisible = false;//このグラフを表示する=true
    private boolean isRecord   = false;//記録を有効とする=true

    private String[] rmnm=new String[50];

    /**
     * 出力 (建物)
     */

    private final String RECORD_message = "室外機Message#-#-";
    private final String RECORD_ID1c = "室内機冷却要求顕熱量合計#W#熱量";
    private final String RECORD_ID1h = "室内機加熱要求顕熱量合計#W#熱量";
    private final String RECORD_ID2c = "室内機冷却要求全熱量合計#W#熱量";
    private final String RECORD_ID2h = "室内機加熱要求全熱量合計#W#熱量";
    private final String RECORD_ID3c = "室内機冷却能力補正比率#-#-";
    private final String RECORD_ID3h = "室内機加熱能力補正比率#-#-";
    private final String RECORD_ID4  = "室外機消費電力#W#消費電力";
    private final String RECORD_ID5c = "室外機冷却可能熱量/最大#W#熱量";
    private final String RECORD_ID5h = "室外機加熱可能熱量/最大#W#熱量";
    private final String RECORD_ID6c = "室外機冷却実処理熱量/現状#W#熱量";
    private final String RECORD_ID6h = "室外機加熱実処理熱量/現状#W#熱量";
    private final String RECORD_QstC = "室外機冷却蓄熱#W#熱量";
    private final String RECORD_QstH = "室外機加熱蓄熱#W#熱量";
    private final String RECORD_stQQ = "室外機蓄熱量#J#熱量";
    private final String RECORD_qT   = "処理全熱量合計負荷#W#-";

    private final String RECORD_DBairInHS = "室外機外気入口DB#℃#-";
    private final String RECORD_WBairInHS = "室外機外気WB#℃#-";
    private final String RECORD_M_airInHS = "室外機外気流量#g/s#-";

    private final String RECORD_out_Qc_Adjust = "室外機調整冷却能力#W#熱量";
```

```
private final String RECORD_out_Qh_Adjust = "室外機調整加熱能力#W#熱量";

/**
 * その他変数
 */

private double Rc;               //部分負荷率　冷房側
private double Rh;               //部分負荷率　暖房側

private double DBoa;          //外気乾球温度[℃]
private double WBoa;          //外気湿球温度[℃]
private double XGoa;          //外気絶対湿度[g/gD.A.]

private double DBra;          //室内機吸込乾球温度[℃]
private double WBra;          //室内機吸込湿球温度[℃]
private double XGra;          //室内機吸込絶対湿度[g/gD.A.]

private double Sc_Loadout;//室外機処理可能熱量（冷却全熱）[W]
private double Sh_Loadout;//室外機処理可能熱量（加熱全熱）[W]
private double Tc_Loadout;//室外機実処理熱量（冷却全熱）[W]
private double Th_Loadout;//室外機実処理熱量（加熱全熱）[W]
private double allSc_heat;//室内機処理熱量（冷却顕熱）[W]
private double allSh_heat;//室内機処理熱量（加熱顕熱）[W]
private double allTc_heat;//室内機処理熱量（冷却全熱）[W]
private double allTh_heat;//室内機処理熱量（加熱全熱）[W]

private double PPE;  //室外機消費電力[W]
private double E_PPE;//室外機無効電力[]

private int num;
private String kiki_file;
private String path="EHP";
private String[] filenames= new String[1];
private String equipmentName;

private double DBra_max;
private double WBra_min;
private double Lratemin;
private double fmc;
private double pmc;
private double fmSc;
private double pmSc;
private double fmh;
private double pmh;

private double out_STQQc_Co;//冷房蓄熱消費量（Ws）
private double out_STQQh_defrost_Co;//暖房デフロストによる暖房蓄熱消費量（Ws）
private double out_STGh_T;//= 0;//暖房蓄熱除霜利用回数

private double out_ST_SA;//蓄熱槽の表面積

private double out_Qcc_S  ;//= "定格蓄熱冷房能力"
private double out_Qhc_S  ;//= "定格蓄熱暖房能力"
private double out_Qcc  ;//= "蓄熱冷房能力"
private double out_Qhc  ;//= "蓄熱暖房能力"
private double out_STQQ;//現在の蓄熱量[J]　０℃基準

private int tInterval;//現在の計算時間間隔[s]

private int opeTimeCount = 0;//積算蓄熱運転時間[s]
private int opeTimeOld = 3600 * 10;//前回の蓄熱運転時間[s]
private int opeTimeMax = 3600 * 10;//予定蓄熱運転時間[s]
private int opeTimeAdd = 0;//蓄熱運転予定時間の補正値[s]
private int opeTimeAddNext = 0;//次回の蓄熱運転予定時間の補正値[s]

//グラフ表示など
private GraphJFrameBuilMultiOut_S20101212 gBMout = null;

//低負荷領域の計算方法
private String calcTypeLowerRangeLoad = null;//"低負荷領域の計算方法";
private int num_calcTypeLowerRangeLoad;//"低負荷領域の計算方法";
```

```
    //仮設調整モード
    //仮設調整モード2012
    private boolean isAdjust2012  = false;//true="台数を調整する";//
    //private boolean isAdjustMode = true;//20110517nino 空衛学会OS論文検討用＝true
    private double out_Qc_Adjust;//= "調整冷却能力";
    private double out_Qh_Adjust;//= "調整加熱能力";
    private double fRate_airEAc ;//冷却時風量[g/s]
    private double fRate_airEAh ;//加熱時風量[g/s]

    private int numAdjustSteps;//"調整の計算ステップ数";
    private LinkedList<Double> rcList = null;//必要台数
    private LinkedList<Double> rhList = null;//必要台数
    private double max_rcAdjust = 1.;
    private double max_rhAdjust = 1.;
    private double ave_rcAdjust = 0;
    private double ave_rhAdjust = 0;
    private double out_rcAdjust = 1.;//= "調整率冷房";
    private double out_rhAdjust = 1.;//= "調整率暖房";
// private double in_run_stop_Num;//= "機器起動停止負荷率"台数補正;

    private double out_Qc_S_1      ;//= "定格冷房能力";//kW
    private double out_Qc_C_1      ;//= "中間冷房能力";//kW
    private double out_PPEc_S_1    ;//= "定格冷房入力(電力)";//kW
    private double out_PPEc_C_1    ;//= "中間冷房入力(電力)";//kW
    private double out_Qh_S_1      ;//= "定格暖房能力";//kW
    private double out_Qh_C_1      ;//= "中間暖房能力";//kW
// private double out_Qh_L_1      ;//= "低温暖房能力";//kW
    private double out_PPEh_S_1    ;//= "定格暖房入力(電力)";//kW
    private double out_PPEh_C_1    ;//= "中間暖房入力(電力)";//kW
// private double out_PPEh_L_1    ;//= "低温暖房入力(電力)";//kW

    private double out_QSc_S_1     ;//= "定格蓄熱利用冷房能力";//kW
    private double out_QSc_C_1     ;//= "中間蓄熱利用冷房能力";//kW
    private double out_PPESc_S_1   ;//= "定格蓄熱利用冷房入力(電力)";//kW
    private double out_PPESc_C_1   ;//= "中間蓄熱利用冷房入力(電力)";//kW
    //
    private double out_STQQc_S_1  ;//= "定格冷房蓄熱量[J]";
// private double out_COP_STc_S  ;//= "定格冷房蓄熱エネルギー消費効率[-]";
// private double out_COP_STc_H  ;//= "高温時冷房蓄熱エネルギー消費効率[-]";
    private double out_STQQh_S_1     ;//= "定格暖房蓄熱量[J]";
    private double out_PEESTh_S_1 ;//= "定格暖房蓄熱消費電力量[Ws]";

    private double out_ST_M_W_1      ;//= "蓄熱槽水量[g]";
// private double out_ST_Lh      ;//= "蓄熱槽外形寸法高さ[m]";
// private double out_ST_LW      ;//= "蓄熱槽外形寸法幅  [m]";
    private double out_ST_LD_1    ;//= "蓄熱槽外形寸法奥行[m]";



    @Override
    public void setProfile(BestSpecs spec) {
        //    外部定義項目取得
        if(spec == null) {
            return;
        }
        Map<String,String> map=spec.getSpec();
        if(map == null) {
            return;
        }

        // 機器名称
        this.name = spec.getSpecValue( this.SPEC_name, this.name );

        // 機器番号
        this.m_no = spec.getSpecValue( this.SPEC_m_no, "" );

        //  機器種別  0_氷蓄熱型
        this.m_ty = spec.getSpecValue( this.SPEC_m_ty, "0_201303_EHP_BM_氷蓄熱_冷暖切替" );

        //"0_EHP_BM_氷蓄熱_冷暖切替_200909"
        //this.kiki_file = "EHP_BM_Ice_200910";//20110126
```

```
        this.kiki_file = "EHP_BM_Ice_201303";//20110126
        //if( this.m_ty.equals("1_EHP_BM_標準_冷暖切替_寒冷地対応200908")){
        // this.kiki_file = "EHP_BM_Standard_ColdArea200908";
        // this.isCOOLandHEATout = false;
        //}

        // 機器型式
        this.m_kt = spec.getSpecValue( this.SPEC_m_kt, "" );

        // 定格蓄熱利用冷房能力
        this.out_QSc_S = spec.getSpecValue( this.SPEC_out_QSc_S, 0. );

        // 中間蓄熱利用冷房能力
        this.out_QSc_C = spec.getSpecValue( this.SPEC_out_QSc_C, 0. );

        // 定格蓄熱利用冷房入力(電力)
        this.out_PPESc_S = spec.getSpecValue( this.SPEC_out_PPESc_S, 0. );

        // 中間蓄熱利用冷房入力(電力)
        this.out_PPESc_C = spec.getSpecValue( this.SPEC_out_PPESc_C, 0. );

        // 定格蓄熱非利用冷房能力
        this.out_Qc_S = spec.getSpecValue( this.SPEC_out_Qc_S, 0. );

        // 中間蓄熱非利用冷房能力
        this.out_Qc_C = spec.getSpecValue( this.SPEC_out_Qc_C, 0. );

        // 定格蓄熱非利用冷房入力(電力)
        this.out_PPEc_S = spec.getSpecValue( this.SPEC_out_PPEc_S, 0. );

        // 中間蓄熱非利用冷房入力(電力)
        this.out_PPEc_C = spec.getSpecValue( this.SPEC_out_PPEc_C, 0. );

        // 定格蓄熱非利用暖房能力
        this.out_Qh_S = spec.getSpecValue( this.SPEC_out_Qh_S, 0. );

        // 中間蓄熱非利用暖房能力
        this.out_Qh_C = spec.getSpecValue( this.SPEC_out_Qh_C, 0. );

        // 低温蓄熱非利用暖房能力
        //this.out_Qh_L = spec.getSpecValue( this.SPEC_out_Qh_L, 0. );

        // 定格蓄熱非利用暖房入力(電力)
        this.out_PPEh_S = spec.getSpecValue( this.SPEC_out_PPEh_S, 0. );

        // 中間蓄熱非利用暖房入力(電力)
        this.out_PPEh_C = spec.getSpecValue( this.SPEC_out_PPEh_C, 0. );

        // 低温蓄熱非利用暖房入力(電力)
        //this.out_PPEh_L = spec.getSpecValue( this.SPEC_out_PPEh_L, 0. );

    /*
        // クランクケースヒータ(運転時)
        this.out_CLEa_R = spec.getSpecValue( this.SPEC_out_CLEa_R, 0. );

        // クランクケースヒータ(停止時)
        this.out_CLEa_S = spec.getSpecValue( this.SPEC_out_CLEa_S, 0. );

        // 待機電力(運転時)
        this.out_TKEa_R = spec.getSpecValue( this.SPEC_out_TKEa_R, 0. );

        // 待機電力(待機時)
        this.out_TKEa_T = spec.getSpecValue( this.SPEC_out_TKEa_T, 0. );

        // 待機電力(停止時)
        this.out_TKEa_S = spec.getSpecValue( this.SPEC_out_TKEa_S, 0. );
    */
        //private double out_STQc_S      ;//= "定格冷房蓄熱量[J]";
        this.out_STQQc_S = spec.getSpecValue( this.SPEC_out_STQc_S, 0. );

        //private double out_COP_STc_S   ;//= "定格冷房蓄熱エネルギー消費効率[-]";
        this.out_COP_STc_S = spec.getSpecValue( this.SPEC_out_COP_STc_S, 2.84 );
```

```
        //private double out_COP_STc_H   ;//= "高温時冷房蓄熱エネルギー消費効率[-]";
        this.out_COP_STc_H = spec.getSpecValue( this.SPEC_out_COP_STc_H, 2.6 );

        //private double out_STQh_S      ;//= "定格暖房蓄熱量[J]";
        this.out_STQQh_S = spec.getSpecValue( this.SPEC_out_STQh_S, 0. );

        //private double out_PEESTh_S    ;//= "定格暖房蓄熱消費電力量[Ws]";
        this.out_PEESTh_S = spec.getSpecValue( this.SPEC_out_PEESTh_S, 0. );

        //private double out_ST_M_W      ;//= "蓄熱槽水量[g]";
        this.out_ST_M_W = spec.getSpecValue( this.SPEC_out_ST_M_W, 0. );

        //private double out_ST_Lh    ;//= "蓄熱槽外形寸法高さ[m]";
        this.out_ST_Lh = spec.getSpecValue( this.SPEC_out_ST_Lh, 0. );

        //private double out_ST_LW    ;//= "蓄熱槽外形寸法幅  [m]";
        this.out_ST_LW = spec.getSpecValue( this.SPEC_out_ST_LW, 0. );

        //private double out_ST_LD    ;//= "蓄熱槽外形寸法奥行[m]";
        this.out_ST_LD = spec.getSpecValue( this.SPEC_out_ST_LD, 0. );

        //ou_ST_SA 蓄熱槽の表面積
        this.out_ST_SA = ( this.out_ST_LD * this.out_ST_Lh + this.out_ST_LD * this.out_ST_LW + this.out_ST_Lh *
this.out_ST_LW ) * 2;


        // 冷媒配管長
        this.out_Lp = spec.getSpecValue( this.SPEC_out_Lp, 0. );

        // 冷媒配管高低差
        this.out_Lh = spec.getSpecValue( this.SPEC_out_Lh, 0. );

        //SPEC_fRate_airEA   = "風量[g/s]";
        this.fRate_airEA = spec.getSpecValue( this.SPEC_fRate_airEA, this.out_Qc_S * 0.11 );//20110505nino

        // 機器起動停止負荷率
        this.in_run_stop = spec.getSpecValue( this.SPEC_in_run_stop, 0.3 );

        // 相数
        this.phase = spec.getSpecValue( this.SPEC_Phase, 3 );

        // 電圧
        this.voltage = spec.getSpecValue( this.SPEC_Voltage, 200. );

        // 周波数
        this.frequency = spec.getSpecValue( this.SPEC_Frequency, 50. );

        // 力率
        this.powerFactor = spec.getSpecValue( this.SPEC_PowerFactor, 0.8 );

        //isGVisibleを取得
        this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

        //低負荷領域の計算方法
        this.calcTypeLowerRangeLoad = spec.getSpecValue( this.SPEC_CalcTypeLowerRangeLoad, "0_発停運転" );
        //calcTypeLowerRangeLoad
        //  0_発停運転
        //  1_下限入力値固定
        //  2_下限COP値固定
        //  3_下限入力値と中間切片
        if( this.calcTypeLowerRangeLoad.equals( "0_発停運転" )){
           this.num_calcTypeLowerRangeLoad = 0;
        }else if(this.calcTypeLowerRangeLoad.equals( "1_下限入力値固定" )){
           this.num_calcTypeLowerRangeLoad = 1;
        }else if(this.calcTypeLowerRangeLoad.equals( "2_下限COP値固定" )){
           this.num_calcTypeLowerRangeLoad = 2;
        }else if(this.calcTypeLowerRangeLoad.equals( "3_下限入力値と中間切片" )){
           this.num_calcTypeLowerRangeLoad = 3;
```

```
}else{
   this.num_calcTypeLowerRangeLoad = 0;
}


//中間性能　冷房蓄熱非利用
if(out_Qc_S<0.001){
   fmc= 0.0;
}else{
   fmc= out_Qc_C/out_Qc_S;
}
if(out_PPEc_S<0.001){
   pmc= 0.0;
}else{
   pmc= out_PPEc_C/out_PPEc_S;
}

//中間性能　冷房蓄熱利用
if(out_QSc_S<0.001){
   fmSc= 0.0;
}else{
   fmSc= out_QSc_C/out_QSc_S;
}
if(out_PPESc_S<0.001){
   pmSc= 0.0;
}else{
   pmSc= out_PPESc_C/out_PPESc_S;
}

//中間性能　暖房
if(out_Qh_S<0.001){
   fmh= 0.0;
}else{
   fmh= out_Qh_C/out_Qh_S;
}
if(out_PPEh_S<0.001){
   pmh= 0.0;
}else{
   pmh= out_PPEh_C/out_PPEh_S;
}

if(pmc>0.9999) {pmc=0.0;fmc=0.0;}
if(pmSc>0.9999) {pmSc=0.0;fmSc=0.0;}
if(pmh>0.9999) {pmh=0.0;fmh=0.0;}



//isAdjust2012 = "台数を調整する";//
this.isAdjust2012 = spec.getSpecValue( this.SPEC_isAdjust2012, false );
//自動調整する場合は０％まで運転とする。20200108
if( this.isAdjust2012 ) {
   this.in_run_stop = 0;
}

//仮設調整
this.fRate_airEAc = this.fRate_airEA;
this.fRate_airEAh = this.fRate_airEA;

// 調整の計算ステップ数
this.numAdjustSteps = spec.getSpecValue( this.SPEC_NumAdjustSteps, 12 );

this.rcList = new LinkedList<Double>();
this.rhList = new LinkedList<Double>();
for( int i=0; i<this.numAdjustSteps; i++ ){
   this.rcList.add( 0. );
   this.rhList.add( 0. );
}

this.out_Qc_S_1   = this.out_Qc_S;//= "定格冷房能力";//kW
this.out_Qc_C_1   = this.out_Qc_C;//= "中間冷房能力";//kW
this.out_PPEc_S_1 = this.out_PPEc_S;//= "定格冷房入力(電力)";//kW
this.out_PPEc_C_1 = this.out_PPEc_C;//= "中間冷房入力(電力)";//kW
```

```
        this.out_Qh_S_1   = this.out_Qh_S;//= "定格暖房能力";//kW
        this.out_Qh_C_1   = this.out_Qh_C;//= "中間暖房能力";//kW
//      this.out_Qh_L_1   = this.out_Qh_L;//= "低温暖房能力";//kW
        this.out_PPEh_S_1 = this.out_PPEh_S;//= "定格暖房入力(電力)";//kW
        this.out_PPEh_C_1 = this.out_PPEh_C;//= "中間暖房入力(電力)";//kW
//      this.out_PPEh_L_1 = this.out_PPEh_L;//= "低温暖房入力(電力)";//kW

        this.out_QSc_S_1 = this.out_QSc_S;//= "定格蓄熱利用冷房能力";//kW
        this.out_QSc_C_1 = this.out_QSc_C;//= "中間蓄熱利用冷房能力";//kW
        this.out_PPESc_S_1 = this.out_PPESc_S;//= "定格蓄熱利用冷房入力(電力)";//kW
        this.out_PPESc_C_1 = this.out_PPESc_C;//= "中間蓄熱利用冷房入力(電力)";//kW
        //
        this.out_STQQc_S_1 = this.out_STQQc_S;//= "定格冷房蓄熱量[J]";
//      this.out_COP_STc_S_1 = this.out_COP_STc_S;//= "定格冷房蓄熱エネルギー消費効率[-]";
//      this.out_COP_STc_H_1 = this.out_COP_STc_H;//= "高温時冷房蓄熱エネルギー消費効率[-]";
        this.out_STQQh_S_1 = this.out_STQQh_S;//= "定格暖房蓄熱量[J]";
        this.out_PEESTh_S_1 = this.out_PEESTh_S;//= "定格暖房蓄熱消費電力量[Ws]";

        this.out_ST_M_W_1 = this.out_ST_M_W;//= "蓄熱槽水量[g]";
//      this.out_ST_Lh_1 = this.out_ST_Lh;//= "蓄熱槽外形寸法高さ[m]";
//      this.out_ST_LW_1 = this.out_ST_LW;//= "蓄熱槽外形寸法幅   [m]";
        this.out_ST_LD_1 = this.out_ST_LD;//= "蓄熱槽外形寸法奥行[m]";

    }


    @Override
    public void initialize(IBestStateMessage stateNodes,
            IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
            IBestDomainSchedule schedule) {
        //System.out.println(this.moduleName + " BuillMultiOutイニシャライズ");
        //状態ノードを受け取る
        super.sm = stateNodes;
        //制御ノードを受け取る
        super.cm = commandNodes;
        //記録ノードを受け取る
        super.rm = recordNodes;

        //eleIn
        this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

        //airInOA
        this.airInOA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInOA );

        //private final String S_NODE_airOutEA   = "LO_airOutEA";//排気
        this.airOutEA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutEA );

        //private final String S_NODE_valInLine  = "LO_valInLine";  //単線
        if( super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine )) != null ){

            //System.out.println( " BM_out S_NODE_valInLine登録済み" );

            this.rmlinemap
            =  (Map<String, BM_EHPdata>) super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine));

            if( this.rmlinemap.size() == 0 ){
                //
                //System.out.println( "rmlinemap のサイズが０ ");
                this.RMdata = new BM_EHPdata();
                this.RMdata.set_path( this.path );
                this.RMdata.set_kiki( this.kiki_file );
                this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
                this.RMdata.setairInHS( this.airInOA );//20120823nino
                this.rmlinemap.put( "Parent", this.RMdata );

                super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );

            }else{
                this.num = 0;
                Set<String> LineEntry = this.rmlinemap.keySet();
                for(Iterator<String> i = LineEntry.iterator();i.hasNext();){
                    String key = (String) i.next();
```

```
                //          System.out.println(" name="+name+" key="+key);
                this.RMdata = this.rmlinemap.get(key);
                this.rmnm[this.num]=key;
                //          System.out.println("key="+key+" 0="+rmdata[0]+" 1="+rmdata[1]+" 2="+rmdata[2]+"
3="+rmdata[3]+" 4="+rmdata[4]+" 5="+rmdata[5]+" 6="+rmdata[6]);
                this.num++;
            }
            //
            //20150409
            double sum_in_Qc_S = 0;
            double sum_in_Qh_S = 0;
            double sum_in_dai = 0;
            //出力設定
            for(int i=0; i<this.num; i++){
               //System.out.println( "kiki_file Name = " +kiki_file );
               this.RMdata = this.rmlinemap.get( this.rmnm[i] );
               this.RMdata.set_kiki(kiki_file);
               this.RMdata.set_path(this.path);
               this.RMdata.setCOOLandHEAT( this.isCOOLandHEATout );
               this.RMdata.setairInHS( this.airInOA );//20120823nino
               rmlinemap.put(rmnm[i],this.RMdata);
               //20150409
               sum_in_Qc_S += this.RMdata.get_in_Qc_S();
               sum_in_Qh_S += this.RMdata.get_in_Qh_S();
               sum_in_dai += this.RMdata.get_in_dai();
            }
            //
            //System.out.println( "rmlinemap のサイズが "+this.num );
            this.RMdata = new BM_EHPdata();
            this.RMdata.set_path( this.path );
            this.RMdata.set_kiki( this.kiki_file );
            this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
            this.RMdata.setairInHS( this.airInOA );//20120823nino
            //20150409
            this.RMdata.set_in_Qc_S( sum_in_Qc_S );
            this.RMdata.set_in_Qh_S( sum_in_Qh_S );
            this.RMdata.set_in_dai( sum_in_dai );

            this.rmlinemap.put( "Parent", this.RMdata );

            //20150409
      /*    BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
            + "接続定格冷却容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qc_S + "[W](="
                  +AirFormat.df_3( sum_in_Qc_S / this.out_Qc_S ) + "倍)" );
            BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
            + "接続定格加熱容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qh_S + "[W](="
                  +AirFormat.df_3( sum_in_Qh_S / this.out_Qh_S ) + "倍)" );
            BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
            + "接続台数="+AirFormat.df_0( sum_in_dai )+"[台]" );
      */
            //接続容量、台数チェック
            this.RMdata.checkConnect( this.out_Qc_S, this.out_Qh_S, sum_in_Qc_S, sum_in_Qh_S, sum_in_dai,
                  this.moduleName, "initialize", this.name, this.isRecord, this.message );

            super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );
            this.dbflg = 1;
         }
      }else{

         //System.out.println( " BM_out S_NODE_valInLine未登録" );

         //System.out.println( this.moduleName + ">>Warning<< rmlinemap is null !!" );
         message.append( "(W)単線の接続なし→作成");
         this.rmlinemap = new HashMap<String,BM_EHPdata>();
         //
         this.RMdata = new BM_EHPdata();
         this.RMdata.set_path( this.path );
         this.RMdata.set_kiki( this.kiki_file );
         this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
         this.RMdata.setairInHS( this.airInOA );//20120823nino
         this.rmlinemap.put( "Parent", this.RMdata );
```

```
        super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );
    }

    //グラフ表示の準備
    if( this.isGVisible ){
        gBMout = new GraphJFrameBuilMultiOut_S20101212( this.name, 300,  out_Qc_S * 1.5, 0 );
    }


    filenames[0]=kiki_file;
    equipmentName=kiki_file;
    pacDB=new PACDBManager(path,filenames);
    pacDB.setEquipmentName(equipmentName);


    //補正定格冷房蓄熱能力[J]
    this.out_STQQc_S *= ( this.pacDB.getKccLpst( this.out_Lp ) * this.pacDB.getKcchust( this.out_Lh ) );//
    this.out_STQQh_S *= ( this.pacDB.getKhcLpst( this.out_Lp ) * this.pacDB.getKhchust( this.out_Lh ) );//

    //定格冷房蓄熱能力[W]
    this.out_Qcc_S = this.out_STQQc_S / 10.  / 3600.;//蓄熱時間10時間に固定
    this.out_Qhc_S = this.out_STQQh_S / 2.   / 3600.;//蓄熱時間2時間に固定

    this.out_STQQh_defrost_Co = this.out_STQQh_S / 14.;//暖房デフロスト１回の暖房蓄熱消費量[Ws]

}




/**
 * ビルマルチ計算及び結果出力/各室内機よりも先に動きます。
 */
@Override
public void outputs() {

    if (null == super.cm || null == super.sm )
        return;

    if( this.dbflg == 0 ){

        //System.out.println( "    BM_out  outputsで  室側データの設定" );

        //System.out.println( " Parent path="+ this.rmlinemap.get( "Parent" ).get_path() );

        if( super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine )) != null ){
            this.rmlinemap
            =  (Map<String, BM_EHPdata>) super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine));
            //
            this.num = 0;
            Set<String> LineEntry = this.rmlinemap.keySet();
            for(Iterator<String> i = LineEntry.iterator();i.hasNext();){
                String key = (String) i.next();
                //          System.out.println(" name="+name+" key="+key);
                this.RMdata = this.rmlinemap.get(key);
                this.rmnm[this.num]=key;
                //          System.out.println("key="+key+" 0="+rmdata[0]+" 1="+rmdata[1]+" 2="+rmdata[2]+"
3="+rmdata[3]+" 4="+rmdata[4]+" 5="+rmdata[5]+" 6="+rmdata[6]);
                this.num++;
            }
            //20150409
            double sum_in_Qc_S = 0;
            double sum_in_Qh_S = 0;
            double sum_in_dai = 0;
            //出力設定
            for(int i=0; i<this.num; i++){
                //System.out.println( "kiki_file Name = " +kiki_file );
                this.RMdata = this.rmlinemap.get( this.rmnm[i] );
                this.RMdata.set_kiki(kiki_file);
                this.RMdata.set_path(this.path);
                this.RMdata.setCOOLandHEAT( this.isCOOLandHEATout );
```

```
            this.RMdata.setairInHS( this.airInOA );//20120823nino
            rmlinemap.put(rmnm[i],this.RMdata);
            //20150409
            sum_in_Qc_S += this.RMdata.get_in_Qc_S();
            sum_in_Qh_S += this.RMdata.get_in_Qh_S();
            sum_in_dai += this.RMdata.get_in_dai();
        }
        //
        //20150409
        this.RMdata = this.rmlinemap.get( "Parent" );
        //20150409
        this.RMdata.set_in_Qc_S( sum_in_Qc_S );
        this.RMdata.set_in_Qh_S( sum_in_Qh_S );
        this.RMdata.set_in_dai( sum_in_dai );

        this.rmlinemap.put( "Parent", this.RMdata );

        //20150409
/*      BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
        + "接続定格冷却容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qc_S + "[W](
              +AirFormat.df_3( sum_in_Qc_S / this.out_Qc_S ) + "倍)" );
        BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
        + "接続定格加熱容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qh_S + "[W](
              +AirFormat.df_3( sum_in_Qh_S / this.out_Qh_S ) + "倍)" );
        BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
        + "接続台数="+AirFormat.df_0( sum_in_dai )+"[台]" );
*/      //
        //接続容量、台数チェック
        this.RMdata.checkConnect( this.out_Qc_S, this.out_Qh_S, sum_in_Qc_S, sum_in_Qh_S, sum_in_dai,
              this.moduleName, "outputs", this.name, this.isRecord, this.message );

    }
    this.dbflg = 1;

}

this.tInterval = BestTimeManager.getCurrentInterval();

//message.setLength(0);
//ノード接続毎回読込み
this.airInOA = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airInOA));
this.swcIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcIn) );
this.modIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modIn));

//外気乾球温度設定
this.DBoa = this.airInOA.getTempDB();

//外気絶対湿度を設定
this.XGoa = this.airInOA.getHumi();

//外気湿球温度を設定
this.WBoa = Psychrometrics.FNWbtx(this.DBoa, this.XGoa);
if( this.WBoa < -9000.0 ){
    this.WBoa = this.WB_cal(this.DBoa, this.XGoa);
}


//mState=this.swc*this.mod;
//
if( Airswc.isOFF( this.swcIn )){
    //停止
    this.mState = 0;
}else if( Airswc.isON( this.swcIn )){
    //運転
    if( Airmod.isCOOL( this.modIn ) && Airmod.isHEAT( this.modIn ) ){
        //冷暖同時
        if( this.isCOOLandHEATout ){
            this.mState = 11;
        }else{
            //冷暖同時機種で無い場合は冷房運転とする
            this.mState = 1;
            this.message.append( "(W)冷暖同時機種ではない→冷房運転で計算" );
```

```
            }
        }else if( Airmod.isCOOL( this.modIn )){
            //冷房運転
            if( Airmod.isC_CHARGE( this.modIn ) ){
                //蓄熱開始時刻　22時　の処理
                if( BestTimeManager.getDateWeatherTime()[5] == 22 && BestTimeManager.getDateWeatherTime()[6] == 0
                      && this.opeTimeCount >= 0 ){//20151116
                    this.message.append( "(W)蓄熱運転判断時刻" );
                    this.opeTimeAdd = this.opeTimeAddNext;

                    //蓄熱運転の判断
                    if( this.out_STQQ > 0 ){
                        this.message.append( "(C)残量=0" );
                        //蓄熱残量＝０
                        this.opeTimeMax = this.opeTimeCount + 1800 + this.opeTimeAdd;
                        this.opeTimeAddNext = 0;
                    }else if( this.out_STQQ > -this.out_STQQc_S * 0.1 ){
                        this.message.append( "(C)残量<10%" );
                        //蓄熱残量１０％未満
                        this.opeTimeMax = this.opeTimeCount + this.opeTimeAdd;
                        this.opeTimeAddNext = 0;
                    }else{
                        this.message.append( "(C)残量>=10%" );
                        //蓄熱残量１０％以上
                        this.opeTimeMax = this.opeTimeCount - 3600 * 2;
                        if( this.opeTimeMax >= 3600 ){
                            this.opeTimeAddNext = 3600 * 3;
                        }else{
                            this.opeTimeMax = 0;
                        }
                    }

                    this.opeTimeOld = this.opeTimeCount;
                    this.opeTimeCount = 0;
                    this.message.append( "(C)opeTimeCount_Max="+this.opeTimeOld+"_ "+this.opeTimeMax );
                }

                //冷房蓄熱運転
                if( this.DBoa >= 10. ){
                    //System.out.println( "out_STQQ " +out_STQQ + " / out_STQQc_S" + out_STQQc_S +" /opeTimeOld
"+opeTimeOld+" /opeTimeMax "+opeTimeMax);
                    if( this.out_STQQ > -this.out_STQQc_S && this.opeTimeCount < this.opeTimeMax ){
                        //冷房蓄熱が満蓄ではない &&　予定蓄熱運転時間に達していない
                        this.message.append( "(C)冷房蓄熱運転" );
                        this.mState = Airmod.COOL + Airmod.C_CHARGE;
                    }else{
                        this.message.append( "(C)満蓄or予定蓄熱運転すみ" );
                        this.mState = 0;
                    }
                }else{
                    //冷房蓄熱する外気温度条件ではない
                    this.message.append( "(W)冷房蓄熱運転の外気温度("+AirFormat.df_2(this.DBoa)+"<10℃)範囲外→停止" );
                    this.mState = 0;
                }

            }else{
                //冷房運転
                if( this.DBoa >= 10. &&  this.out_STQQ < 0 ){
                    //冷房蓄熱利用運転
                    this.mState = Airmod.COOL * 100;
                }else if( this.DBoa >= -5. ){
                    //冷房蓄熱非利用運転
                    this.mState = Airmod.COOL;
                }else{
                    //条件範囲外
                    this.mState = Airmod.VENTILATE;
                    this.message.append( "(W)外気温度が冷房運転条件範囲外→換気" );
                }
            }
        }else if( Airmod.isHEAT( this.modIn )){
            //暖房運転
            if( Airmod.isH_CHARGE( this.modIn )){
```

```
            //暖房蓄熱運転
            if( this.DBoa <= 11. ){
               if( this.out_STQQ < this.out_STQQh_S ){
                  //暖房蓄熱が満蓄ではない
                  this.message.append( "(W)暖房蓄熱運転" );
                  this.mState = Airmod.HEAT + Airmod.H_CHARGE;
               }else{
                  this.mState = 0;
               }
            }else{
               //暖房蓄熱する外気温度条件ではない
               this.message.append( "(W)暖房蓄熱運転の外気温度条件範囲外→停止" );
               this.mState = 0;
            }

         }else{
            //暖房運転
            if( this.DBoa >= -18. && this.DBoa <= 18. ){
               this.mState = Airmod.HEAT;
            }else{
               this.mState = Airmod.VENTILATE;
               this.message.append( "(W)外気温度が暖房運転条件範囲外→換気" );
            }
         }
      }else if( Airmod.isVENTILATE( this.modIn )){
         //換気モード
         this.mState = Airmod.VENTILATE;
      }else{
         //停止
         this.mState = 0;
      }
   }


   /**
    * 入力
    */

   this.rmlinemap
   = (Map<String,BM_EHPdata>) super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine ));

   this.CAPrateC = 0.;
   this.CAPrateH = 0.;

   this.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
   this.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
   this.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
   this.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
   this.Sc_Loadout = 0.;
   this.Sh_Loadout = 0.;
   this.Tc_Loadout = 0.;
   this.Th_Loadout = 0.;
   this.out_Qcc = 0.;
   this.out_Qhc = 0.;

   this.DBra_max = -100.0;
   this.WBra_min = 1000.0;
   this.Lratemin = 100.0;

   double dmy1 = 100.0;

   //室内機データの読込
   for( int i=0; i < this.num; i++ ){
      this.RMdata = this.rmlinemap.get( this.rmnm[i] );
      dmy1 = this.RMdata.get_L_rate();
      if( this.Lratemin > dmy1 ){
         this.Lratemin=dmy1;
      }
      dmy1 = this.RMdata.get_R_wb();
      if( this.WBra_min > dmy1 ){
         this.WBra_min=dmy1;
      }
```

```
        dmy1 = this.RMdata.get_R_db();
        if( this.DBra_max < dmy1 ){
           this.DBra_max=dmy1;
        }
        this.allSc_heat += this.RMdata.get_R_Sc_load();//室内機処理熱量（冷却顕熱）[W]
        this.allTc_heat += this.RMdata.get_R_Tc_load();//室内機処理熱量（冷却全熱）[W]
        //
        this.allSh_heat += this.RMdata.get_R_Sh_load();//室内機処理熱量（加熱顕熱）[W]
        this.allTh_heat += this.RMdata.get_R_Th_load();//室内機処理熱量（加熱全熱）[W]

        // System.out.println( "this.RMdata.get_R_Tc_load()" + this.RMdata.get_R_Tc_load() +
"this.RMdata.get_R_Th_load()" + this.RMdata.get_R_Th_load());
      }

      this.WBra = this.WBra_min;
      this.DBra = this.DBra_max;



      //*************************----------------
      if( this.isAdjust2012 ){
         double rc;
         double c_OutTemp;
         double rh;
         double h_OutTemp;

         //処理可能熱量　仮計算
         if( this.allTc_heat==0 && this.allTh_heat==0 ){
            //冷却・加熱要求なし
            c_OutTemp = this.out_Qc_S_1;
            h_OutTemp = this.out_Qh_S_1;
         }else if( this.allTh_heat == 0 ){
            //冷却要求のみ coolingOnly_cal()
            c_OutTemp = this.pacDB.getKcta( this.DBoa) * this.pacDB.getKcti( this.WBra) * this.Lratemin *
this.out_Qc_S_1;//仮計算　室外機処理可能熱量（冷却全熱）[W]
            h_OutTemp = this.out_Qh_S_1;
         }else if( this.allTc_heat == 0 ){
            //加熱要求のみ heatingOnly_cal()
            c_OutTemp = this.out_Qc_S_1;
            h_OutTemp = this.pacDB.getKhta( this.WBoa) * this.pacDB.getKhti( this.DBra) * this.Lratemin *
this.out_Qh_S_1;//仮計算　室外機処理可能熱量（加熱全熱）[W]
         }else if( this.allTc_heat >= this.allTh_heat ){
            //冷却主体運転 cooling_and_heating_cal()
            c_OutTemp = this.pacDB.getKcmta( this.DBoa) * this.pacDB.getKcti( this.WBra) * this.Lratemin *
this.out_Qc_S_1;//仮計算　室外機処理可能熱量（冷却全熱）[W]
            h_OutTemp = this.pacDB.getKhti( this.DBra ) * this.Lratemin * this.out_Qh_S_1;//室外機処理可能熱量（加熱全
熱）[W]
         }else{
            //加熱主体運転 heating_and_cooling_cal()
            c_OutTemp = this.pacDB.getKcti( this.WBra) * this.Lratemin * out_Qc_S_1;//室外機処理可能熱量（冷却全熱）[W]

            h_OutTemp  = this.pacDB.getKhta( this.WBoa ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin *
out_Qh_S_1;//室外機処理可能熱量（加熱全熱）[W]
            double Rch = this.allTc_heat / h_OutTemp / this.out_rhAdjust;
            double deltaWB = this.pacDB.getKdwb( Rch );
            //補正暖房容量
            h_OutTemp  = this.pacDB.getKhmta( this.WBoa + deltaWB ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin *
out_Qh_S_1;//室外機処理可能熱量（加熱全熱）[W]
         }

         //冷却能力調整
         if( c_OutTemp == 0 ){
            //rc = this.allTc_heat / this.out_Qc_S;
            rc = this.allTc_heat / this.out_Qc_S_1;
         }else{
            rc = this.allTc_heat / c_OutTemp;
         }

         if( rc > 1.1 ){
         // rc = 1.1;
         }
```

```
        //加熱能力調整
        if( h_OutTemp == 0 ){
           //rh = this.allTh_heat / this.out_Qh_S;
           rh = this.allTh_heat / this.out_Qh_S_1;
        }else{
           rh = this.allTh_heat / h_OutTemp;
        }

        if( rh > 1.1 ){
        // rh = 1.1;
        }

        //*************************----------------
        if( true ){
           //移動平均の最大値で調整していく
           this.rcList.removeLast();
           this.rcList.addFirst( rc );
           //
           double sum_rc = 0;
           for( int i=0; i<this.numAdjustSteps; i++ ){
              sum_rc += this.rcList.get( i );
           }
           this.ave_rcAdjust = sum_rc / this.numAdjustSteps;
           //
           if( sum_rc > this.max_rcAdjust * this.numAdjustSteps ){
              this.max_rcAdjust = sum_rc / this.numAdjustSteps;

              this.out_rcAdjust = this.max_rcAdjust;
              //this.in_run_stop_Num = this.in_run_stop / this.out_rcAdjust;

              this.out_Qc_S   = this.out_Qc_S_1    * this.out_rcAdjust;
              this.out_Qc_C   = this.out_Qc_C_1    * this.out_rcAdjust;
              this.out_PPEc_S = this.out_PPEc_S_1  * this.out_rcAdjust;
              this.out_PPEc_C = this.out_PPEc_C_1  * this.out_rcAdjust;

              this.out_QSc_S = this.out_QSc_S_1   * this.out_rhAdjust;//= "定格蓄熱利用冷房能力";//kW
              this.out_QSc_C = this.out_QSc_C_1   * this.out_rhAdjust;//= "中間蓄熱利用冷房能力";//kW
              this.out_PPESc_S = this.out_PPESc_S_1   * this.out_rhAdjust;//= "定格蓄熱利用冷房入力(電力)";//kW
              this.out_PPESc_C = this.out_PPESc_C_1   * this.out_rhAdjust;//= "中間蓄熱利用冷房入力(電力)";//kW
              //
              this.out_STQQc_S = this.out_STQQc_S_1   * this.out_rhAdjust;//= "定格冷房蓄熱量[J]";

              //定格冷房蓄熱能力[W]
              this.out_Qcc_S = this.out_STQQc_S / 10. / 3600.;//蓄熱時間10時間に固定

//            this.out_COP_STc_S = this.out_COP_STc_S_1   * this.out_rhAdjust;//= "定格冷房蓄熱エネルギー消費効率[-]";
//            this.out_COP_STc_H = this.out_COP_STc_H_1   * this.out_rhAdjust;//= "高温時冷房蓄熱エネルギー消費効率[-]";

              this.out_ST_M_W = this.out_ST_M_W_1   * this.out_rhAdjust;//= "蓄熱槽水量[g]";
//            this.out_ST_Lh = this.out_ST_Lh_1   * this.out_rhAdjust;//= "蓄熱槽外形寸法高さ[m]";
//            this.out_ST_LW = this.out_ST_LW_1   * this.out_rhAdjust;//= "蓄熱槽外形寸法幅  [m]";
              this.out_ST_LD = this.out_ST_LD_1   * this.out_rhAdjust;//= "蓄熱槽外形寸法奥行[m]";

              this.out_ST_SA = ( this.out_ST_LD * this.out_ST_Lh + this.out_ST_LD * this.out_ST_LW + this.out_ST_Lh *
this.out_ST_LW ) * 2;

              this.fRate_airEAc = this.fRate_airEA * this.out_rcAdjust;
           }
        }


        if( true ){
           //移動平均の最大値で調整していく
           this.rhList.removeLast();
           this.rhList.addFirst( rh );
           //
           double sum_rh = 0;
           for( int i=0; i<this.numAdjustSteps; i++ ){
              sum_rh += this.rhList.get( i );
           }
           this.ave_rhAdjust = sum_rh / this.numAdjustSteps;
           //
```

```
            if( sum_rh > this.max_rhAdjust * this.numAdjustSteps ){
               this.max_rhAdjust = sum_rh / this.numAdjustSteps;

               this.out_rhAdjust = this.max_rhAdjust;
               //this.in_run_stop_Num = this.in_run_stop / this.out_rcAdjust;

               this.out_Qh_S   = this.out_Qh_S_1   * this.out_rhAdjust;
               this.out_Qh_C   = this.out_Qh_C_1   * this.out_rhAdjust;
//             this.out_Qh_L   = this.out_Qh_L_1   * this.out_rhAdjust;
               this.out_PPEh_S = this.out_PPEh_S_1 * this.out_rhAdjust;
               this.out_PPEh_C = this.out_PPEh_C_1 * this.out_rhAdjust;
//             this.out_PPEh_L = this.out_PPEh_L_1 * this.out_rhAdjust;

               this.out_STQQh_S = this.out_STQQh_S_1   * this.out_rhAdjust;//= "定格暖房蓄熱量[J]";

               //定格暖房蓄熱能力[W]
               this.out_Qhc_S = this.out_STQQh_S / 2.  / 3600.;//蓄熱時間2時間に固定
               this.out_STQQh_defrost_Co = this.out_STQQh_S / 14.;//暖房デフロスト１回の暖房蓄熱消費量[Ws]

               this.out_PEESTh_S = this.out_PEESTh_S_1   * this.out_rhAdjust;//= "定格暖房蓄熱消費電力量[Ws]";

               this.out_ST_M_W = this.out_ST_M_W_1   * this.out_rhAdjust;//= "蓄熱槽水量[g]";
//             this.out_ST_Lh = this.out_ST_Lh_1   * this.out_rhAdjust;//= "蓄熱槽外形寸法高さ[m]";
//             this.out_ST_LW = this.out_ST_LW_1   * this.out_rhAdjust;//= "蓄熱槽外形寸法幅  [m]";
               this.out_ST_LD = this.out_ST_LD_1   * this.out_rhAdjust;//= "蓄熱槽外形寸法奥行[m]";

               this.out_ST_SA = ( this.out_ST_LD * this.out_ST_Lh + this.out_ST_LD * this.out_ST_LW + this.out_ST_Lh *
this.out_ST_LW ) * 2;

               this.fRate_airEAh = this.fRate_airEA* this.out_rhAdjust;
            }
         }
         //*************************----------------


         //調整補正
/*       if( rc > 1.0 ){
            this.out_Qc_S *= rc;
            this.out_Qc_C *= rc;
            this.out_PPEc_S *= rc;
            this.out_PPEc_C *= rc;
            //
            this.fRate_airEAc *= rc;
         }
*/
         //調整補正
/*       if( rh > 1.0 ){
            this.out_Qh_S *= rh;
            this.out_Qh_C *= rh;
            this.out_Qh_L *= rh;
            this.out_PPEh_S *= rh;
            this.out_PPEh_C *= rh;
            this.out_PPEh_L *= rh;
            //
            this.fRate_airEAh *= rh;
         }
*/
         this.out_Qc_Adjust = this.out_Qc_S;
         this.out_Qh_Adjust = this.out_Qh_S;

         //System.out.println( "BM out  out_Qc_Adjust=" +out_Qc_Adjust+"  out_Qh_Adjust="+out_Qh_Adjust);
      }


      //*************************----------------


      switch( this.mState ){
      case Airmod.VENTILATE:
         //昼間運転の換気モード　室外機は停止
         //cse 0　と同じ
```

```
case 0:
   //停止
   this.message.append("(C)停止");

   this.calc_Stop();
   break;
case Airmod.COOL:
   //冷房
   if( this.out_Qc_S_1 > 0 ){
      this.message.append("(C)冷房蓄熱非利用運転");
      this.coolingOnly_cal();
   }else{
      this.message.append("(C)冷能力=0停止");//20130625
      this.calc_Stop();
   }

   break;
case (Airmod.COOL * 100):
   //冷房
   if( this.out_QSc_S_1 > 0 ){
      this.message.append("(C)冷房蓄熱利用運転");
      this.coolingOnly_useSTIce_cal();
   }else{
      this.message.append("(C)蓄熱利用冷能力=0停止");//20130625
      this.calc_Stop();
   }

   break;
case (Airmod.C_CHARGE + Airmod.COOL):
   //冷房　蓄熱
   if( this.out_STQQc_S_1 > 0 ){
      this.message.append("(C)冷房蓄熱運転");
      this.cool_ChargingIce_cal();
      this.opeTimeCount += this.tInterval;
   }else{
      this.message.append("(C)蓄熱槽=0停止");//20130625
      this.calc_Stop();
   }

   break;
case Airmod.HEAT:
   //暖房
   if( this.out_Qh_S_1 > 0 ){
      this.message.append("(C)暖房運転");
      this.heatingOnly_cal();
   }else{
      this.message.append("(C)暖能力=0停止");//20130625
      this.calc_Stop();
   }

   break;
case (Airmod.HEAT + Airmod.H_CHARGE):
   //暖房　蓄熱
   if( this.out_STQQh_S_1 > 0 ){
      this.message.append("(C)暖房蓄熱運転");
      this.heat_Charging_cal();
   }else{
      this.message.append("(C)蓄熱槽=0停止");//20130625
      this.calc_Stop();
   }

   break;
case 11:
   //冷暖同時
   this.message.append("(C)冷暖房運転");
   if( this.allTh_heat == 0 && this.out_Qc_S_1 > 0 ){
      //冷房運転
      this.coolingOnly_cal();

   }else if( this.allTc_heat == 0 && this.out_Qh_S_1 > 0 ){
      //暖房運転
      this.heatingOnly_cal();
```

```
        }else if( this.allTc_heat > this.allTh_heat && this.out_Qc_S_1 > 0 ){
            //冷房主体運転
            this.cooling_and_heating_cal();

        }else if( this.allTh_heat > this.allTc_heat && this.out_Qh_S_1 > 0 ){
            //暖房主体運転
            this.heating_and_cooling_cal();
        }else{
            this.message.append("(C)能力=0停止");//20130625
            this.calc_Stop();
        }

        break;

    default:
        this.message.append("(C)運転モード？停止");//20130625
        this.calc_Stop();
        //System.out.println(this.moduleName + ">>Error<< onOff*modeが範囲外");
    }

    this.E_PPE = this.PPE * Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 );
    this.eleIn.setActivePower( this.PPE );
    this.eleIn.setReactivePower( this.E_PPE );
    this.eleIn.setPhase( this.phase );
    this.eleIn.setVoltage( this.voltage );
    this.eleIn.setFrequency( this.frequency );

    //出力設定
    for(int i=0;i<num;i++){

        //System.out.println( "(C)num="+i+"/"+num+" rmlinemap.get"+this.rmlinemap.get( this.rmnm[i] )  );

        this.RMdata = this.rmlinemap.get( this.rmnm[i] );
        this.RMdata.set_CAPrateC( this.CAPrateC );
        this.RMdata.set_CAPrateH( this.CAPrateH );
        this.rmlinemap.put( this.rmnm[i], this.RMdata );
    }
    super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );

    super.sm.setState( super.getConnectionNode( this.S_NODE_airOutEA ), this.airOutEA );

    //グラフデータ追加
    if( this.isGVisible ){
        this.gBMout.addData( BestTimeManager.getDateWeatherTime(),
        this.Sc_Loadout,//室外機処理可能熱量（冷却）[W]
        this.Sh_Loadout,//室外機処理可能熱量（加熱）[W]
        this.allTc_heat,//室内機処理要求熱量（冷却）[W]
        this.allTh_heat,//室内機処理要求熱量（加熱）[W]
        this.PPE,
        this.DBoa,
        this.DBra,
        this.XGoa,
        this.XGra,
        this.CAPrateC.doubleValue() );
    }

    //記録ノード
    if( this.isRecord && super.rm != null ){
        this.record();
    }

    this.message.setLength(0);
}


private void calc_Stop(){
    //蓄熱量
    this.out_STQQ += this.st_heatLoss_calc();

    this.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
    this.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
```

```
        this.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
        this.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.PPE = 0.0;
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );

    }
    /**
     * 記録
     */
    private void record(){
        //記録ノード
        if( super.rm != null ){
        if( CheckPrintModule.isPrintMessage ){
            //message
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_message, this.name, message.toString() );
        }

        if( CheckPrintModule.isPrintEnergy ){
            //記録ノードに室外機消費電力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID4, this.name, AirFormat.df_2(this.PPE));
        }

        if( CheckPrintModule.isPrintLoad ){
            //記録ノードに室外機実処理熱量（全熱）を設定
            if( this.Tc_Loadout > 0 ){
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT, this.name, AirFormat.df_2(-
this.Tc_Loadout));//室外機冷却実処理熱量/現状#W#熱量
            }else{
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT, this.name,
AirFormat.df_2(this.Th_Loadout));//室外機加熱実処理熱量/現状#W#熱量
            }
        }

        if( CheckPrintModule.isPrintStateOut ){
        }

        if( CheckPrintModule.isPrintStateMy ){
            //記録ノードに室内機加熱能力補正比率を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID3c, this.name,
AirFormat.df_2(this.CAPrateC));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID3h, this.name,
AirFormat.df_2(this.CAPrateH));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_stQQ, this.name,
AirFormat.df_2(this.out_STQQ));
            //記録ノードに室内機要求熱量（顕熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID1c, this.name,
AirFormat.df_2(this.allSc_heat));//室内機冷却要求顕熱量合計#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID1h, this.name,
AirFormat.df_2(this.allSh_heat));//室内機加熱要求顕熱量合計#W#熱量
            //記録ノードに室内機要求熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID2c, this.name,
AirFormat.df_2(this.allTc_heat));//室内機冷却要求全熱量合計#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID2h, this.name,
AirFormat.df_2(this.allTh_heat));//室内機加熱要求全熱量合計#W#熱量
            //記録ノードに室外機処理可能熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID5c, this.name,
AirFormat.df_2(this.Sc_Loadout));//室外機冷却可能熱量/最大#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID5h, this.name,
AirFormat.df_2(this.Sh_Loadout));//室外機加熱可能熱量/最大#W#熱量
            //記録ノードに室外機実処理熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID6c, this.name,
AirFormat.df_2(this.Tc_Loadout));//室外機冷却実処理熱量/現状#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID6h, this.name,
AirFormat.df_2(this.Th_Loadout));//室外機加熱実処理熱量/現状#W#熱量
```

```
            //
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_QstC, this.name,
AirFormat.df_2(this.out_Qcc));//室外機冷却蓄熱#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_QstH, this.name,
AirFormat.df_2(this.out_Qhc));//室外機加熱蓄熱#W#熱量
        }

        if( CheckPrintModule.isPrintStateIn ){
            //記録ノードに室外機外気入口DBを設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBairInHS, this.name,
AirFormat.df_2(this.airInOA.getTempDB()) );
            //記録ノードに室外機外気入口WBを設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_WBairInHS, this.name,
AirFormat.df_2(this.airInOA.getTempWB()) );
            //記録ノードに室外機外気入口流量を設定
            //super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_M_airInHS, this.name,
AirFormat.df_2(this.airInOA.getFlowRate()) );
        }

        if( CheckPrintModule.isPrintAdjust ){
            //
            if( this.isAdjust2012 ){
                //記録ノードに室外機調整能力を設定
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qc_Adjust, this.name,
this.out_Qc_Adjust );//室外機調整能力[W]
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qh_Adjust, this.name,
this.out_Qh_Adjust );//室外機調整能力[W]
                //記録ノードに室外機調整能力を設定
                //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rcAdjust", this.name, this.out_rcAdjust );//
室外機調整率[-]
                //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rhAdjust", this.name, this.out_rhAdjust );//
室外機調整率[-]
            }
        }

        }
    }

    /**
     * ビルマルチ計算/各室内機の集計後
     */

    @Override
    public void update() {
        double rCode0 = 1.;
        double rCode1 = 1.;

        //能力補正係数のうち室外機側分を求める
        //機種によっては次のステップで室内機側の能力設定に使用することがある

        switch( this.mState ){
        case (Airmod.COOL * 100):
            //cooling useSTIce
            rCode1 = this.pacDB.getKcsta( this.DBoa);
            break;
        case Airmod.COOL:
            //冷房
            //CoolingOnly
            rCode1 = this.pacDB.getKcta( this.DBoa);
            break;
        }
        for( int i=0; i<this.num; i++ ){//20110623nino
            this.RMdata = this.rmlinemap.get( this.rmnm[i] );
            this.RMdata.set_out_c_CAPrevision( rCode0 * rCode1 );
        }
        //this.RMdata.set_out_c_CAPrevision( rCode0 * rCode1 );

        //Heating_only
        rCode1 = this.pacDB.getKhta( this.WBoa);
        for( int i=0; i<this.num; i++ ){//20110623nino
            this.RMdata = this.rmlinemap.get( this.rmnm[i] );
            this.RMdata.set_out_h_CAPrevision( rCode0 * rCode1 );
```

```
    }
    //  this.RMdata.set_out_h_CAPrevision( rCode0 * rCode1 );

}

public Object viewInternal(TestCommand cmd) {

    ArrayList<Object> result = new ArrayList<Object>();
    result.add(super.sm);
    result.add(super.cm);
    result.add(super.rm);
    //外部定義
    result.add(this.name);

    result.add(this.out_Qc_S);
    result.add(this.out_Qc_C);
    result.add(this.out_PPEc_S);
    result.add(this.out_PPEc_C);
    result.add(this.out_Qh_S);
    result.add(this.out_Qh_C);
    //result.add(this.out_Qh_L);
    result.add(this.out_PPEh_S);
    result.add(this.out_PPEh_C);
    //result.add(this.out_PPEh_L);

    result.add(this.Lp);
    result.add(this.Lh);

    result.add(this.DBoa);
    result.add(this.XGoa);
    result.add(this.DBra);
    result.add(this.XGra);

    return result;
}


/**
 * 蓄熱槽からの熱損失
 */
private double st_heatLoss_calc(){
    //停止時のみ計算する
    //冷房放熱特性＝放熱量×表面積×放熱時間×（外気温度 - 0）
    //暖房放熱特性＝放熱量×表面積×放熱時間×（蓄熱量/蓄熱水量/4.18605 - 外気温度）
    //暖房蓄熱時水温上限温度＝19.5[℃]
    //基本放熱量＝2.5[ｋJ/(h･m2･℃)]＝2500/3600[J/s/(m2･℃)]＝2500/3600[W/(m2･℃)]

    double heatLoss = 0;
    double heatLossU = 2500./3600.;//基本放熱量[W/(m2･℃)]
    double stWTemp;

    //水温（0℃基準）＝蓄熱量 / 蓄熱水量 / 4.18605
    stWTemp = this.out_STQQ / this.out_ST_M_W / 4.18605;

    if( stWTemp > 0 ){
        heatLoss = heatLossU * this.out_ST_SA * this.tInterval * ( this.DBoa - stWTemp );
    }else{
        heatLoss = heatLossU * this.out_ST_SA * this.tInterval * ( this.DBoa - 0 );
    }

        //System.out.println( "(st_heatLoss_calc)氷蓄熱槽  水温＝" + stWTemp + " /heatLoss="+ heatLoss);

    return heatLoss;
}


/**
 * 蓄熱槽からの熱損失
 * 冷房
 */
private void cooling_STheatLoss_calc(){
    //停止時のみ計算する
```

```
    //放熱量＝2.5[ｋJ/(h･m2･℃)」

}


/**
 * 暖房 蓄熱運転
 */
private void heat_Charging_cal(){
    double rCode0,rCode1,rCode2,rCode3;
    double D_fmc,D_pmc,alph,beta;

    D_fmc    = this.fmc;
    D_pmc    = this.pmc;


    //上下限のチェック 20120821nino
    double dbOA = this.DBoa;
    //double dbRA = this.DBra;

    if( dbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhcta() ) ){
        //外気DB温度＞上限の時　上限値の特性で運転
        this.message.append( "(C)外気DB＞上限→上限値で特性計算");
        dbOA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhcta() );
    }
    if( dbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhcta() ) ){
        //外気DB温度＜下限の時　停止
        this.message.append( "(C)外気DB＜下限→停止");
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.PPE = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );

        this.out_Qhc = 0;
        return;
    }
    //if( dbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() ) ){
    // //室内乾球温度＞上限の時　上限値の特性で運転
    // this.message.append( "(C)室DB＞上限→上限値で特性計算");
    // dbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
    //}
    //if( dbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() ) ){
    // //室内乾球温度＜下限の時　下限値の特性で運転
    // this.message.append( "(C)室DB＜下限→下限値で特性計算");
    // dbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
    //}

    //室温補正
    //能力補正

    //暖房蓄熱能力
    //暖房蓄熱能力＝定格補正後暖房蓄熱能力×能力補正（Ｉ）
    //能力補正（Ｉ）：外気補正
    rCode1 = this.pacDB.getKhcta( dbOA );//外気補正

    this.out_Qhc = rCode1 * this.out_Qhc_S;//暖房蓄熱能力[W]


    //暖房蓄熱消費電力
    rCode2 = this.pacDB.getKhcwta( dbOA );

    //this.PPE = rCode2 * this.out_PPEh_S;//////////////////////////////////
    this.PPE = rCode2 * this.out_PEESTh_S / 3600 / 2.;

    //蓄熱除霜利用量
    //this.//////////////////////
```

```
    //airOutEA
    this.airOutEA.setFlowRate( this.fRate_airEAh );
    this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPE ) / this.fRate_airEAh + this.airInOA.getTempDB() );


    //蓄熱量
    this.out_STQQ += this.out_Qhc * this.tInterval;



    //if( this.Rc < 0.0001 || this.Rc < this.in_run_stop){
    // this.CAPrateC = 0.0;
    // this.CAPrateH = 0.0;
    // this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
    // this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
    // this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
    // this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
    // this.PPE = 0.0;
    // this.E_PPE = 0.0;
    // this.airOutEA.setFlowRate( 0. );
    // this.airOutEA.setTempDB( this.airInOA.getTempDB() );
    //}


    //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

}

/**
 * 冷房　氷蓄熱運転
 */
private void cool_ChargingIce_cal(){
    double rCode0,rCode1,rCode2,rCode3;
    double D_fmc,D_pmc,alph,beta;

    D_fmc    = this.fmc;
    D_pmc    = this.pmc;

    //上下限のチェック 20120821nino
    double dbOA = this.DBoa;
    //double wbRA = this.WBra;

    if( dbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKccta() ) ){
        //外気乾球温度＞上限の時停止
        this.message.append( "(C)外気DB＞上限→停止");
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.PPE = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );
        return;
    }
    if( dbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKccta() ) ){
        //外気乾球温度＜下限の時　下限値の特性で運転
        this.message.append( "(C)外気DB＜下限→下限値で特性計算");
        dbOA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKccta() );
    }
    //if( wbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() ) ){
    // //室内湿球温度＞上限の時　上限値の特性で運転
    // this.message.append( "(C)室WB＞上限→上限値で特性計算");
    // wbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
    //}
    //if( wbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() ) ){
    // //室内湿球温度＜下限の時　下限値の特性で運転
    // this.message.append( "(C)室WB＜下限→下限値で特性計算");
    // wbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
    //}
```

```
    //室温補正
    //能力補正

    //冷房蓄熱能力
    rCode1 = this.pacDB.getKccta( dbOA );//外気補正

    this.out_Qcc = rCode1 * this.out_Qcc_S;//冷房蓄熱能力[W]


    //冷房蓄熱消費電力
    rCode2 = this.pacDB.getKccwta( dbOA );

    rCode3 = this.out_COP_STc_S;////////////////////////////////////////////////////////////////////////////////
    if( rCode3 <= 0 ){
       rCode3 = 0.1;
    }

    this.PPE = rCode2 / rCode3 * this.out_Qcc;

    //airOutEA
    this.airOutEA.setFlowRate( this.fRate_airEAc );
    this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPE ) / this.fRate_airEAc + this.airInOA.getTempDB() );


    //蓄熱量
    this.out_STQQ -= this.out_Qcc * this.tInterval;



    //if( this.Rc < 0.0001 || this.Rc < this.in_run_stop){
    // this.CAPrateC = 0.0;
    // this.CAPrateH = 0.0;
    // this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
    // this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
    // this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
    // this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
    // this.PPE = 0.0;
    // this.E_PPE = 0.0;
    // this.airOutEA.setFlowRate( 0. );
    // this.airOutEA.setTempDB( this.airInOA.getTempDB() );
    //}


    //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

}



/**
 * 全　冷房運転
 * 蓄熱　利用運転
 */
private void coolingOnly_useSTIce_cal(){
    double rCode0,rCode1,rCode2,rCode3;
    double D_fmc,D_pmc,alph,beta;
    double Rp;

    D_fmc    = this.fmSc;
    D_pmc    = this.pmSc;

    //上下限のチェック 20120821nino
    double dbOA = this.DBoa;
    double wbRA = this.WBra;

    if( dbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcsta() ) ){
       //外気乾球温度＞上限の時停止
       this.message.append( "(C)外気DB＞上限→停止");
       this.CAPrateC = 0.0;
       this.CAPrateH = 0.0;
       this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
```

```
    this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
    this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
    this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
    this.PPE = 0.0;
    this.airOutEA.setFlowRate( 0. );
    this.airOutEA.setTempDB( this.airInOA.getTempDB() );

    this.out_STQQc_Co = 0;
    return;
}
if( dbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcsta() ) ){
    //外気乾球温度＜下限の時　下限値の特性で運転
    this.message.append( "(C)外気DB＜下限→下限値で特性計算");
    dbOA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcsta() );
}
if( wbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcsti() ) ){
    //室内湿球温度＞上限の時　上限値の特性で運転
    this.message.append( "(C)室WB＞上限→上限値で特性計算");
    wbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcsti() );
}
if( wbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcsti() ) ){
    //室内湿球温度＜下限の時　下限値の特性で運転
    this.message.append( "(C)室WB＜下限→下限値で特性計算");
    wbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcsti() );
}

//室温補正
//能力補正

rCode1 = this.pacDB.getKcsta( dbOA );
rCode2 = this.pacDB.getKcsti( wbRA );
rCode3 = this.Lratemin;

this.Sc_Loadout = rCode1 * rCode2 * rCode3 * this.out_QSc_S;//室外機処理可能熱量（冷却全熱）[W]

//       System.out.println("Tc_Loadout="+Tc_Loadout+" Lratemin"+Lratemin+" / allTc_heat="+this.allTc_heat );

this.Rc = this.allTc_heat / this.Sc_Loadout;
this.CAPrateC = 1.0;
this.CAPrateH = 0.0;
this.Tc_Loadout =  this.allTc_heat;//20120731
if( this.Rc > 1.0 ) {
    this.CAPrateC = 1.0/ this.Rc;
    this.Rc = 1.0;
    this.Tc_Loadout = this.Sc_Loadout;//210120731
}

//低負荷領域の計算仕分け:Rh
if( this.Rc < this.in_run_stop ){
    switch( this.num_calcTypeLowerRangeLoad){
    case 0://  0_発停運転
        Rp = 0;
        break;

    case 1://  1_下限入力値固定
    case 2://  2_下限COP値固定
    case 3://  3_下限入力値と中間切片
        Rp = this.in_run_stop;
        break;

    default:
        Rp = 0;
    }
}else{
    Rp = this.Rc;
}

alph = this.pacDB.getAlphac( Rp );

beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc );

if( D_pmc * D_fmc < 0.001 ){
```

```
            beta = 1.0;
        }

        // System.out.println("Rc="+Rc+" /D_fmc="+D_fmc+" /D_pmc="+D_pmc+" /alph= "+alph +" /beat= "+beta);

        rCode0 = alph * beta;
        rCode1 = this.pacDB.getKcshpid( Rp );
        rCode2 = this.pacDB.getKcswta( dbOA );

        //       System.out.println("WBra="+WBra);

        rCode3 = this.pacDB.getKcswti( wbRA );

        this.PPE = rCode0 * rCode1 * rCode2 * rCode3 * this.out_PPESc_S;
        //低負荷領域の計算仕分け:PPE
        if( this.Rc < this.in_run_stop && Rp > 0 ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                this.PPE = 0.0;
                break;

            case 1://  1_下限入力値固定
                //何もしない
                break;

            case 2://  2_下限COP値固定
                this.PPE *= ( this.Rc / Rp );
                break;

            case 3://  3_下限入力値と中間切片
                this.PPE *= ( 0.5 + this.Rc / Rp / 2. );
                break;

            default:
            }
        }else{
            //何もしない
        }


        //airOutEA
        this.airOutEA.setFlowRate( this.fRate_airEAc );
        this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPE ) / this.fRate_airEAc + this.airInOA.getTempDB() );


        //氷蓄熱槽　の熱量関連の計算
        //冷房蓄熱消費量＝蓄熱利用冷房能力×冷房蓄熱消費量補正
        //冷房蓄熱消費量補正＝外気温度補正係数×冷房蓄熱消費量係数
        //冷房蓄熱消費量係数＝定格冷房蓄熱量/蓄熱利用時間（10h）/定格蓄熱利用冷房能力
        this.out_STQQc_Co
        = this.Tc_Loadout * this.pacDB.getKcsjta( dbOA ) * this.out_STQQc_S / 36000. / this.out_QSc_S * this.tInterval;  //
冷房蓄熱消費量（Ws）

        //蓄熱量
        this.out_STQQ += this.out_STQQc_Co;

        if( this.Rc < 0.0001 || this.Rc < this.in_run_stop){
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPE = 0.0;
            this.airOutEA.setFlowRate( 0. );
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );
        }else if( this.Rc < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                this.CAPrateC = 0.0;
                this.CAPrateH = 0.0;
                this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
```

```
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPE = 0.0;
            this.airOutEA.setFlowRate( 0. );
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );
            break;

         case 1://  1_下限入力値固定
            break;

         case 2://  2_下限COP値固定
            break;

         case 3://  3_下限入力値と中間切片
            break;

         default:
         }
      }



      //       System.out.println(" PPE="+PPE+" /Rc="+Rc+" /rCode1="+rCode1+" /rCode2="+rCode2+" /rCode3="+rCode3);


   }


   /**
    * 全　冷房運転
    * 蓄熱　非利用運転
    */
   private void coolingOnly_cal(){
      double rCode0,rCode1,rCode2,rCode3;
      double D_fmc,D_pmc,alph,beta;
      double Rp;

      D_fmc    = this.fmc;
      D_pmc    = this.pmc;

      //上下限のチェック 20120821nino
      double dbOA = this.DBoa;
      double wbRA = this.WBra;

      if( dbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcta() ) ){
         //外気乾球温度＞上限の時停止
         this.message.append( "(C)外気DB＞上限→停止");
         this.CAPrateC = 0.0;
         this.CAPrateH = 0.0;
         this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
         this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
         this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
         this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
         this.PPE = 0.0;
         this.airOutEA.setFlowRate( 0. );
         this.airOutEA.setTempDB( this.airInOA.getTempDB() );

         //蓄熱量
         this.out_STQQ += this.st_heatLoss_calc();

         return;
      }
      if( dbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcta() ) ){
         //外気乾球温度＜下限の時　下限値の特性で運転
         this.message.append( "(C)外気DB＜下限→下限値で特性計算");
         dbOA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcta() );
      }
      if( wbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() ) ){
         //室内湿球温度＞上限の時　上限値の特性で運転
         this.message.append( "(C)室WB＞上限→上限値で特性計算");
         wbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
```

```
}
if( wbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() ) ){
   //室内湿球温度＜下限の時　下限値の特性で運転
   this.message.append( "(C)室WB＜下限→下限値で特性計算");
   wbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
}

//室温補正
//能力補正

rCode1 = this.pacDB.getKcta( dbOA );
rCode2 = this.pacDB.getKcti( wbRA );
rCode3 = this.Lratemin;

this.Sc_Loadout = rCode1 * rCode2 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

//       System.out.println("Tc_Loadout="+Tc_Loadout+" Lratemin"+Lratemin+" / allTc_heat="+this.allTc_heat );

this.Rc = this.allTc_heat / this.Sc_Loadout;
this.CAPrateC = 1.0;
this.CAPrateH = 0.0;
this.Tc_Loadout = this.allTc_heat;//20120731
if( this.Rc > 1.0 ) {
   this.CAPrateC = 1.0/ this.Rc;
   this.Rc = 1.0;
   this.Tc_Loadout = this.Sc_Loadout;//20120731
}

//低負荷領域の計算仕分け:Rh
if( this.Rc < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = this.Rc;
}

alph = this.pacDB.getAlphac( Rp );

//       System.out.println("Rc="+Rc+" "+D_fmc+" "+D_pmc);

beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc );

if( D_pmc * D_fmc < 0.001 ){
   beta = 1.0;
}

rCode0 = alph * beta;
rCode1 = this.pacDB.getKchpid( Rp );
rCode2 = this.pacDB.getKcwta( dbOA );

//       System.out.println("WBra="+WBra);

rCode3 = this.pacDB.getKcwti( wbRA );

this.PPE = rCode0 *rCode1 * rCode2 * rCode3 * this.out_PPEc_S;
//低負荷領域の計算仕分け:PPE
if( this.Rc < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.PPE = 0.0;
```

```
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      this.PPE *= ( this.Rc / Rp );
      break;

   case 3://  3_下限入力値と中間切片
      this.PPE *= ( 0.5 + this.Rc / Rp / 2. );
      break;

   default:
   }
}else{
   //何もしない
}


//airOutEA
this.airOutEA.setFlowRate( this.fRate_airEAc );
this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPE ) / this.fRate_airEAc + this.airInOA.getTempDB() );

//蓄熱量
this.out_STQQ += this.st_heatLoss_calc();


//氷蓄熱槽　の熱量関連の計算
this.out_STQQc_Co = 0;  //冷房蓄熱消費量（Ws）

if( this.Rc < 0.0001 || this.Rc < this.in_run_stop){
   this.CAPrateC = 0.0;
   this.CAPrateH = 0.0;
   this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
   this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
   this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
   this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
   this.PPE = 0.0;
   this.airOutEA.setFlowRate( 0. );
   this.airOutEA.setTempDB( this.airInOA.getTempDB() );
}else if( this.Rc < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.CAPrateC = 0.0;
      this.CAPrateH = 0.0;
      this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
      this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
      this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
      this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
      this.PPE = 0.0;
      this.airOutEA.setFlowRate( 0. );
      this.airOutEA.setTempDB( this.airInOA.getTempDB() );
      break;

   case 1://  1_下限入力値固定
      break;

   case 2://  2_下限COP値固定
      break;

   case 3://  3_下限入力値と中間切片
      break;

   default:
   }

}


//       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
```

```
}


/**
 * 冷暖　同時運転　冷房主体
 */
private void cooling_and_heating_cal(){
   double rCode0,rCode1,rCode2,rCode3;
   double D_fmc,D_pmc,alph,beta;
   double Rp;

   D_fmc    = this.fmc;
   D_pmc    = this.pmc;

   //上下限のチェック 20120821nino
   double dbOA = this.DBoa;
   double wbRA = this.WBra;

   //冷房側
   if( dbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcmta() ) ){
      //外気乾球温度＞上限の時停止
      this.message.append( "(C)外気DB＞上限→停止");
      this.CAPrateC = 0.0;
      this.CAPrateH = 0.0;
      this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
      this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
      this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
      this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
      this.PPE = 0.0;
      this.airOutEA.setFlowRate( 0. );
      this.airOutEA.setTempDB( this.airInOA.getTempDB() );
      return;
   }
   if( dbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcmta() ) ){
      //外気乾球温度＜下限の時　下限値の特性で運転
      this.message.append( "(C)外気DB＜下限→下限値で特性計算");
      dbOA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcmta() );
   }
   if( wbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() ) ){
      //室内湿球温度＞上限の時　上限値の特性で運転
      this.message.append( "(C)室WB＞上限→上限値で特性計算");
      wbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
   }
   if( wbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() ) ){
      //室内湿球温度＜下限の時　下限値の特性で運転
      this.message.append( "(C)室WB＜下限→下限値で特性計算");
      wbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
   }

   //上下限のチェック 20120821nino
   //double wbOA = this.WBoa;
   double dbRA = this.DBra;

   //if( wbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() ) ){
   // //外気湿球温度＞上限の時　上限値の特性で運転
   // this.message.append( "(C)外気WB＞上限→上限値で特性計算");
   // wbOA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() );
   //}
   //if( wbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhta() ) ){
   // //外気湿球温度＜下限の時　停止
   // this.message.append( "(C)外気WB＜下限→停止");
   // this.CAPrateC = 0.0;
   // this.CAPrateH = 0.0;
   // this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
   // this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
   // this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
   // this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
   // this.PPE = 0.0;
   // this.E_PPE = 0.0;
   // this.airOutEA.setFlowRate( 0. );
   // this.airOutEA.setTempDB( this.airInOA.getTempDB() );
```

```
// return;
//}
if( dbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() ) ){
   //室内乾球温度＞上限の時　上限値の特性で運転
   this.message.append( "(C)室DB＞上限→上限値で特性計算");
   dbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
}
if( dbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() ) ){
   //室内乾球温度＜下限の時　下限値の特性で運転
   this.message.append( "(C)室DB＜下限→下限値で特性計算");
   dbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
}


//室温補正
//能力補正

//冷房側　処理熱量
rCode1 = this.pacDB.getKcmta( dbOA );//***
rCode2 = this.pacDB.getKcti( wbRA );
rCode3 = this.Lratemin;

this.Sc_Loadout = rCode1 * rCode2 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

//容量比
this.Rc = this.allTc_heat / this.Sc_Loadout;

this.CAPrateC = 1.0;
this.Tc_Loadout = this.allTc_heat;//20120731
if( this.Rc > 1.0 ) {
   this.CAPrateC = 1.0/ this.Rc;
   this.Rc = 1.0;
   this.Tc_Loadout = this.Sc_Loadout;//20120731
}

//低負荷領域の計算仕分け:Rh
if( this.Rc < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = this.Rc;
}

//消費電力
alph = this.pacDB.getAlphac( Rp );

//       System.out.println("Rc="+Rc+" "+D_fmc+" "+D_pmc);
beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc );

if( D_pmc * D_fmc < 0.001 ){
   beta = 1.0;
}

rCode0 = alph * beta;
rCode1 = this.pacDB.getKchpid( Rp );
rCode2 = this.pacDB.getKcmwta( dbOA );//***

//       System.out.println("WBra="+WBra);

rCode3 = this.pacDB.getKcwti( wbRA );
```

```
this.PPE = rCode0 *rCode1 * rCode2 * rCode3 * this.out_PPEc_S;
//低負荷領域の計算仕分け:PPE
if( this.Rc < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.PPE = 0.0;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      this.PPE *= ( this.Rc / Rp );
      break;

   case 3://  3_下限入力値と中間切片
      this.PPE *= ( 0.5 + this.Rc / Rp / 2. );
      break;

   default:
   }
}else{
   //何もしない
}



//暖房側  処理熱量
rCode2 = this.pacDB.getKhti( dbRA );
rCode3 = this.Lratemin;

this.Sh_Loadout = rCode2 * rCode3 * this.out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

//容量比
this.Rh = this.allTh_heat / this.Sh_Loadout;

this.CAPrateH = 1.0;
this.Th_Loadout = this.allTh_heat;//20120731
if( this.Rh > 1.0 ) {
   this.CAPrateH = 1.0/ this.Rh;
   this.Rh = 1.0;
   this.Th_Loadout = this.Sh_Loadout;//20120731
}


//airOutEA
this.airOutEA.setFlowRate( this.fRate_airEAc );
this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPE ) / this.fRate_airEAc + this.airInOA.getTempDB() );


if( this.Rc < 0.0001 ){
   this.CAPrateC = 0.0;
   this.CAPrateH = 0.0;
   this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
   this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
   this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
   this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
   this.PPE = 0.0;
   this.airOutEA.setFlowRate( 0. );
   this.airOutEA.setTempDB( this.airInOA.getTempDB() );
}else if( this.Rc < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.CAPrateC = 0.0;
      this.CAPrateH = 0.0;
      this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
      this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
      this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
      this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
      this.PPE = 0.0;
```

```
            this.airOutEA.setFlowRate( 0. );
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );
            break;

        case 1://  1_下限入力値固定
            break;

        case 2://  2_下限COP値固定
            break;

        case 3://  3_下限入力値と中間切片
            break;

        default:
        }

    }

    //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

}


/**
 * 全  暖房運転
 * 蓄熱  非利用
 */
private void heatingOnly_cal(){
    double rCode0,rCode1,rCode2,rCode3;
    double D_fmh,D_pmh,D_out_Qh_S,D_out_PPEh_S,alph,beta;
    double Rp;

    D_fmh    = this.fmh;
    D_pmh    = this.pmh;


    D_out_Qh_S  = this.out_Qh_S;
    D_out_PPEh_S= this.out_PPEh_S;

    //if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L > 0.0001) { //低温入力の場合
    //    D_fmh    = this.fmh;
    //    D_pmh    = this.pmh;
    //    D_out_Qh_S  = this.out_Qh_L;
    //    D_out_PPEh_S= this.out_PPEh_L;
    //}

    //上下限のチェック 20120821nino
    double wbOA = this.WBoa;
    double dbRA = this.DBra;

    if( wbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() ) ){
        //外気湿球温度＞上限の時  上限値の特性で運転
        this.message.append( "(C)外気WB＞上限→上限値で特性計算");
        wbOA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() );
    }
    if( wbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhta() ) ){
        //外気湿球温度＜下限の時  停止
        this.message.append( "(C)外気WB＜下限→停止");
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.PPE = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );

        this.out_STGh_T = 0;//暖房蓄熱除霜利用回数
        //蓄熱量
        this.out_STQQ +=  this.st_heatLoss_calc();
```

```
            return;
        }
        if( dbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() ) ){
            //室内乾球温度＞上限の時　上限値の特性で運転
            this.message.append( "(C)室DB＞上限→上限値で特性計算");
            dbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
        }
        if( dbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() ) ){
            //室内乾球温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)室DB＜下限→下限値で特性計算");
            dbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
        }


        //室温補正
        //能力補正


        //暖房　処理熱量
        rCode1 = this.pacDB.getKhta( wbOA );
        rCode2 = this.pacDB.getKhti( dbRA );
        rCode3 = this.Lratemin;
        //        System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode1+" "+rCode2+" "+rCode3);

        //        System.out.println("allTheat="+allTheat+" T_Loadout="+T_Loadout+" Rh="+Rc+" "+rCode1+" "+rCode2+"
"+rCode3);


        this.Sh_Loadout  = rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

        this.Rh = this.allTh_heat / this.Sh_Loadout;
        this.CAPrateC = 0.0;
        this.CAPrateH = 1.0;
        this.Th_Loadout = this.allTh_heat;//20120731
        if( this.Rh > 1.0 ) {
            this.CAPrateH = 1.0 / this.Rh;
            this.Rh = 1.0;
            this.Th_Loadout = this.Sh_Loadout;//21020731
        }
        //        System.out.println("Rh="+Rc);

        //低負荷領域の計算仕分け:Rh
        if( this.Rh < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                Rp = 0;
                break;

            case 1://  1_下限入力値固定
            case 2://  2_下限COP値固定
            case 3://  3_下限入力値と中間切片
                Rp = this.in_run_stop;
                break;

            default:
                Rp = 0;
            }
        }else{
            Rp = this.Rh;
        }

        //暖房　消費電力
        alph = this.pacDB.getAlphah( Rp );
        beta = this.pacDB.getBetah( Rp, D_fmh, D_pmh );

        if( D_pmh * D_fmh < 0.001 ){
            beta = 1.0;
        }

        rCode0 = alph * beta;
        rCode1 = this.pacDB.getKhhpid( Rp );
        rCode2 = this.pacDB.getKhwta( wbOA );
```

```
rCode3 = this.pacDB.getKhwti( dbRA );

this.PPE = rCode0 *rCode1 * rCode2 * rCode3 * D_out_PPEh_S;
//低負荷領域の計算仕分け:PPE
if( this.Rh < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.PPE = 0.0;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      this.PPE *= ( this.Rh / Rp );
      break;

   case 3://  3_下限入力値と中間切片
      this.PPE *= ( 0.5 + this.Rh / Rp / 2. );
      break;

   default:
   }
}else{
   //何もしない
}


//       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);

//暖房蓄熱除霜利用回数
if( this.DBoa > -7 && this.DBoa < 2. ){
   this.out_STGh_T = ( this.DBoa - (-7) ) / ( 2 - (-7) ) * this.tInterval / 3600. ;//暖房蓄熱除霜利用回数
}else if( this.DBoa >= 2. && this.DBoa < 5.5 ){
   this.out_STGh_T = ( 5.5 - this.DBoa ) / ( 5.5 - 2. ) * this.tInterval / 3600. ;//暖房蓄熱除霜利用回数
}else{
   this.out_STGh_T = 0;//暖房蓄熱除霜利用回数
}

   //System.out.println(" out_STGh_T="+this.out_STGh_T+" /DBoa="+this.DBoa );

//airOutEA
this.airOutEA.setFlowRate( this.fRate_airEAh );
this.airOutEA.setTempDB( ( -this.Th_Loadout + this.PPE ) / this.fRate_airEAh + this.airInOA.getTempDB() );


if( this.Rh < 0.0001 || this.Rh < this.in_run_stop ){
   this.CAPrateC = 0.0;
   this.CAPrateH = 0.0;
   this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
   this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
   this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
   this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
   this.PPE = 0.0;
   this.airOutEA.setFlowRate( 0. );
   this.airOutEA.setTempDB( this.airInOA.getTempDB() );

   this.out_STGh_T = 0;//暖房蓄熱除霜利用回数
}else if( this.Rh < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.CAPrateC = 0.0;
      this.CAPrateH = 0.0;
      this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
      this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
      this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
      this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
      this.PPE = 0.0;
      this.airOutEA.setFlowRate( 0. );
```

```
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );
            break;

         case 1://  1_下限入力値固定
            break;

         case 2://  2_下限COP値固定
            break;

         case 3://  3_下限入力値と中間切片
            break;

         default:
         }

      }

      //蓄熱量
      this.out_STQQ += ( this.st_heatLoss_calc() - this.out_STQQh_defrost_Co * this.out_STGh_T );

               ///System.out.println(" PPE="+PPE+" /Rc="+Rc+" /rCode1="+rCode1+" /rCode2="+rCode2+" /rCode3="+rCode3+"
/defrost="+(- this.out_STQQh_defrost_Co * this.out_STGh_T)+" /heatLoss="+this.st_heatLoss_calc());
   }

   /**
    * 冷暖　同時運転　暖房主体
    */
   private void heating_and_cooling_cal(){
      double rCode0,rCode1,rCode2,rCode3;
      double D_fmh,D_pmh;
      double D_out_Qh_S, D_out_Qc_S;
      double D_out_PPEh_S,alph,beta;
      double deltaWB;
      double Rch;
      double Rp;

       D_fmh      = this.fmh;
       D_pmh      = this.pmh;
       D_out_Qh_S = this.out_Qh_S;
       D_out_Qc_S = this.out_Qc_S;
       D_out_PPEh_S= this.out_PPEh_S;

       //if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L > 0.0001) { //低温入力の場合
       //   D_fmh    = this.fmh;
      //    D_pmh    = this.pmh;
      //    D_out_Qh_S  = this.out_Qh_L;
      //    D_out_PPEh_S= this.out_PPEh_L;
       //}

      //上下限のチェック 20120821nino
      double wbOA = this.WBoa;
      double dbRA = this.DBra;
      //暖房側
      if( wbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() ) ){
         //外気湿球温度＞上限の時　上限値の特性で運転
         this.message.append( "(C)外気WB＞上限→上限値で特性計算");
         wbOA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() );
      }
      if( wbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhta() ) ){
         //外気湿球温度＜下限の時　停止
         this.message.append( "(C)外気WB＜下限→停止");
         this.CAPrateC = 0.0;
         this.CAPrateH = 0.0;
         this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
         this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
         this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
         this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
         this.PPE = 0.0;
         this.airOutEA.setFlowRate( 0. );
         this.airOutEA.setTempDB( this.airInOA.getTempDB() );
         return;
      }
```

```
        if( dbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() ) ){
           //室内乾球温度＞上限の時　上限値の特性で運転
           this.message.append( "(C)室DB＞上限→上限値で特性計算");
           dbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
        }
        if( dbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() ) ){
           //室内乾球温度＜下限の時　下限値の特性で運転
           this.message.append( "(C)室DB＜下限→下限値で特性計算");
           dbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
        }


        //室温補正
        //能力補正


        //上下限のチェック 20120821nino
        //double dbOA = this.DBoa;
        double wbRA = this.WBra;

        //冷房側
        //if( dbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcta() ) ){
        // //外気乾球温度＞上限の時停止
        // this.message.append( "(C)外気DB＞上限→停止");
        // this.CAPrateC = 0.0;
        // this.CAPrateH = 0.0;
        // this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        // this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        // this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        // this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        // this.PPE = 0.0;
        // this.E_PPE = 0.0;
        // this.airOutEA.setFlowRate( 0. );
        // this.airOutEA.setTempDB( this.airInOA.getTempDB() );
        // return;
        //}
        //if( dbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcta() ) ){
        // //外気乾球温度＜下限の時　下限値の特性で運転
        // this.message.append( "(C)外気DB＜下限→下限値で特性計算");
        // dbOA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcta() );
        //}
        if( wbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() ) ){
           //室内湿球温度＞上限の時　上限値の特性で運転
           this.message.append( "(C)室WB＞上限→上限値で特性計算");
           wbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
        }
        if( wbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() ) ){
           //室内湿球温度＜下限の時　下限値の特性で運転
           this.message.append( "(C)室WB＜下限→下限値で特性計算");
           wbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
        }

        //室温補正
        //能力補正


         //暖房　処理熱量
         //定格暖房容量
         rCode1 = this.pacDB.getKhta( wbOA );
        rCode2 = this.pacDB.getKhti( dbRA );
        rCode3 = this.Lratemin;
        this.Sh_Loadout  = rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

        Rch = this.allTc_heat / this.Sh_Loadout;
        deltaWB = this.pacDB.getKdwb( Rch );

        //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode1+" "+rCode2+" "+rCode3);

        //       System.out.println("allTheat="+allTheat+" T_Loadout="+T_Loadout+" Rh="+Rc+" "+rCode1+" "+rCode2+"
"+rCode3);

        //補正暖房容量
```

```
 rCode1 = this.pacDB.getKhmta( wbOA + deltaWB );
this.Sh_Loadout  = rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

this.Rh = this.allTh_heat / this.Sh_Loadout;
this.CAPrateH = 1.0;
this.Th_Loadout = this.allTh_heat;//21020731
if( this.Rh > 1.0 ) {
   this.CAPrateH = 1.0 / this.Rh;
   this.Rh = 1.0;
   this.Th_Loadout = this.Sh_Loadout;//20120731
}
//       System.out.println("Rh="+Rc);

//低負荷領域の計算仕分け:Rh
if( this.Rh < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = this.Rh;
}

//暖房　消費電力
alph = this.pacDB.getAlphah( Rp );
beta = this.pacDB.getBetah( Rp, D_fmh, D_pmh );

if( D_pmh * D_fmh < 0.001 ){
   beta = 1.0;
}

rCode0 = alph * beta;
rCode1 = this.pacDB.getKhhpid( Rp );
rCode2 = this.pacDB.getKhmwta( wbOA + deltaWB );
rCode3 = this.pacDB.getKhwti( dbRA );

this.PPE = rCode0 * rCode1 * rCode2 * rCode3 * D_out_PPEh_S;
//低負荷領域の計算仕分け:PPE
if( this.Rh < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.PPE = 0.0;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      this.PPE *= ( this.Rh / Rp );
      break;

   case 3://  3_下限入力値と中間切片
      this.PPE *= ( 0.5 + this.Rh / Rp / 2. );
      break;

   default:
   }
}else{
   //何もしない
}
```

```
    //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);

    //冷房　処理熱量
    rCode2 = this.pacDB.getKcti( wbRA );
    rCode3 = this.Lratemin;

    this.Sc_Loadout  = rCode2 * rCode3 * D_out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]
    this.Rc = this.allTc_heat / this.Sc_Loadout;
    this.CAPrateC = 1.0;
    this.Tc_Loadout = this.allTc_heat;//20120731
    if( this.Rc > 1.0 ) {
       this.CAPrateC = 1.0 / this.Rc;
       this.Rc = 1.0;
       this.Tc_Loadout = this.Sc_Loadout;//21020731
    }

    //airOutEA
    this.airOutEA.setFlowRate( this.fRate_airEAh );
    this.airOutEA.setTempDB( ( -this.Tc_Loadout + this.PPE ) / this.fRate_airEAh + this.airInOA.getTempDB() );


    if( this.Rh < 0.0001 ){
       this.CAPrateC = 0.0;
       this.CAPrateH = 0.0;
       this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
       this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
       this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
       this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
       this.PPE = 0.0;
       this.airOutEA.setFlowRate( 0. );
       this.airOutEA.setTempDB( this.airInOA.getTempDB() );
    }else if( this.Rh < this.in_run_stop ){
       switch( this.num_calcTypeLowerRangeLoad){
       case 0://  0_発停運転
          this.CAPrateC = 0.0;
          this.CAPrateH = 0.0;
          this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
          this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
          this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
          this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
          this.PPE = 0.0;
          this.airOutEA.setFlowRate( 0. );
          this.airOutEA.setTempDB( this.airInOA.getTempDB() );
          break;

       case 1://  1_下限入力値固定
          break;

       case 2://  2_下限COP値固定
          break;

       case 3://  3_下限入力値と中間切片
          break;

       default:
       }

    }
       //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
}


private double WB_cal( double ddb, double xx ){
    double wwb, x1, de = 1;
    int j, kk = -1, j = 0;
    wwb = ddb;

    do{
       j++;
       x1 = Psychrometrics.FNXtw( ddb, wwb );
       if( Math.abs(x1-xx) < 0.000003 ) {
```

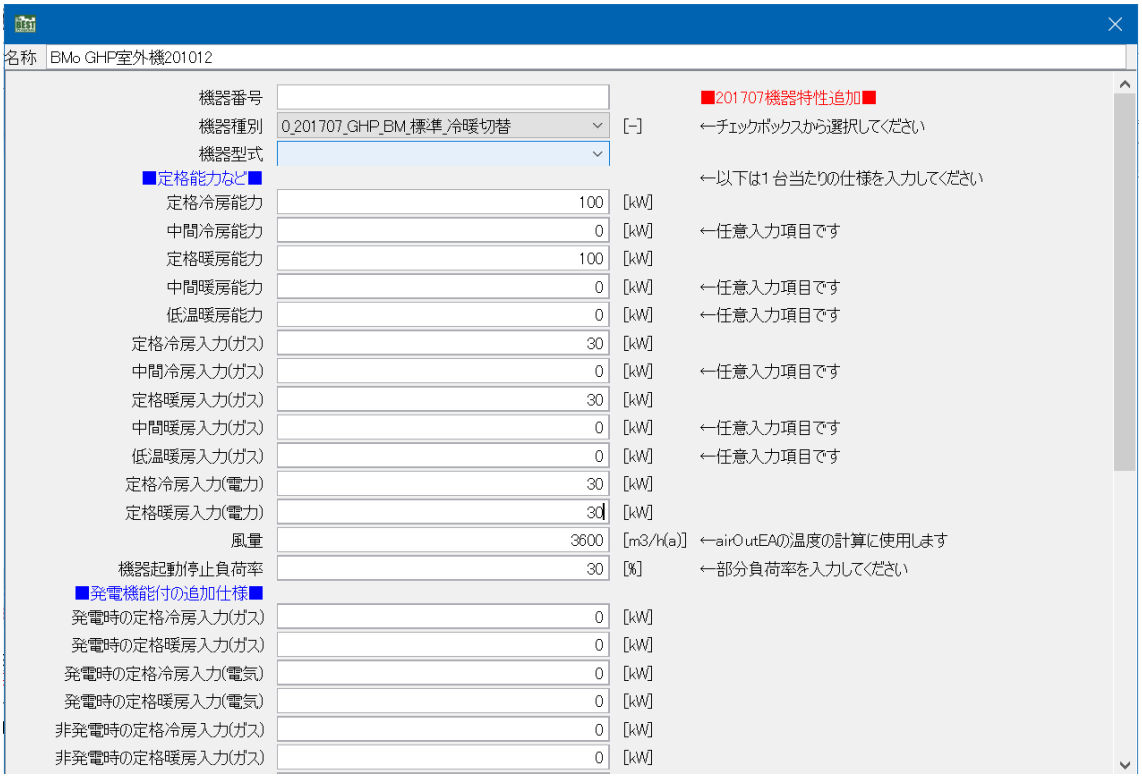
```
//          System.out.println(" j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
            return wwb;
         }
         if( x1 < xx ) {
            if( kk == -1 ){
               de = de / 2.0;
            }
            wwb = wwb + de;
            kk = 1;
         }
            if(x1 >= xx ) {
            if( kk == 1 ){
               de = de / 2.0;
            }
            wwb = wwb - de;
            kk = -1;
         }

      }while( j < 1000 );
      //    System.out.println("*j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
      return wwb;

   }
}
```

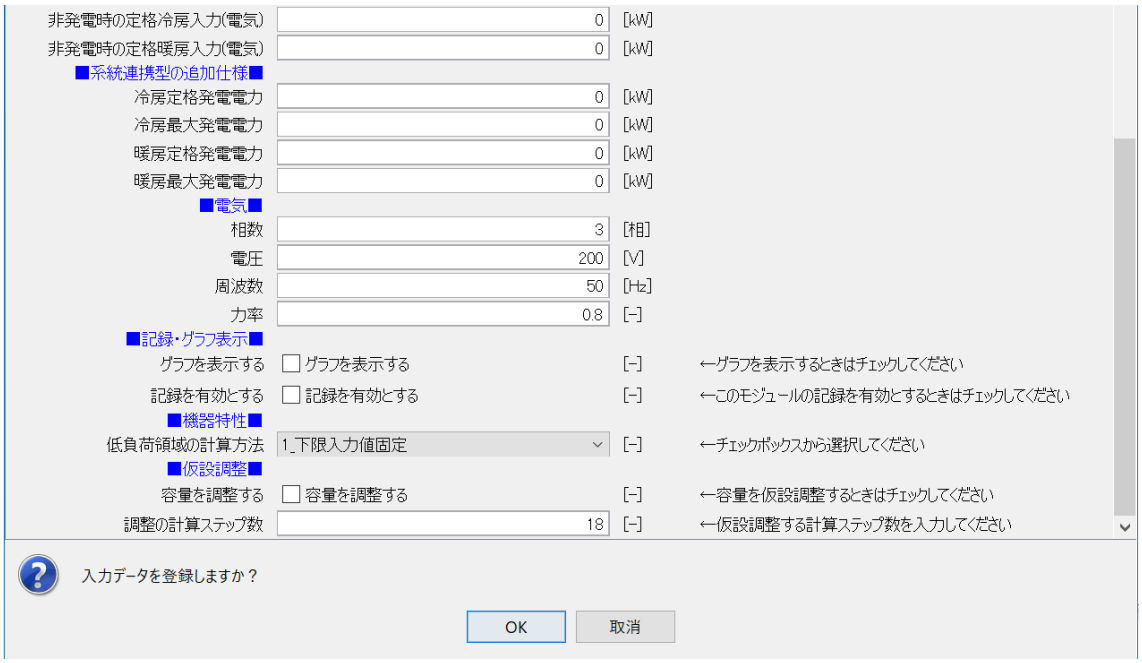
「BMo GHP 室外機 201012」（場所：設備 2015／個別分散 2015／）

モジュール名	BMo GHP 室外機 201012
クラス	BuillMultiGHPOut_SModule20101212

（1）入力画面

・スペック





（2）モジュールの概要

GHP タイプのビル用マルチ室外機モジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

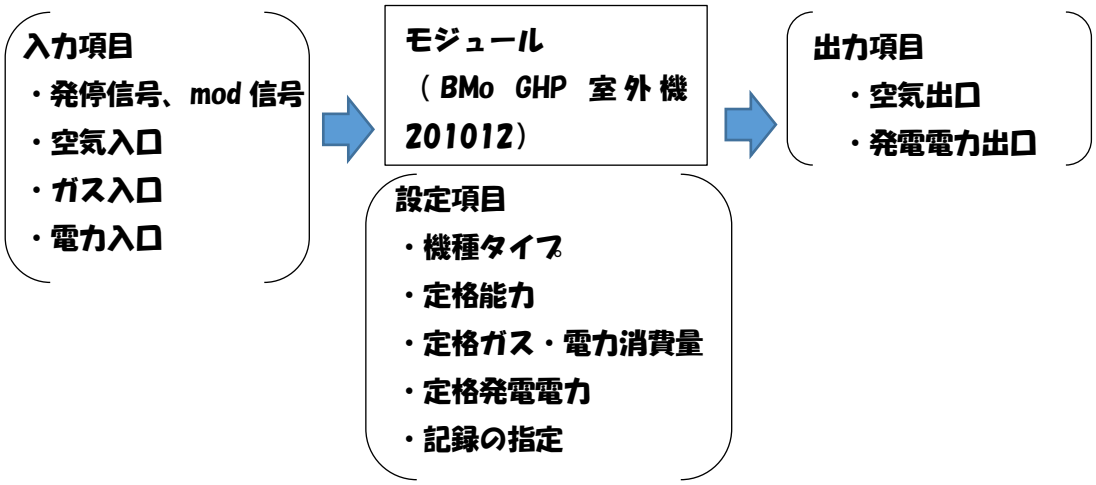


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	機器番号	String	m_no			ー	ー		
2	機器種別	String	m_ty	0_201707_GHP_BM_標準_冷暖切替、0_201303_GHP_BM_標準_冷暖切替、1_201303_GHP_BM_発電機付自己消費型、2_201303_GHP_BM_発電機付系統連携型、3_201303_GHP_BM_標準_冷暖同時、0_GHP_BM_標準_冷暖切替 201006、1_GHP_BM_発電機付自己消費型 201006、2_GHP_BM_発電機付系統連携型 201006、3_GHP_BM_標準_冷暖同時 201006	[-]	ー	ー		←チェックボックスから選択してください
3	機器型式	String	m_kt	#File 、BMList¥¥BMGHPoutListA.csv		ー	ー		
4	■定格能力など■					ー	ー		←以下は１台当たりの仕様を入力してください
5	定格冷房能力	double	out_Qc_S	100	[kW]	ー	0		
6	中間冷房能力	double	out_Qc_C	0	[kW]	ー	0		←任意入力項目です
7	定格暖房能力	double	out_Qh_S	100	[kW]	ー	0		
8	中間暖房能力	double	out_Qh_C	0	[kW]	ー	0		←任意入力項目です
9	低温暖房能力	double	out_Qh_L	0	[kW]	ー	0		←任意入力項目です
10	定格冷房入力(ガス)	double	out_GASc_S	30	[kW]	ー	0		

11	中間冷房入力(ガス)	double	out_GASc_C	0	[kW]	―	0		←任意入力項目です
12	定格暖房入力(ガス)	double	out_GASh_S	30	[kW]	―	0		
13	中間暖房入力(ガス)	double	out_GASh_C	0	[kW]	―	0		←任意入力項目です
14	低温暖房入力(ガス)	double	out_GASh_L	0	[kW]	―	0		←任意入力項目です
15	定格冷房入力(電力)	double	out_PPEc_S	30	[kW]	―	0		
16	定格暖房入力(電力)	double	out_PPEh_S	30	[kW]	―	0		
17	風量	double	fRate_airEA	3600	[m3/h(a)]	―	0		←airOutEA の温度の計算に使用します
18	機器起動停止負荷率	double	in_run_stop	30	[%]	100	0		←部分負荷率を入力してください
19	■発電機能付の追加仕様■					―	―		
20	発電時の定格冷房入力(ガス)	double	out_GASc_S_ari	0	[kW]	―	0		
21	発電時の定格暖房入力(ガス)	double	out_GASh_S_ari	0	[kW]	―	0		
22	発電時の定格冷房入力(電気)	double	out_PPEc_S_ari	0	[kW]	―	0		
23	発電時の定格暖房入力(電気)	double	out_PPEh_S_ari	0	[kW]	―	0		
24	非発電時の定格冷房入力(ガス)	double	out_GASc_S_nasi	0	[kW]	―	0		
25	非発電時の定格暖房入力(ガス)	double	out_GASh_S_nasi	0	[kW]	―	0		
26	非発電時の定格冷房入力(電気)	double	out_PPEc_S_nasi	0	[kW]	―	0		
27	非発電時の定格暖房入力	double	out_PPEh_S_nasi	0	[kW]	―	0		

	(電気)								
28	■系統連携型の追加仕様■					―	―		
29	冷房定格発電電力	double	out_ELEc_dy	0	[kW]	―	0		
30	冷房最大発電電力	double	out_ELEc_dymax	0	[kW]	―	0		
31	暖房定格発電電力	double	out_ELEh_dy	0	[kW]	―	0		
32	暖房最大発電電力	double	out_ELEh_dymax	0	[kW]	―	0		
33	■電気■					―	―		
34	相数	int	phase	3	[相]	―	1		
35	電圧	double	voltage	200	[V]	―	100		
36	周波数	double	frequency	50	[Hz]	60	50		
37	力率	double	powerFactor	0.8	[-]	1	0		
38	■記録・グラフ表示■					―	―		
39	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
40	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください
41	■機器特性■					―	―		
42	低負荷領域の計算方法	String	calcTypeLowerRangeLoad	0_発停運転、1_下限入力値固定、2_下限 COP 値固定、3_下限入力値と中間切片	[-]	―	―		←チェックボックスから選択してください
43	■仮設調整■					―	―		
44	容量を調整する	boolean	isAdjust2012	FALSE	[-]	―	―		←容量を仮設調整するときはチェックしてください
45	調整の計算ステップ数	int	numAdjustSteps	18	[-]	―	6		←仮設調整する計算ステップ数を入力してください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	外気入口	L0_airInOA	airInOA	-	状態	空気	入口	
5	排気出口	L0_airOutEA	airOutEA	-	状態	空気	出口	
6	冷媒入口	L0_valInLine	valInLine	-	値	値	入口	
7	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	
8	ガス入口	L0_gasIn	_gasIn	-	状態	ガス	入口	
9	発電電力出口	L0_eleOutGEN	eleOutGEN	-	状態	電気	出口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	室外機 Message#-#-	メッセージ	—	メッセージ
2	室内機冷却要求顕熱量合計#W#熱量	室内機冷却要求顕熱量合計	W	My
3	室内機加熱要求顕熱量合計#W#熱量	室内機加熱要求顕熱量合計	W	My
4	室内機冷却要求全熱量合計#W#熱量	室内機冷却要求全熱量合計	W	My
5	室内機加熱要求全熱量合計#W#熱量	室内機加熱要求全熱量合計	W	My
6	室内機冷却能力補正比率	室内機冷却能力補正比率	—	My
7	室内機加熱能力補正比率	室内機加熱能力補正比率	—	My
8	室外機消費電力#W#消費電力	室外機消費電力	W	エネルギー消費
9	室外機冷却可能熱量/最大#W#熱量	室外機冷却可能熱量/最大	W	My
10	室外機加熱可能熱量/最大#W#熱量	室外機加熱可能熱量/最大	W	My
11	室外機冷却実処理熱量/現状#W#熱量	室外機冷却実処理熱量/現状	W	My
12	室外機加熱実処理熱量/現状#W#熱量	室外機加熱実処理熱量/現状	W	My
13	処理全熱量合計負荷#W#-	処理全熱量合計負荷	W	負荷
14	室外機消費ガス(空調)#W#消費ガス(空調)	室外機消費ガス(空調)	W	エネルギー消費
15	室外機発電電力#W#発電電力	室外機発電電力	W	エネルギー消費
16	室外機消費ガス(自己消費)#W#消費ガス(自己消費)	室外機消費ガス(自己消費)	W	エネルギー消費
17	室外機消費ガス(系統連携)#W#消費ガス(系統連携)	室外機消費ガス(系統連携)	W	エネルギー消費
18	冷房(暖房)能力比#-#-	冷房(暖房)能力比	—	My
19	室外機空気入口 DB#℃#-	室外機空気入口 DB	℃	入口
20	室外機空気入口 WB#℃#-	室外機空気入口 WB	℃	入口
21	室外機空気入口流量#g/s#-	室外機空気入口流量	g/s	入口
22	室外機調整冷却能力#W#熱量	室外機調整冷却能力	W	調整
23	室外機調整加熱能力#W#熱量	室外機調整加熱能力	W	調整

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.Map;
import java.util.Set;

import jp.or.ibec.best.DO.BM_EHPdata;
//import jp.or.ibec.best.DO.BM_GHPdata;
import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestGas;
import jp.or.ibec.best.client.bestgui.file.service.FileConstants;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.essentials.utility.Psychrometrics;

/**
 * @author SUGANAGA 2008/04/07 /20080428 ビルマルチ：室外機クラス
 *
 * 記録の有効無効判断を追加 /20080909
 *
 * tabata  自己消費型を追加
 *
 * /20101212nino  冷暖同時に対応するため、変数などを冷房、暖房に分離
 *   機器特性201006へ対応
 *
 */
public class BuillMultiGHPOut_SModule20101212 extends AbstractBestModule implements
      IBestMetaModule {

   /**
    * 論理アドレス
    */
   private final String moduleName = "(BuillMultiGHPOut_S20101212)";
   private final String S_NODE_airInOA   = "L0_airInOA";// 外気
   private final String S_NODE_airOutEA  = "L0_airOutEA";// 排気
   private final String S_NODE_valInLine = "L0_valInLine"; // 単線
   private final String S_NODE_eleIn     = "L0_eleIn";// 電力
   private final String S_NODE_eleOut    = "L0_eleOutGEN";// 発電電力
   private final String S_NODE_gasIn     = "L0_gasIn";// ガス
   private final String C_NODE_swcIn = "L1_swcIn";// 運転状態：off/on
   private final String C_NODE_modIn = "L1_modIn";// 空調モード：停止/冷却/加熱
   private final String R_NODE = "L2_recOut";
   private Map<String, BM_EHPdata> rmlinemap = null;

   /**
    * 外部定義
    */
   private final String SPEC_name = "名称";
   private final String SPEC_m_no = "機器番号";
   private final String SPEC_m_ty = "機器種別";
   private final String SPEC_m_kt = "機器型式";
   private final String SPEC_out_Qc_S = "定格冷房能力[W]";// kW
   private final String SPEC_out_Qc_C = "中間冷房能力[W]";// kW
   private final String SPEC_out_PPEc_S = "定格冷房入力(電力)[W]";// kW
   // private final String SPEC_out_PPEc_C = "中間冷房入力(電力)";//kW
   private final String SPEC_out_Qh_S = "定格暖房能力[W]";// kW
   private final String SPEC_out_Qh_C = "中間暖房能力[W]";// kW
```

```
private final String SPEC_out_Qh_L = "低温暖房能力[W]";// kW
private final String SPEC_out_PPEh_S = "定格暖房入力(電力)[W]";// kW
// private final String SPEC_out_PPEh_C = "中間暖房入力(電力)";//kW
// private final String SPEC_out_PPEh_L = "低温暖房入力(電力)";//kW
// tabata追記
private final String SPEC_out_GASc_S = "定格冷房入力(ガス)[W]";// kW
private final String SPEC_out_GASc_C = "中間冷房入力(ガス)[W]";// kW
private final String SPEC_out_GASh_S = "定格暖房入力(ガス)[W]";// kW
private final String SPEC_out_GASh_C = "中間暖房入力(ガス)[W]";// kW
private final String SPEC_out_GASh_L = "低温暖房入力(ガス)[W]";// kW
/*
 * private final String SPEC_out_CLEa_R = "クランクケースヒータ(運転時)";//W private
 * final String SPEC_out_CLEa_S = "クランクケースヒータ(停止時)";//W private final String
 * SPEC_out_TKEa_R = "定格待機電力(運転時)";//W private final String SPEC_out_TKEa_T =
 * "定格待機電力(待機時)";//W private final String SPEC_out_TKEa_S =
 * "定格待機電力(停止時)";//W
 */
private final String SPEC_fRate_airEA   = "風量[g/s]";

private final String SPEC_in_run_stop = "機器起動停止負荷率[-]";// [%]
private final String SPEC_Phase = "相数[-]";
private final String SPEC_Voltage = "電圧[V]";
private final String SPEC_Frequency = "周波数[Hz]";
private final String SPEC_PowerFactor = "力率[-]";

//tabata20100301
private final String SPEC_out_GASc_S_ari  = "発電時の定格冷房入力(ガス)[W]";
private final String SPEC_out_GASc_S_nasi = "非発電時の定格冷房入力(ガス)[W]";
private final String SPEC_out_PPEc_S_ari  = "発電時の定格冷房入力(電気)[W]";
private final String SPEC_out_PPEc_S_nasi = "非発電時の定格冷房入力(電気)[W]";
private final String SPEC_out_GASh_ari    = "発電時の定格暖房入力(ガス)[W]";
private final String SPEC_out_GASh_nasi   = "非発電時の定格暖房入力(ガス)[W]";
private final String SPEC_out_PPEh_S_ari  = "発電時の定格暖房入力(電気)[W]";
private final String SPEC_out_PPEh_S_nasi = "非発電時の定格暖房入力(電気)[W]";
private final String SPEC_out_ELEc_dy     = "冷房定格発電電力[W]";
private final String SPEC_out_ELEc_dymax  = "冷房最大発電電力[W]";
private final String SPEC_out_ELEh_dy     = "暖房定格発電電力[W]";
private final String SPEC_out_ELEh_dymax  = "暖房最大発電電力[W]";

private final String SPEC_isGVisible = "グラフを表示する";// このグラフを表示する
private final String SPEC_isRecord = "記録を有効とする";// このモジュールの記録を有効とする

private final String SPEC_CalcTypeLowerRangeLoad = "低負荷領域の計算方法";

private final String SPEC_isAdjust2012   = "容量を調整する";
private final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";

private StringBuffer message = new StringBuffer();

private BestAir airInOA = null;
private BestAir airOutEA = null;
private BestElectricity eleIn = null; // 電力
private BestElectricity eleOut = null; // 電力
private BestGas gasIn = null; // ガス
private Double CAPrateC =  new Double( 1 );//室内機能力補正比率[-]冷房側
private Double CAPrateH =  new Double( 1 );//室内機能力補正比率[-]暖房側
private BM_EHPdata RMdata = null; // 電力 //GHPでもそのまま使用
private PACDBManager pacDB = null;

private int modIn;
private int swcIn;
private int mState; // 停止、冷房、暖房フラグ

private boolean isCOOLandHEATout = false;//冷暖同時機種の場合=true
private int dbflg=0;


/**
 * 変数（外部定義に対応）
 */

private String name;// "機器名称";
```

```
private String m_no;// = "機器番号";
private String m_ty;// = "機器種別";
private String m_kt;// = "機器型式";
private double out_Qc_S;// = "定格冷房能力";//kW
private double out_Qc_C;// = "中間冷房能力";//kW
private double out_PPEc_S;// = "定格冷房入力(電力)";//kW
// private double out_PPEc_C ;//= "中間冷房入力(電力)";//kW
private double out_Qh_S;// = "定格暖房能力";//kW
private double out_Qh_C;// = "中間暖房能力";//kW
private double out_Qh_L;// = "低温暖房能力";//kW
private double out_PPEh_S;// = "定格暖房入力(電力)";//kW
// private double out_PPEh_C ;//= "中間暖房入力(電力)";//kW
// private double out_PPEh_L ;//= "低温暖房入力(電力)";//kW
// tabata追加
private double out_GASc_S;// = "定格冷房入力(ガス)";//kW
private double out_GASc_C;// = "中間冷房入力(ガス)";//kW
private double out_GASh_S;// = "定格暖房入力(ガス)";//kW
private double out_GASh_C;// = "中間暖房入力(ガス)";//kW
private double out_GASh_L;// = "低温暖房入力(ガス)";//kW

//tabata20100301
private double out_GASc_S_ari;// = "発電時の定格冷房入力(ガス)";//kW
private double out_GASc_S_nasi;// = "発電なしの定格冷房入力(ガス)";//kW
private double out_PPEc_S_ari;// = "発電時の定格冷房入力(電気)";//kW
private double out_PPEc_S_nasi;// = "発電なしの定格冷房入力(電気)";//kW
private double out_GASh_S_ari;// = "発電時の定格暖房入力(ガス)";//kW
private double out_GASh_S_nasi;// = "発電なしの定格暖房入力(ガス)";//kW
private double out_PPEh_S_ari;// = "発電時の定格暖房入力(電気)";//kW
private double out_PPEh_S_nasi;// = "発電なしの定格暖房入力(電気)";//kW

private double out_ELEc_dy;//冷房定格発電電力
private double out_ELEc_dymax;//冷房最大発電電力
private double out_ELEh_dy;//暖房定格発電電力
private double out_ELEh_dymax;//暖房最大発電電力


/*
 * private double out_CLEa_R=0.0 ;//= "クランクケースヒータ(運転時)";//W private double
 * out_CLEa_S=0.0 ;//= "クランクケースヒータ(停止時)";//W private double out_TKEa_R =0.0
 * ;//= "定格待機電力(運転時)";//W private double out_TKEa_T =0.0 ;//=
 * "定格待機電力(待機時)";//W private double out_TKEa_S =0.0 ;//= "定格待機電力(停止時)";//W
 */

private double fRate_airEA ;//風量[g/s]

private double in_run_stop;// = "機器起動停止負荷率";//[%]
private int phase; // 相数[-]
private double voltage; // 電圧[V]
private double frequency; // 周波数[Hz]
private double powerFactor; // 力率[-]

private double Lp; // 冷媒管長さ[m]
private double Lh; // 室内機/室外機高低差[m]
private boolean isGVisible = false;// このグラフを表示する=true
private boolean isRecord = false;// 記録を有効とする=true

private String[] rmnm = new String[50];

/**
 * 出力（建物）
 */

private final String RECORD_message = "室外機Message#-#-";
private final String RECORD_ID1c = "室内機冷却要求顕熱量合計#W#熱量";
private final String RECORD_ID1h = "室内機加熱要求顕熱量合計#W#熱量";
private final String RECORD_ID2c = "室内機冷却要求全熱量合計#W#熱量";
private final String RECORD_ID2h = "室内機加熱要求全熱量合計#W#熱量";
private final String RECORD_ID3c = "室内機冷却能力補正比率";
private final String RECORD_ID3h = "室内機加熱能力補正比率";
private final String RECORD_ID4  = "室外機消費電力#W#消費電力";
private final String RECORD_ID5c = "室外機冷却可能熱量/最大#W#熱量";
private final String RECORD_ID5h = "室外機加熱可能熱量/最大#W#熱量";
```

```
private final String RECORD_ID6c = "室外機冷却実処理熱量/現状#W#熱量";
private final String RECORD_ID6h = "室外機加熱実処理熱量/現状#W#熱量";
private final String RECORD_qT   = "処理全熱量合計負荷#W#-";
// tabata追記
private final String RECORD_ID7  = "室外機消費ガス(空調)#W#消費ガス(空調)";
private final String RECORD_ID8  = "室外機発電電力#W#発電電力";
private final String RECORD_ID9  = "室外機消費ガス(自己消費)#W#消費ガス(自己消費)";
private final String RECORD_ID10 = "室外機消費ガス(系統連携)#W#消費ガス(系統連携)";
private final String RECORD_ID11 = "冷房(暖房)能力比#-#-";
//
private final String RECORD_DB_airInHS = "室外機空気入口DB#℃#-";
private final String RECORD_WB_airInHS = "室外機空気入口WB#℃#-";
private final String RECORD_M_airInHS  = "室外機空気入口流量#g/s#-";

private final String RECORD_out_Qc_Adjust = "室外機調整冷却能力#W#熱量";
private final String RECORD_out_Qh_Adjust = "室外機調整加熱能力#W#熱量";


/**
 * その他変数
 */

private double Rc; // 部分負荷率　冷房側
private double Rh; // 部分負荷率　暖房側

private double DBoa; // 外気乾球温度[℃]
private double WBoa; // 外気湿球温度[℃]
private double XGoa; // 外気絶対湿度[g/gD.A.]

private double DBra; // 室内機吸込乾球温度[℃]
private double WBra; // 室内機吸込湿球温度[℃]
private double XGra; // 室内機吸込絶対湿度[g/gD.A.]

private double Sc_Loadout;//室外機処理可能熱量（冷却全熱）[W]
private double Sh_Loadout;//室外機処理可能熱量（加熱全熱）[W]
private double Tc_Loadout;//室外機実処理熱量（冷却全熱）[W]
private double Th_Loadout;//室外機実処理熱量（加熱全熱）[W]
private double allSc_heat;//室内機処理熱量（冷却顕熱）[W]
private double allSh_heat;//室内機処理熱量（加熱顕熱）[W]
private double allTc_heat;//室内機処理熱量（冷却全熱）[W]
private double allTh_heat;//室内機処理熱量（加熱全熱）[W]

private double PPE;
private double E_PPE;
// tabata追記
private double GAS;
private double GASself;
private double GASout;
private double PPEout;
private double E_PPEout;

private int num;
private String kiki_file;
private String path = "GHP";
private String[] filenames = new String[1];
private String equipmentName;

private double DBra_max;
private double WBra_min;
private double Lratemin;
private double fmc;
private double pmc;
private double fmh;
private double pmh;

private boolean isStandard = false;
private boolean isStandardCHFree = false;
private boolean isDynamo = false;
private boolean isDynamoLink = false;
private boolean isPchoerunK = false;//20170707　消費電力が負荷率の特性式（これまでは外気の特性式）
private boolean is2018SinseiCHFree = false;//2018冷暖同時誘導基準申請計算モード
```

```
    private boolean isSelectSPECList = false;//機器SPECListの機器を指定している=true

    // グラフ表示など
    private GraphJFrameBuilMultiGHPOut_S20101212 gBMout = null;


    //低負荷領域の計算方法
    private String calcTypeLowerRangeLoad = null;//"低負荷領域の計算方法";
    private int num_calcTypeLowerRangeLoad;//"低負荷領域の計算方法";

    //仮設調整モード
    //仮設調整モード2012
    private boolean isAdjust2012  = false;//true="台数を調整する";//
    //private boolean isAdjustMode = true;//20110517nino 空衛学会OS論文検討用＝true
    private double out_Qc_Adjust;//= "調整冷却能力";
    private double out_Qh_Adjust;//= "調整加熱能力";
    private double fRate_airEAc ;//冷却時風量[g/s]
    private double fRate_airEAh ;//加熱時風量[g/s]

    private int numAdjustSteps;//"調整の計算ステップ数";
    private LinkedList<Double> rcList = null;//必要台数
    private LinkedList<Double> rhList = null;//必要台数
    private double max_rcAdjust = 1.;
    private double max_rhAdjust = 1.;
    private double ave_rcAdjust = 0;
    private double ave_rhAdjust = 0;
    private double out_rcAdjust = 1.;//= "調整率冷房";
    private double out_rhAdjust = 1.;//= "調整率暖房";
// private double in_run_stop_Num;//= "機器起動停止負荷率"台数補正;


    private double out_Qc_S_1;// = "定格冷房能力";//kW
    private double out_Qc_C_1;// = "中間冷房能力";//kW
    private double out_PPEc_S_1;// = "定格冷房入力(電力)";//kW
    // private double out_PPEc_C ;//= "中間冷房入力(電力)";//kW
    private double out_Qh_S_1;// = "定格暖房能力";//kW
    private double out_Qh_C_1;// = "中間暖房能力";//kW
    private double out_Qh_L_1;// = "低温暖房能力";//kW
    private double out_PPEh_S_1;// = "定格暖房入力(電力)";//kW
    // private double out_PPEh_C ;//= "中間暖房入力(電力)";//kW
    // private double out_PPEh_L ;//= "低温暖房入力(電力)";//kW
    // tabata追加
    private double out_GASc_S_1;// = "定格冷房入力(ガス)";//kW
    private double out_GASc_C_1;// = "中間冷房入力(ガス)";//kW
    private double out_GASh_S_1;// = "定格暖房入力(ガス)";//kW
    private double out_GASh_C_1;// = "中間暖房入力(ガス)";//kW
    private double out_GASh_L_1;// = "低温暖房入力(ガス)";//kW

    //tabata20100301
    private double out_GASc_S_ari_1;// = "発電時の定格冷房入力(ガス)";//kW
    private double out_GASc_S_nasi_1;// = "発電なしの定格冷房入力(ガス)";//kW
    private double out_PPEc_S_ari_1;// = "発電時の定格冷房入力(電気)";//kW
    private double out_PPEc_S_nasi_1;// = "発電なしの定格冷房入力(電気)";//kW
    private double out_GASh_S_ari_1;// = "発電時の定格暖房入力(ガス)";//kW
    private double out_GASh_S_nasi_1;// = "発電なしの定格暖房入力(ガス)";//kW
    private double out_PPEh_S_ari_1;// = "発電時の定格暖房入力(電気)";//kW
    private double out_PPEh_S_nasi_1;// = "発電なしの定格暖房入力(電気)";//kW

    private double out_ELEc_dy_1;//冷房定格発電電力
    private double out_ELEc_dymax_1;//冷房最大発電電力
    private double out_ELEh_dy_1;//暖房定格発電電力
    private double out_ELEh_dymax_1;//暖房最大発電電力

    @Override
    public void setProfile(BestSpecs spec) {
        // 外部定義項目取得
        if (spec == null) {
            return;
        }
        Map<String, String> map = spec.getSpec();
        if (map == null) {
```

```
    return;
}

// 機器名称
this.name = spec.getSpecValue( this.SPEC_name, this.name );

// 機器番号
this.m_no = spec.getSpecValue( this.SPEC_m_no, "" );

// 機器種別 0_標準型、1_発電機付自己消費型、2_発電機付系統連携型、3_冷暖同時
//
// "0_GHP_BM_標準_冷暖切替201006"
// "1_GHP_BM_発電機付自己消費型201006"
// "2_GHP_BM_発電機付系統連携型201006"
// "3_GHP_BM_標準_冷暖同時201006"
//
//this.m_ty = spec.getSpecValue( this.SPEC_m_ty, "0_201303_GHP_BM_標準_冷暖切替" );
this.m_ty = spec.getSpecValue( this.SPEC_m_ty, "0_201707_GHP_BM_標準_冷暖切替" );//201707機器特性変更

//kiki_file = "GHP_BM_Standard201303";
kiki_file = "GHP_BM_Standard201707";
//201707
if ( m_ty.equals("0_201707_GHP_BM_標準_冷暖切替")){
    kiki_file = "GHP_BM_Standard201707";
    this.isCOOLandHEATout = false;
    this.isStandard = true;
    this.isPchoerunK = true;
}
//
if ( m_ty.equals("0_201303_GHP_BM_標準_冷暖切替")){
    kiki_file = "GHP_BM_Standard201303";
    this.isCOOLandHEATout = false;
    this.isStandard = true;
    this.isPchoerunK = false;
}
if ( m_ty.equals("1_201303_GHP_BM_発電機付自己消費型")){
    kiki_file = "GHP_BM_Dynamo201303";
    this.isCOOLandHEATout = false;
    this.isDynamo = true;
    this.isPchoerunK = false;
}
if ( m_ty.equals("2_201303_GHP_BM_発電機付系統連携型")){
    kiki_file = "GHP_BM_DynamoLink201303";
    this.isCOOLandHEATout = false;
    this.isDynamoLink = true;
    this.isPchoerunK = false;
}
if ( m_ty.equals("3_201303_GHP_BM_標準_冷暖同時")){
    kiki_file = "GHP_BM_Standard_CHFree201303";
    this.isCOOLandHEATout = true;
    this.isStandardCHFree = true;
    this.isPchoerunK = false;
}
if ( m_ty.equals("3_201303_GHP_BM_標準_冷暖同時2018申請")){//20181017
    kiki_file = "GHP_BM_Standard_CHFree201303";
    this.isCOOLandHEATout = true;
    this.isStandardCHFree = true;
    this.isPchoerunK = false;
    this.is2018SinseiCHFree = true;
}

//
if ( m_ty.equals("0_GHP_BM_標準_冷暖切替201006")){
    kiki_file = "GHP_BM_Standard201006";
    this.isCOOLandHEATout = false;
    this.isStandard = true;
    this.isPchoerunK = false;
}
if ( m_ty.equals("1_GHP_BM_発電機付自己消費型201006")){
    kiki_file = "GHP_BM_Dynamo201006";
    this.isCOOLandHEATout = false;
    this.isDynamo = true;
```

```
    this.isPchoerunK = false;
}
if ( m_ty.equals("2_GHP_BM_発電機付系統連携型201006")){
    kiki_file = "GHP_BM_DynamoLink201006";
    this.isCOOLandHEATout = false;
    this.isDynamoLink = true;
    this.isPchoerunK = false;
}
if ( m_ty.equals("3_GHP_BM_標準_冷暖同時201006")){
    kiki_file = "GHP_BM_Standard_CHFree201006";
    this.isCOOLandHEATout = true;
    this.isStandardCHFree = true;
    this.isPchoerunK = false;
}

// 機器型式
this.m_kt = spec.getSpecValue( this.SPEC_m_kt, "" );

FileReadBMSpecModule20110221 bmSpecFR = null;

//機器specの取り込み
if( this.m_kt == null || this.m_kt.equals( "" )){
    //機器型番指定が無い時は、ダイアログ入力値を使用する
}else{
    //機器型番指定の時は、外部SPECListのデータを使用する
    this.isSelectSPECList = true;
    //BM機器SPECListA　のデータの取込み
      String filepathName = FileConstants.DIR_SYSTEM + "/BMList/BMGHPoutSPECListA.csv";
    bmSpecFR = new FileReadBMSpecModule20110221( filepathName );
}

//PerformanceCurves
//Class1
//Class2
//Class3
//MakerName
//SiriesName
//ModelCode
//OutdoorUnitCode
//RefrigerantType
//Fuel
//FrequencyEleIn
//NCCapacity
//NCEleInputMain
//NCEleInputSub
//NCGasInput
//NCOilInput
//NCTairIn
//NCTairOut
//NCFRair
//NCPLossairIn
//NCTwatInCD
//NCTwatOutCD
//NCFRwatInCD
//NCLossPwatInCD
//NHCapacity
//NHEleInputMain
//NHEleInputSub
//NHGasInput
//NHOilInput
//NHTairIn
//NHTairOut
//NHFRair
//NHPLossairIn
//NHTwatInCD
//NHTwatOutCD
//NHFRwatInCD
//NHLossPwatInCD
//RestartDelayTime
//SubPreOpeTime
//SubAftOpeTime
//note
```

```
// 定格冷房能力
this.out_Qc_S = spec.getSpecValue( this.SPEC_out_Qc_S, 0. );

if( this.isSelectSPECList ){
   this.out_Qc_S = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NCCapacity" ) ) * 1000.;//kW->W
}

// 中間冷房能力
this.out_Qc_C = spec.getSpecValue( this.SPEC_out_Qc_C, 0. );

if( this.isSelectSPECList ){
   this.out_Qc_C = 0.;
}

// 定格冷房入力(電力)
this.out_PPEc_S = spec.getSpecValue( this.SPEC_out_PPEc_S, 0. );

if( this.isSelectSPECList ){
   this.out_PPEc_S = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NCEleInputMain" ) ) * 1000.;//kW->W;
}

// tabata追記
// 定格冷房入力(ガス)
this.out_GASc_S = spec.getSpecValue( this.SPEC_out_GASc_S, 0. );

if( this.isSelectSPECList ){
   this.out_GASc_S = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NCGasInput" ) ) * 1000.;//kW->W;
}

// 中間冷房入力(ガス)
this.out_GASc_C = spec.getSpecValue( this.SPEC_out_GASc_C, 0. );

if( this.isSelectSPECList ){
   this.out_GASc_C = 0.;
}

// 定格暖房能力
this.out_Qh_S = spec.getSpecValue( this.SPEC_out_Qh_S, 0. );

if( this.isSelectSPECList ){
   this.out_Qh_S = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NHCapacity" ) ) * 1000.;//kW->W;
}

// 中間暖房能力
this.out_Qh_C = spec.getSpecValue( this.SPEC_out_Qh_C, 0. );

if( this.isSelectSPECList ){
   this.out_Qh_C = 0.;
}

// 低温暖房能力
this.out_Qh_L = spec.getSpecValue( this.SPEC_out_Qh_L, 0. );
if ( this.out_Qh_L <= 0 ){
   if( this.out_Qh_S < 70000 ){
      this.out_Qh_L = this.out_Qh_S * 1.0;
   }else{
      this.out_Qh_L = this.out_Qh_S * 0.75;
   }
}
if( this.isSelectSPECList ){
   if( this.out_Qh_S < 70000 ){
      this.out_Qh_L = this.out_Qh_S * 1.0;
   }else{
      this.out_Qh_L = this.out_Qh_S * 0.75;
   }
}

// 定格暖房入力(電力)
this.out_PPEh_S = spec.getSpecValue( this.SPEC_out_PPEh_S, 0. );
```

```
if( this.isSelectSPECList ){
   this.out_PPEh_S = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NHEleInputMain" ) ) * 1000.;//kW->W;
}

// tabata追記
// 定格暖房入力(ガス)
this.out_GASh_S = spec.getSpecValue( this.SPEC_out_GASh_S, 0. );

if( this.isSelectSPECList ){
   this.out_GASh_S = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NHGasInput" ) ) * 1000.;//kW->W;
}

// 中間暖房入力(ガス)
this.out_GASh_C = spec.getSpecValue( this.SPEC_out_GASh_C, 0. );

if( this.isSelectSPECList ){
   this.out_GASh_C = 0.;
}

// 低温暖房入力(ガス)
this.out_GASh_L = spec.getSpecValue( this.SPEC_out_GASh_L, 0. );

if( this.isSelectSPECList ){
   this.out_GASh_L = 0.;
}

/*
 * // クランクケースヒータ(運転時) if(null!=map.get(this.SPEC_out_CLEa_R)){
 * this.out_CLEa_R =
 * Double.parseDouble((String)map.get(this.SPEC_out_CLEa_R)); } else {
 * System.out.println("(E)SPEC_クランクケースヒータ(運転時)がありません"); } //
 * クランクケースヒータ(停止時) if(null!=map.get(this.SPEC_out_CLEa_S)){
 * this.out_CLEa_S =
 * Double.parseDouble((String)map.get(this.SPEC_out_CLEa_S)); } else {
 * System.out.println("(E)SPEC_クランクケースヒータ(停止時)がありません"); } // 待機電力(運転時)
 * if(null!=map.get(this.SPEC_out_TKEa_R)){ this.out_TKEa_R =
 * Double.parseDouble((String)map.get(this.SPEC_out_TKEa_R)); } else {
 * System.out.println("(E)SPEC_待機電力(運転時)がありません"); } // 待機電力(待機時)
 * if(null!=map.get(this.SPEC_out_TKEa_T)){ this.out_TKEa_T =
 * Double.parseDouble((String)map.get(this.SPEC_out_TKEa_T)); } else {
 * System.out.println("(E)SPEC_待機電力(待機時)がありません"); } // 待機電力(停止時)
 * if(null!=map.get(this.SPEC_out_TKEa_S)){ this.out_TKEa_S =
 * Double.parseDouble((String)map.get(this.SPEC_out_TKEa_S)); } else {
 * System.out.println("(E)SPEC_待機電力(停止時)がありません"); }
 */

//SPEC_fRate_airEA   = "風量[g/s]";
this.fRate_airEA = spec.getSpecValue( this.SPEC_fRate_airEA, this.out_Qc_S * 0.11 );//20110505nino

if( this.isSelectSPECList ){
   this.fRate_airEA = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "NCFRair" ) ) * 1200 / 3600.;//CMH->g/s
   if( this.fRate_airEA == 0 ){
      this.fRate_airEA = this.out_Qc_S * 0.11;//20110505nino
   }
}

// 機器起動停止負荷率
this.in_run_stop = spec.getSpecValue( this.SPEC_in_run_stop, 0.3 );

// 相数
this.phase = spec.getSpecValue( this.SPEC_Phase, 3 );

// 電圧
this.voltage = spec.getSpecValue( this.SPEC_Voltage, 200. );

// 周波数
this.frequency = spec.getSpecValue( this.SPEC_Frequency, 50 );

if( this.isSelectSPECList ){
   if( bmSpecFR.getSpec( this.m_kt, "FrequencyEleIn" ).equals( "50 60" ) ){
      this.frequency = 50;
```

```
    }else{
        this.frequency = Double.parseDouble( bmSpecFR.getSpec( this.m_kt, "FrequencyEleIn" ) );
    }
}

// 力率
this.powerFactor = spec.getSpecValue( this.SPEC_PowerFactor, 0.8 );

//*****************************************************************************
// 発電時　　定格冷房入力(ガス)
this.out_GASc_S_ari = spec.getSpecValue( this.SPEC_out_GASc_S_ari, 0. );

// 発電なし　定格冷房入力(ガス)
this.out_GASc_S_nasi = spec.getSpecValue( this.SPEC_out_GASc_S_nasi, 0. );

// 発電時　　定格冷房入力(電力)
this.out_PPEc_S_ari = spec.getSpecValue( this.SPEC_out_PPEc_S_ari, 0. );

// 発電なし　定格冷房入力(電力)
this.out_PPEc_S_nasi = spec.getSpecValue( this.SPEC_out_PPEc_S_nasi, 0. );

// 発電時　　定格暖房入力(ガス)
this.out_GASh_S_ari = spec.getSpecValue( this.SPEC_out_GASh_ari, 0. );

// 発電なし　定格暖房入力(ガス)
this.out_GASh_S_nasi = spec.getSpecValue( this.SPEC_out_GASh_nasi, 0. );

// 発電時　　定格暖房入力(電力)
this.out_PPEh_S_ari = spec.getSpecValue( this.SPEC_out_PPEh_S_ari, 0. );

// 発電なし　定格暖房入力(電力)
this.out_PPEh_S_nasi = spec.getSpecValue( this.SPEC_out_PPEh_S_nasi, 0. );

//冷房定格発電電力
this.out_ELEc_dy = spec.getSpecValue( this.SPEC_out_ELEc_dy, 0. );

//冷房最大発電電力
this.out_ELEc_dymax = spec.getSpecValue( this.SPEC_out_ELEc_dymax, 0. );

//暖房定格発電電力
this.out_ELEh_dy = spec.getSpecValue( this.SPEC_out_ELEh_dy, 0. );

//暖房最大発電電力
this.out_ELEh_dymax = spec.getSpecValue( this.SPEC_out_ELEh_dymax, 0. );


//*************************************************************

// isGVisibleを取得
this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

// isRecordを取得
this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

//低負荷領域の計算方法
this.calcTypeLowerRangeLoad = spec.getSpecValue( this.SPEC_CalcTypeLowerRangeLoad, "0_発停運転" );
//calcTypeLowerRangeLoad
//  0_発停運転
//  1_下限入力値固定
//  2_下限COP値固定
//  3_下限入力値と中間切片
if( this.calcTypeLowerRangeLoad.equals( "0_発停運転" )){
    this.num_calcTypeLowerRangeLoad = 0;
}else if(this.calcTypeLowerRangeLoad.equals( "1_下限入力値固定" )){
    this.num_calcTypeLowerRangeLoad = 1;
}else if(this.calcTypeLowerRangeLoad.equals( "2_下限COP値固定" )){
    this.num_calcTypeLowerRangeLoad = 2;
}else if(this.calcTypeLowerRangeLoad.equals( "3_下限入力値と中間切片" )){
    this.num_calcTypeLowerRangeLoad = 3;
}else{
    this.num_calcTypeLowerRangeLoad = 0;
}
```

```
//入力データの書き換え


// tabata以下ガスに変更
if ( out_Qc_S < 0.001){
   fmc = 0.0;
}else{
   fmc = out_Qc_C / out_Qc_S;
}

//if( m_ty.equals("O_GHP_BM_標準_冷暖切替201006")){
if( this.isStandard || this.isStandardCHFree ){
   if (out_GASc_S < 0.001 ){
      pmc = 0.0;
   }else{
      pmc = out_GASc_C / out_GASc_S;
   }
   if (out_GASh_S < 0.001 ){
      pmh = 0.0;
   }else{
      pmh = out_GASh_C / out_GASh_S;
   }
}else{
   if (out_GASc_S_nasi < 0.001 ){
      pmc = 0.0;
   }else{
      pmc = out_GASc_C / out_GASc_S_nasi;
   }
   if (out_GASh_S_nasi < 0.001 ){
      pmh = 0.0;
   }else{
      pmh = out_GASh_C / out_GASh_S_nasi;
   }
}


if ( out_Qh_S < 0.001 ){
   fmh = 0.0;
}else{
   fmh = out_Qh_C / out_Qh_S;
}

if ( pmc > 0.9999 ) {
   pmc = 0.0;
   fmc = 0.0;
}
if ( pmh > 0.9999 ) {
   pmh = 0.0;
   fmh = 0.0;
}


//isAdjust2012 = "台数を調整する";//
this.isAdjust2012 = spec.getSpecValue( this.SPEC_isAdjust2012, false );
//自動調整する場合は０％まで運転とする。20200108
if( this.isAdjust2012 ) {
   this.in_run_stop = 0;
}

//仮設調整
this.fRate_airEAc = this.fRate_airEA;
this.fRate_airEAh = this.fRate_airEA;

// 調整の計算ステップ数
this.numAdjustSteps = spec.getSpecValue( this.SPEC_NumAdjustSteps, 12 );

this.rcList = new LinkedList<Double>();
this.rhList = new LinkedList<Double>();
for( int i=0; i<this.numAdjustSteps; i++ ){
   this.rcList.add( 0. );
```

```
            this.rhList.add( 0. );
        }

        this.out_Qc_S_1   = this.out_Qc_S;//= "定格冷房能力";//kW
        this.out_Qc_C_1   = this.out_Qc_C;//= "中間冷房能力";//kW
        this.out_PPEc_S_1 = this.out_PPEc_S;//= "定格冷房入力(電力)";//kW
        //this.out_PPEc_C_1 = this.out_PPEc_C;//= "中間冷房入力(電力)";//kW
        this.out_Qh_S_1   = this.out_Qh_S;//= "定格暖房能力";//kW
        this.out_Qh_C_1   = this.out_Qh_C;//= "中間暖房能力";//kW
        this.out_Qh_L_1   = this.out_Qh_L;//= "低温暖房能力";//kW
        this.out_PPEh_S_1 = this.out_PPEh_S;//= "定格暖房入力(電力)";//kW
        //this.out_PPEh_C_1 = this.out_PPEh_C;//= "中間暖房入力(電力)";//kW
        //this.out_PPEh_L_1 = this.out_PPEh_L;//= "低温暖房入力(電力)";//kW

        // tabata追加
        this.out_GASc_S_1   = this.out_GASc_S;// = "定格冷房入力(ガス)";//kW
        this.out_GASc_C_1   = this.out_GASc_C;// = "中間冷房入力(ガス)";//kW
        this.out_GASh_S_1   = this.out_GASh_S;// = "定格暖房入力(ガス)";//kW
        this.out_GASh_C_1   = this.out_GASh_C;// = "中間暖房入力(ガス)";//kW
        this.out_GASh_L_1   = this.out_GASh_L;// = "低温暖房入力(ガス)";//kW

        //tabata20100301
        this.out_GASc_S_ari_1  = this.out_GASc_S_ari;// = "発電時の定格冷房入力(ガス)";//kW
        this.out_GASc_S_nasi_1 = this.out_GASc_S_nasi;// = "発電なしの定格冷房入力(ガス)";//kW
        this.out_PPEc_S_ari_1  = this.out_PPEc_S_ari;// = "発電時の定格冷房入力(電気)";//kW
        this.out_PPEc_S_nasi_1 = this.out_PPEc_S_nasi;// = "発電なしの定格冷房入力(電気)";//kW
        this.out_GASh_S_ari_1  = this.out_GASh_S_ari;// = "発電時の定格暖房入力(ガス)";//kW
        this.out_GASh_S_nasi_1 = this.out_GASh_S_nasi;// = "発電なしの定格暖房入力(ガス)";//kW
        this.out_PPEh_S_ari_1  = this.out_PPEh_S_ari;// = "発電時の定格暖房入力(電気)";//kW
        this.out_PPEh_S_nasi_1 = this.out_PPEh_S_nasi;// = "発電なしの定格暖房入力(電気)";//kW

        this.out_ELEc_dy_1    = this.out_ELEc_dy;//冷房定格発電電力
        this.out_ELEc_dymax_1 = this.out_ELEc_dymax;//冷房最大発電電力
        this.out_ELEh_dy_1    = this.out_ELEh_dy;//暖房定格発電電力
        this.out_ELEh_dymax_1 = this.out_ELEh_dymax;//暖房最大発電電力
    }

    @Override
    public void initialize(IBestStateMessage stateNodes,
            IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
            IBestDomainSchedule schedule) {
        //System.out.println(this.moduleName + " BuillMultiOutイニシャライズ");
        // 状態ノードを受け取る
        super.sm = stateNodes;
        // 制御ノードを受け取る
        super.cm = commandNodes;
        // 記録ノードを受け取る
        super.rm = recordNodes;

        // eleIn
        this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

        // eleOut
        this.eleOut = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleOut );

        // tabata追加
        // gasIn
        this.gasIn = BestGas.bindnode( super.transferMapState, super.sm, this.S_NODE_gasIn );

        this.airInOA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInOA );

        //private final String S_NODE_airOutEA   = "LO_airOutEA";//排気
        this.airOutEA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutEA );

        //private final String S_NODE_valInLine  = "LO_valInLine";  //単線
        if( super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine )) != null ){

            //System.out.println( " BM_out S_NODE_valInLine登録済み" );

            this.rmlinemap
            =  (Map<String, BM_EHPdata>) super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine));
```

```
        if( this.rmlinemap.size() == 0 ){
            //
            //System.out.println( "rmlinemap のサイズが０ ");
            this.RMdata = new BM_EHPdata();
            this.RMdata.set_path( this.path );
            this.RMdata.set_kiki( this.kiki_file );
            this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
            this.RMdata.setairInHS( this.airInOA );//20120824nino
            this.rmlinemap.put( "Parent", this.RMdata );

            super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );

        }else{
            this.num = 0;
            Set<String> LineEntry = this.rmlinemap.keySet();
            for(Iterator<String> i = LineEntry.iterator();i.hasNext();){
                String key = (String) i.next();
                //          System.out.println(" name="+name+" key="+key);
                this.RMdata = this.rmlinemap.get(key);
                this.rmnm[this.num]=key;
                //          System.out.println("key="+key+" 0="+rmdata[0]+" 1="+rmdata[1]+" 2="+rmdata[2]+"
3="+rmdata[3]+" 4="+rmdata[4]+" 5="+rmdata[5]+" 6="+rmdata[6]);
                this.num++;
            }
            //
            //20150409
            double sum_in_Qc_S = 0;
            double sum_in_Qh_S = 0;
            double sum_in_dai = 0;
            //出力設定
            for(int i=0; i<this.num; i++){
                //System.out.println( "kiki_file Name = " +kiki_file );
                this.RMdata = this.rmlinemap.get( this.rmnm[i] );
                this.RMdata.set_kiki(kiki_file);
                this.RMdata.set_path(this.path);
                this.RMdata.setCOOLandHEAT( this.isCOOLandHEATout );
                this.RMdata.setairInHS( this.airInOA );//20120824nino
                rmlinemap.put(rmnm[i],this.RMdata);
                //20150409
                sum_in_Qc_S += this.RMdata.get_in_Qc_S();
                sum_in_Qh_S += this.RMdata.get_in_Qh_S();
                sum_in_dai += this.RMdata.get_in_dai();
            }
            //
            //System.out.println( "rmlinemap のサイズが "+this.num );
            this.RMdata = new BM_EHPdata();
            this.RMdata.set_path( this.path );
            this.RMdata.set_kiki( this.kiki_file );
            this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
            this.RMdata.setairInHS( this.airInOA );//20120824nino
            //20150409
            this.RMdata.set_in_Qc_S( sum_in_Qc_S );
            this.RMdata.set_in_Qh_S( sum_in_Qh_S );
            this.RMdata.set_in_dai( sum_in_dai );

            this.rmlinemap.put( "Parent", this.RMdata );

            //20150409
        /*  BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
            + "接続定格冷却容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qc_S + "[W]("
                  +AirFormat.df_3( sum_in_Qc_S / this.out_Qc_S ) + "倍)" );
            BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
            + "接続定格加熱容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qh_S + "[W]("
                  +AirFormat.df_3( sum_in_Qh_S / this.out_Qh_S ) + "倍)" );
            BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_initialize)["+this.name+"]"
            + "接続台数="+AirFormat.df_0( sum_in_dai )+"[台]" );
         */
            //接続容量、台数チェック
            this.RMdata.checkConnect( this.out_Qc_S, this.out_Qh_S, sum_in_Qc_S, sum_in_Qh_S, sum_in_dai,
                  this.moduleName, "initialize", this.name, this.isRecord, this.message );

            super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );
```

```
            this.dbflg = 1;
         }
      }else{

         //System.out.println( " BM_out S_NODE_valInLine未登録" );

         //System.out.println( this.moduleName + ">>Warning<< rmlinemap is null !!" );
         message.append( "(W)単線の接続なし→作成");
         this.rmlinemap = new HashMap<String,BM_EHPdata>();
         //
         this.RMdata = new BM_EHPdata();
         this.RMdata.set_path( this.path );
         this.RMdata.set_kiki( this.kiki_file );
         this.RMdata.setCOOLandHEAT(this.isCOOLandHEATout );
         this.RMdata.setairInHS( this.airInOA );//20120824nino
         this.rmlinemap.put( "Parent", this.RMdata );

         super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine ), this.rmlinemap );
      }

      // グラフ表示の準備
      if (this.isGVisible) {
         this.gBMout = new GraphJFrameBuilMultiGHPOut_S20101212(this.name, 300,
               this.out_Qc_S * 1.5, 0);
      }


      this.filenames[0] = this.kiki_file;
      this.equipmentName = this.kiki_file;
      this.pacDB = new PACDBManager( this.path, this.filenames );
      this.pacDB.setEquipmentName( this.equipmentName );

   }

   /**
    * ビルマルチ計算及び結果出力/各室内機よりも先に動きます。
    */
   @Override
   public void outputs() {

      if (null == super.cm || null == super.sm)
         return;

      if( this.dbflg == 0 ){

         //System.out.println( "    BM_out  outputsで  室側データの設定" );

         //System.out.println( " Parent path="+ this.rmlinemap.get( "Parent" ).get_path() );

         if( super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine )) != null ){
            this.rmlinemap
            =  (Map<String, BM_EHPdata>) super.sm.getState( super.getConnectionNode( this.S_NODE_valInLine));
            //
            this.num = 0;
            Set<String> LineEntry = this.rmlinemap.keySet();
            for(Iterator<String> i = LineEntry.iterator();i.hasNext();){
               String key = (String) i.next();
               //          System.out.println(" name="+name+" key="+key);
               this.RMdata = this.rmlinemap.get(key);
               this.rmnm[this.num]=key;
               //          System.out.println("key="+key+" 0="+rmdata[0]+" 1="+rmdata[1]+" 2="+rmdata[2]+"
3="+rmdata[3]+" 4="+rmdata[4]+" 5="+rmdata[5]+" 6="+rmdata[6]);
               this.num++;
            }
            //20150409
            double sum_in_Qc_S = 0;
            double sum_in_Qh_S = 0;
            double sum_in_dai = 0;
            //出力設定
            for(int i=0; i<this.num; i++){
               //System.out.println( "kiki_file Name = " +kiki_file );
               this.RMdata = this.rmlinemap.get( this.rmnm[i] );
```

```
            this.RMdata.set_kiki(kiki_file);
            this.RMdata.set_path(this.path);
            this.RMdata.setCOOLandHEAT( this.isCOOLandHEATout );
            this.RMdata.setairInHS( this.airInOA );//20120824nino
            rmlinemap.put(rmnm[i],this.RMdata);
            //20150409
            sum_in_Qc_S += this.RMdata.get_in_Qc_S();
            sum_in_Qh_S += this.RMdata.get_in_Qh_S();
            sum_in_dai += this.RMdata.get_in_dai();
        }
        //
        //20150409
        this.RMdata = this.rmlinemap.get( "Parent" );
        //20150409
        this.RMdata.set_in_Qc_S( sum_in_Qc_S );
        this.RMdata.set_in_Qh_S( sum_in_Qh_S );
        this.RMdata.set_in_dai( sum_in_dai );

        this.rmlinemap.put( "Parent", this.RMdata );

        //20150409
/*      BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
        + "接続定格冷却容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qc_S + "[W](" 
              +AirFormat.df_3( sum_in_Qc_S / this.out_Qc_S ) + "倍)" );
        BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
        + "接続定格加熱容量="+AirFormat.df_1( sum_in_Qc_S )+"[W]室外機定格容量=" + this.out_Qh_S + "[W](" 
              +AirFormat.df_3( sum_in_Qh_S / this.out_Qh_S ) + "倍)" );
        BestLogger.info( AirSystemControl.getTimeStr()+"("+this.moduleName+"_outputs)["+this.name+"]"
        + "接続台数="+AirFormat.df_0( sum_in_dai )+"[台]" );
*/      //
        //接続容量、台数チェック
        this.RMdata.checkConnect( this.out_Qc_S, this.out_Qh_S, sum_in_Qc_S, sum_in_Qh_S, sum_in_dai,
              this.moduleName, "outputs", this.name, this.isRecord, this.message );

    }
    this.dbflg = 1;

}

// message.setLength(0);
// ノード接続毎回読込み
this.airInOA = (BestAir) super.sm.getState(super.getConnectionNode(this.S_NODE_airInOA));
this.swcIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcIn));
this.modIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modIn));

//mState = this.swcIn * this.modIn;

//初期化
this.Tc_Loadout = 0;
this.Th_Loadout = 0;

//
if( Airswc.isOFF( this.swcIn )){
    //停止
    this.mState = 0;
}else if( Airswc.isON( this.swcIn )){
    //運転
    if( Airmod.isCOOL( this.modIn ) && Airmod.isHEAT( this.modIn ) ){
        //冷暖同時
        if( this.isCOOLandHEATout ){
            this.mState = 11;
        }else{
            //冷暖同時機種で無い場合は冷房運転とする
            this.mState = 1;
            this.message.append( "(W)冷暖同時機種ではないので冷房運転で計算" );
        }
    }else if( Airmod.isCOOL( this.modIn )){
        //冷房運転
        this.mState = 1;
    }else if( Airmod.isHEAT( this.modIn )){
        //暖房運転
        this.mState = 2;
```

```
      }else if( Airmod.isVENTILATE( this.modIn )){
         //換気モード
         this.mState = 3;
      }else{
         //停止
         this.mState = 0;
      }
   }


   /**
    * 入力
    */
   // 外気乾球温度設定
   this.DBoa = this.airInOA.getTempDB();

   // 外気絶対湿度を設定
   this.XGoa = this.airInOA.getHumi();

   // 外気湿球温度を設定
   this.WBoa = Psychrometrics.FNWbtx( this.DBoa, this.XGoa );

   if( this.WBoa<-9000.0){
      this.WBoa =WB_cal( this.DBoa, this.XGoa );
   }

   this.rmlinemap = (Map<String, BM_EHPdata>) super.sm.getState(super
         .getConnectionNode(this.S_NODE_valInLine));

   //データのクリア
   this.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
   this.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
   this.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
   this.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
   this.DBra_max = -100.0;
   this.WBra_min = 1000.0;
   this.Lratemin = 100.0;

   double dmy1 = 100.0;

   // 室内機データの読込
   for (int i = 0; i < this.num; i++) {
      this.RMdata = this.rmlinemap.get( this.rmnm[i]);
      dmy1 = this.RMdata.get_L_rate();
      if (this.Lratemin > dmy1){
         this.Lratemin = dmy1;
      }
      dmy1 = this.RMdata.get_R_wb();
      if (this.WBra_min > dmy1 && this.RMdata.get_R_Tc_load()>0 ){
         this.WBra_min = dmy1;
      }
      dmy1 = this.RMdata.get_R_db();
      if (this.DBra_max < dmy1 && this.RMdata.get_R_Th_load()>0 ){
         this.DBra_max = dmy1;
      }
      this.allSc_heat += this.RMdata.get_R_Sc_load();//室内機処理熱量（冷却顕熱）[W]
      this.allTc_heat += this.RMdata.get_R_Tc_load();//室内機処理熱量（冷却全熱）[W]

      //
      this.allSh_heat += this.RMdata.get_R_Sh_load();//室内機処理熱量（加熱顕熱）[W]
      this.allTh_heat += this.RMdata.get_R_Th_load();//室内機処理熱量（加熱全熱）[W]

   }

   this.WBra = this.WBra_min;
   this.DBra = this.DBra_max;

   //*************************----------------
   if( this.isAdjust2012 ){
      double rc;
      double c_OutTemp;
      double rh;
```

```
        double h_OutTemp;

        //処理可能熱量　仮計算
        if( this.allTc_heat==0 && this.allTh_heat==0 ){
           //冷却・加熱要求なし
           c_OutTemp = this.out_Qc_S_1;
           h_OutTemp = this.out_Qh_S_1;
        }else if( this.allTh_heat == 0 ){
           //冷却要求のみ coolingOnly_cal()
           c_OutTemp = this.pacDB.getKcta( this.DBoa) * this.pacDB.getKcti( this.WBra) * this.Lratemin *
this.out_Qc_S_1;//仮計算　室外機処理可能熱量（冷却全熱）[W]
           h_OutTemp = this.out_Qh_S_1;
        }else if( this.allTc_heat == 0 ){
           //加熱要求のみ heatingOnly_cal()
           c_OutTemp = this.out_Qc_S_1;
           if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L_1 > 0.0001) { //低温入力の場合
              h_OutTemp  = this.pacDB.getKhta( this.WBoa) * this.pacDB.getKhti( this.DBra) * this.Lratemin *
this.out_Qh_L_1;//仮計算　室外機処理可能熱量（加熱全熱）[W]
           }else{
              h_OutTemp  = this.pacDB.getKhta( this.WBoa) * this.pacDB.getKhti( this.DBra) * this.Lratemin *
this.out_Qh_S_1;//仮計算　室外機処理可能熱量（加熱全熱）[W]
           }
        }else if( this.allTc_heat >= this.allTh_heat ){
           //冷却主体運転 cooling_and_heating_cal()
           c_OutTemp = this.pacDB.getKcmta( this.DBoa) * this.pacDB.getKcti( this.WBra) * this.Lratemin *
this.out_Qc_S_1;//仮計算　室外機処理可能熱量（冷却全熱）[W]
           h_OutTemp = this.pacDB.getKhti( this.DBra ) * this.Lratemin * this.out_Qh_S_1;//室外機処理可能熱量（加熱全
熱）[W]
        }else{
           //加熱主体運転 heating_and_cooling_cal()
           c_OutTemp = this.pacDB.getKcti( this.WBra) * this.Lratemin * out_Qc_S_1;//室外機処理可能熱量（冷却全熱）[W]

           if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L_1 > 0.0001) { //低温入力の場合
              h_OutTemp  = this.pacDB.getKhta( this.WBoa ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin *
out_Qh_L_1;//室外機処理可能熱量（加熱全熱）[W]
              double Rch = this.allTc_heat / h_OutTemp / this.out_rhAdjust;
              double deltaWB = this.pacDB.getKdwb( Rch );
              //補正暖房容量
              h_OutTemp  = this.pacDB.getKhmta( this.WBoa + deltaWB ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin
* out_Qh_L_1;//室外機処理可能熱量（加熱全熱）[W]
           }else{
              h_OutTemp  = this.pacDB.getKhta( this.WBoa ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin *
out_Qh_S_1;//室外機処理可能熱量（加熱全熱）[W]
              double Rch = this.allTc_heat / h_OutTemp / this.out_rhAdjust;
              double deltaWB = this.pacDB.getKdwb( Rch );
              //補正暖房容量
              h_OutTemp  = this.pacDB.getKhmta( this.WBoa + deltaWB ) * this.pacDB.getKhti( this.DBra ) * this.Lratemin
* out_Qh_S_1;//室外機処理可能熱量（加熱全熱）[W]
           }
        }

        //冷却能力調整
        if( c_OutTemp == 0 ){
           //rc = this.allTc_heat / this.out_Qc_S;
           rc = this.allTc_heat / this.out_Qc_S_1;
        }else{
           rc = this.allTc_heat / c_OutTemp;
        }

        if( rc > 1.1 ){
        // rc = 1.1;
        }

        //加熱能力調整
        if( h_OutTemp == 0 ){
           //rh = this.allTh_heat / this.out_Qh_S;
           rh = this.allTh_heat / this.out_Qh_S_1;
        }else{
           rh = this.allTh_heat / h_OutTemp;
        }

        if( rh > 1.1 ){
```

```
// rh = 1.1;
}

//*************************----------------
if( true ){
   //移動平均の最大値で調整していく
   this.rcList.removeLast();
   this.rcList.addFirst( rc );
   //
   double sum_rc = 0;
   for( int i=0; i<this.numAdjustSteps; i++ ){
      sum_rc += this.rcList.get( i );
   }
   this.ave_rcAdjust = sum_rc / this.numAdjustSteps;
   //
   if( sum_rc > this.max_rcAdjust * this.numAdjustSteps ){
      this.max_rcAdjust = sum_rc / this.numAdjustSteps;

      this.out_rcAdjust = this.max_rcAdjust;
      //this.in_run_stop_Num = this.in_run_stop / this.out_rcAdjust;

      this.out_Qc_S   = this.out_Qc_S_1    * this.out_rcAdjust;//= "定格冷房能力";//kW
      this.out_Qc_C   = this.out_Qc_C_1    * this.out_rcAdjust;//= "中間冷房能力";//kW
      this.out_PPEc_S = this.out_PPEc_S_1  * this.out_rcAdjust;//= "定格冷房入力(電力)";//kW
      //this.out_PPEc_C_1 = this.out_PPEc_C_1    * this.out_rcAdjust;//= "中間冷房入力(電力)";//kW

      // tabata追加
      this.out_GASc_S   = this.out_GASc_S_1 * this.out_rcAdjust;// = "定格冷房入力(ガス)";//kW
      this.out_GASc_C   = this.out_GASc_C_1 * this.out_rcAdjust;// = "中間冷房入力(ガス)";//kW

      //tabata20100301
      this.out_GASc_S_ari  = this.out_GASc_S_ari_1  * this.out_rcAdjust;// = "発電時の定格冷房入力(ガス)";//kW
      this.out_GASc_S_nasi = this.out_GASc_S_nasi_1 * this.out_rcAdjust;// = "発電なしの定格冷房入力(ガス)";//kW
      this.out_PPEc_S_ari  = this.out_PPEc_S_ari_1  * this.out_rcAdjust;// = "発電時の定格冷房入力(電気)";//kW
      this.out_PPEc_S_nasi = this.out_PPEc_S_nasi_1 * this.out_rcAdjust;// = "発電なしの定格冷房入力(電気)";//kW

      this.out_ELEc_dy    = this.out_ELEc_dy_1    * this.out_rcAdjust;//冷房定格発電電力
      this.out_ELEc_dymax = this.out_ELEc_dymax_1 * this.out_rcAdjust;//冷房最大発電電力

      this.fRate_airEAc = this.fRate_airEA * this.out_rcAdjust;
   }
}


if( true ){
   //移動平均の最大値で調整していく
   this.rhList.removeLast();
   this.rhList.addFirst( rh );
   //
   double sum_rh = 0;
   for( int i=0; i<this.numAdjustSteps; i++ ){
      sum_rh += this.rhList.get( i );
   }
   this.ave_rhAdjust = sum_rh / this.numAdjustSteps;
   //
   if( sum_rh > this.max_rhAdjust * this.numAdjustSteps ){
      this.max_rhAdjust = sum_rh / this.numAdjustSteps;

      this.out_rhAdjust = this.max_rhAdjust;
      //this.in_run_stop_Num = this.in_run_stop / this.out_rcAdjust;

      this.out_Qh_S   = this.out_Qh_S_1    * this.out_rhAdjust;//= "定格暖房能力";//kW
      this.out_Qh_C   = this.out_Qh_C_1    * this.out_rhAdjust;//= "中間暖房能力";//kW
      this.out_Qh_L   = this.out_Qh_L_1    * this.out_rhAdjust;//= "低温暖房能力";//kW
      this.out_PPEh_S = this.out_PPEh_S_1  * this.out_rhAdjust;//= "定格暖房入力(電力)";//kW
      //this.out_PPEh_C_1 = this.out_PPEh_C_1    * this.out_rhAdjust;//= "中間暖房入力(電力)";//kW
      //this.out_PPEh_L_1 = this.out_PPEh_L_1    * this.out_rhAdjust;//= "低温暖房入力(電力)";//kW

      this.out_GASh_S   = this.out_GASh_S_1 * this.out_rhAdjust;// = "定格暖房入力(ガス)";//kW
      this.out_GASh_C   = this.out_GASh_C_1 * this.out_rhAdjust;// = "中間暖房入力(ガス)";//kW
      this.out_GASh_L   = this.out_GASh_L_1 * this.out_rhAdjust;// = "低温暖房入力(ガス)";//kW
```

```
            this.out_GASh_S_ari  = this.out_GASh_S_ari_1  * this.out_rhAdjust;// = "発電時の定格暖房入力(ガス)";//kW
            this.out_GASh_S_nasi = this.out_GASh_S_nasi_1 * this.out_rhAdjust;// = "発電なしの定格暖房入力(ガス)";//kW
            this.out_PPEh_S_ari  = this.out_PPEh_S_ari_1  * this.out_rhAdjust;// = "発電時の定格暖房入力(電気)";//kW
            this.out_PPEh_S_nasi = this.out_PPEh_S_nasi_1 * this.out_rhAdjust;// = "発電なしの定格暖房入力(電気)";//kW

            this.out_ELEh_dy    = this.out_ELEh_dy_1    * this.out_rhAdjust;//暖房定格発電電力
            this.out_ELEh_dymax = this.out_ELEh_dymax_1 * this.out_rhAdjust;//暖房最大発電電力

            this.fRate_airEAh = this.fRate_airEA* this.out_rhAdjust;
         }
      }
      //*************************----------------


      //調整補正
/*    if( rc > 1.0 ){
         this.out_Qc_S *= rc;
         this.out_Qc_C *= rc;
         this.out_PPEc_S *= rc;
         this.out_PPEc_C *= rc;
         //
         this.fRate_airEAc *= rc;
      }
*/
      //調整補正
/*    if( rh > 1.0 ){
         this.out_Qh_S *= rh;
         this.out_Qh_C *= rh;
         this.out_Qh_L *= rh;
         this.out_PPEh_S *= rh;
         this.out_PPEh_C *= rh;
         this.out_PPEh_L *= rh;
         //
         this.fRate_airEAh *= rh;
      }
*/
      this.out_Qc_Adjust = this.out_Qc_S;
      this.out_Qh_Adjust = this.out_Qh_S;

      //System.out.println( "BM out  out_Qc_Adjust=" +out_Qc_Adjust+"  out_Qh_Adjust="+out_Qh_Adjust);
   }


   //*************************----------------


   switch (mState) {
   case 3://20120313nino
      //換気
      //室外機は停止とする
   case 0:
      // 停止
      this.message.append("(C)停止");
      this.calc_Stop();
      break;
   case 1:
      // 冷房
      if( this.out_Qc_S_1 > 0 ){
         this.message.append("(C)冷房運転");
         coolingOnly_cal();
      }else{
         this.message.append("(C)冷能力=0停止");//20130625
         this.calc_Stop();
      }

      break;
   case 2:
      // 暖房
      if( this.out_Qh_S_1 > 0 ){
         this.message.append("(C)暖房運転");
         heatingOnly_cal();
      }else{
```

```
            this.message.append("(C)暖能力=0停止");//20130625
            this.calc_Stop();
        }

        break;
    case 11:
        //冷暖同時
        this.message.append("(C)冷暖房運転");
        if( this.allTh_heat == 0 && this.out_Qc_S_1 > 0 ){
            //冷房運転
            this.coolingOnly_cal();

        }else if( this.allTc_heat == 0 && this.out_Qh_S_1 > 0){
            //暖房運転
            this.heatingOnly_cal();

        }else if( this.allTc_heat > this.allTh_heat && this.out_Qc_S_1 > 0){
            //冷房主体運転
            //20181017
            if( this.is2018SinseiCHFree ) {
                this.coolingOnly_cal();
            }else {
                this.cooling_and_heating_cal();
            }

        }else if( this.allTh_heat > this.allTc_heat && this.out_Qh_S_1 > 0){
            //暖房主体運転
            //20181017
            if( this.is2018SinseiCHFree ) {
                this.heatingOnly_cal();
            }else {
                this.heating_and_cooling_cal();
            }
        }else{
            this.message.append("(C)能力=0停止");//20130625
            this.calc_Stop();
        }

        break;
    default:
        this.message.append("(C)運転モード？停止");//20130625
        this.calc_Stop();
        //System.out.println(this.moduleName + ">>Error<< onOff*modeが範囲外");
    }

    this.E_PPE    = this.PPE * Math.pow(1 / this.powerFactor / this.powerFactor - 1, 0.5);
    this.E_PPEout = this.PPEout * Math.pow(1 / this.powerFactor / this.powerFactor - 1, 0.5);

    this.eleIn.setActivePower( this.PPE );
    this.eleIn.setReactivePower( this.E_PPE );
    this.eleIn.setPhase( this.phase );
    this.eleIn.setVoltage( this.voltage );
    this.eleIn.setFrequency( this.frequency );
    // tabata追記
    this.gasIn.setWatt( this.GAS + this.GASself + this.GASout );
    this.eleOut.setActivePower( this.PPEout );
    this.eleOut.setReactivePower( this.E_PPEout );
    this.eleOut.setPhase( this.phase );
    this.eleOut.setVoltage( this.voltage );
    this.eleOut.setFrequency(this.frequency);

    // 出力設定
    for (int i = 0; i < this.num; i++) {
        this.RMdata = this.rmlinemap.get( this.rmnm[i]);
        this.RMdata.set_CAPrateC( this.CAPrateC );
        this.RMdata.set_CAPrateH( this.CAPrateH );
        this.rmlinemap.put( this.rmnm[i], this.RMdata );
    }
    super.sm.setState(super.getConnectionNode(this.S_NODE_valInLine), this.rmlinemap );

    super.sm.setState(super.getConnectionNode(this.S_NODE_airOutEA), this.airOutEA );
```

```
        super.sm.setState(super.getConnectionNode(this.S_NODE_eleOut), this.eleOut );

        // グラフデータ追加
        if (this.isGVisible) {
            gBMout.addData(BestTimeManager.getDateWeatherTime(),
                    this.Sc_Loadout, this.Sh_Loadout,
                    this.allTc_heat, this.allTh_heat,
                    this.PPE, this.PPEout, this.GAS,
                    this.DBoa, this.DBra, this.WBoa,
                    this.WBra,
                    this.CAPrateC.doubleValue(), this.CAPrateH.doubleValue());
        }

        // 記録ノード
        if (this.isRecord && super.rm != null) {
            this.record();
        }

        this.message.setLength(0);
    }


    private void calc_Stop(){
        this.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
        this.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
        this.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
        this.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]

        this.PPE = 0.0;
        this.GAS = 0.0;// tabata追記
        this.GASself = 0.0;
        this.GASout = 0.0;

        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;

        this.PPEout = 0.0;
    }

    /**
     * 記録
     */
    private void record() {
        //記録ノード
        if( super.rm != null ){
        if( CheckPrintModule.isPrintMessage ){
            //message
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                    this.RECORD_message, this.name, message.toString() );
        }

        if( CheckPrintModule.isPrintEnergy ){
            //記録ノードに室外機消費電力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID4, this.name, AirFormat.df_2(this.PPE));
            // 記録ノードに室外機消費ガスを設定 //tabata追記
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID7, this.name, AirFormat.df_2(this.GAS));
            // 記録ノードに室外機発電電力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID8, this.name,
AirFormat.df_2(this.PPEout));
            // 記録ノードに室外機消費ガスを設定 //tabata追記
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID9, this.name,
AirFormat.df_2(this.GASself));
            // 記録ノードに室外機消費ガスを設定 //tabata追記
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID10, this.name,
AirFormat.df_2(this.GASout));
        }

        if( CheckPrintModule.isPrintLoad ){
```

```
        //記録ノードに室外機実処理熱量（全熱）を設定
        if( this.Tc_Loadout > 0 ){
           super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT, this.name, AirFormat.df_2(-
this.Tc_Loadout));//室外機冷却実処理熱量/現状#W#熱量
        }else{
           super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT, this.name,
AirFormat.df_2(this.Th_Loadout));//室外機加熱実処理熱量/現状#W#熱量
        }
     }

     if( CheckPrintModule.isPrintStateOut ){
     }

     if( CheckPrintModule.isPrintStateMy ){
        //記録ノードに室内機加熱能力補正比率を設定
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID3c, this.name,
AirFormat.df_2(this.CAPrateC));
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID3h, this.name,
AirFormat.df_2(this.CAPrateH));
        // test
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
              this.RECORD_ID11, this.name, AirFormat.df_3(this.Rc));
        //記録ノードに室内機要求理熱量（顕熱）を設定
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID1c, this.name,
AirFormat.df_2(this.allSc_heat));//室内機冷却要求顕熱量合計#W#熱量
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID1h, this.name,
AirFormat.df_2(this.allSh_heat));//室内機加熱要求顕熱量合計#W#熱量
        //記録ノードに室内機要求熱量（全熱）を設定
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID2c, this.name,
AirFormat.df_2(this.allTc_heat));//室内機冷却要求全熱量合計#W#熱量
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID2h, this.name,
AirFormat.df_2(this.allTh_heat));//室内機加熱要求全熱量合計#W#熱量
        //記録ノードに室外機処理可能熱量（全熱）を設定
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID5c, this.name,
AirFormat.df_2(this.Sc_Loadout));//室外機冷却可能熱量/最大#W#熱量
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID5h, this.name,
AirFormat.df_2(this.Sh_Loadout));//室外機加熱可能熱量/最大#W#熱量
        //記録ノードに室外機実処理熱量（全熱）を設定
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID6c, this.name,
AirFormat.df_2(this.Tc_Loadout));//室外機冷却実処理熱量/現状#W#熱量
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID6h, this.name,
AirFormat.df_2(this.Th_Loadout));//室外機加熱実処理熱量/現状#W#熱量
     }

     if( CheckPrintModule.isPrintStateIn ){
        //記録ノードに室外機空気入口DBを設定
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DB_airInHS, this.name,
AirFormat.df_2(this.airInOA.getTempDB()) );
        //記録ノードに室外機空気入口WBを設定
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_WB_airInHS, this.name,
AirFormat.df_2(this.airInOA.getTempWB()) );
        //記録ノードに室外機空気入口流量を設定
        super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_M_airInHS, this.name,
AirFormat.df_2(this.airInOA.getFlowRate()) );
     }

     if( CheckPrintModule.isPrintAdjust ){
        //
        if( this.isAdjust2012 ){
           //記録ノードに室外機調整能力を設定
           super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qc_Adjust, this.name,
this.out_Qc_Adjust );//室外機調整能力[W]
           super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qh_Adjust, this.name,
this.out_Qh_Adjust );//室外機調整能力[W]
           //記録ノードに室外機調整能力を設定
           //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rcAdjust", this.name, this.out_rcAdjust );//
室外機調整率[-]
           //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rhAdjust", this.name, this.out_rhAdjust );//
室外機調整率[-]
        }
     }
```

```
    }
}

/**
 * ビルマルチ計算/各室内機の集計後
 */

@Override
public void update() {
    double rCode0 = 1.;
    double rCode1 = 1.;

    //能力補正係数のうち室外機側分を求める
    //機種によっては次のステップで室内機側の能力設定に使用することがある

    //CoolingOnly
    rCode1 = this.pacDB.getKcta( this.DBoa);
    for( int i=0; i<this.num; i++ ){//20110623nino
        this.RMdata = this.rmlinemap.get( this.rmnm[i] );
        this.RMdata.set_out_c_CAPrevision( rCode0 * rCode1 );
    }
    //this.RMdata.set_out_c_CAPrevision( rCode0 * rCode1 );

    //Cooling and heating
    //rCode1 = this.pacDB.getKcmta( this.DBoa);//***

    //Heating_only
    rCode1 = this.pacDB.getKhta( this.WBoa);
    for( int i=0; i<this.num; i++ ){//20110623nino
        this.RMdata = this.rmlinemap.get( this.rmnm[i] );
        this.RMdata.set_out_h_CAPrevision( rCode0 * rCode1 );
    }
    //  this.RMdata.set_out_h_CAPrevision( rCode0 * rCode1 );

    //Heating and cooling
     //rCode1 = this.pacDB.getKhta( this.WBoa );

}

public Object viewInternal(TestCommand cmd) {

    ArrayList<Object> result = new ArrayList<Object>();
    result.add(super.sm);
    result.add(super.cm);
    result.add(super.rm);
    // 外部定義
    result.add(this.name);

    result.add(this.out_Qc_S);
    result.add(this.out_Qc_C);
    result.add(this.out_PPEc_S);
    result.add(this.out_PPEc_S_nasi);
    result.add(this.out_PPEc_S_ari);
    // result.add(this.out_PPEc_C);
    result.add(this.out_GASc_S);
    result.add(this.out_GASc_S_nasi);
    result.add(this.out_GASc_S_ari);
    result.add(this.out_GASc_C);
    result.add(this.out_Qh_S);
    result.add(this.out_Qh_C);
    result.add(this.out_Qh_L);
    result.add(this.out_PPEh_S);
    result.add(this.out_PPEh_S_nasi);
    result.add(this.out_PPEh_S_ari);
    // result.add(this.out_PPEh_C);
    // result.add(this.out_PPEh_L);
    result.add(this.out_GASh_S);
    result.add(this.out_GASh_S_nasi);
    result.add(this.out_GASh_S_ari);
    result.add(this.out_GASh_C);
    result.add(this.out_GASh_L);
```

```
        result.add(this.Lp);
        result.add(this.Lh);

        result.add(this.DBoa);
        result.add(this.XGoa);
        result.add(this.DBra);
        result.add(this.XGra);

        return result;
    }

    private void coolingOnly_cal() {
        double rCode0, rCode1, rCode2, rCode3;
        double D_fmc, D_pmc, alph, beta;
        double Rp;

        D_fmc = this.fmc;
        D_pmc = this.pmc;

        //上下限のチェック 20120821nino
        double dbOA = this.DBoa;
        double wbRA = this.WBra;

        if( dbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcta() ) ){
            //外気乾球温度＞上限の時停止
            this.message.append( "(C)外気DB＞上限→停止");
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.GAS = 0.0;
            this.GASself = 0.0;
            this.GASout = 0.0;
            this.PPE = 0.0;
            this.PPEout = 0.0;
            this.airOutEA.setFlowRate( 0. );
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );
            return;
        }
        if( dbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcta() ) ){
            //外気乾球温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)外気DB＜下限→下限値で特性計算");
            dbOA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcta() );
        }
        if( wbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() ) ){
            //室内湿球温度＞上限の時　上限値の特性で運転
            this.message.append( "(C)室WB＞上限→上限値で特性計算");
            wbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
        }
        if( wbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() ) ){
            //室内湿球温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)室WB＜下限→下限値で特性計算");
            wbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
        }

        //室温補正
        //能力補正

        rCode1 = this.pacDB.getKcta( dbOA );// 外気温度補正
        rCode2 = this.pacDB.getKcti( wbRA );// 室温補正
        rCode3 = this.Lratemin;// 室内モジュールで補正(配管長、高低差補正)

        this.Sc_Loadout = rCode1 * rCode2 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

        //容量比
        this.Rc = this.allTc_heat / this.Sc_Loadout;

        this.CAPrateC = 1.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = this.allTc_heat;//20120731
```

```
if ( this.Rc > 1.0) {
   this.CAPrateC = 1.0 / this.Rc;
   this.Rc = 1.0;
   this.Tc_Loadout = this.Sc_Loadout;//21020731
}

//消費
//低負荷領域の計算仕分け:Rh
if( this.Rc < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = this.Rc;
}

alph = this.pacDB.getAlphac( Rp );
beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc ,"GHP" );

if (D_pmc * D_fmc < 0.001){
   beta = 1.0;
}

rCode0 = alph * beta;
rCode1 = this.pacDB.getKchpid( Rp );
rCode2 = this.pacDB.getKcgta( dbOA );
rCode3 = this.pacDB.getKcgti( wbRA );

// "0_GHP_BM_標準_冷暖切替201006"
// "1_GHP_BM_発電機付自己消費型201006"
// "2_GHP_BM_発電機付系統連携型201006"
// "3_GHP_BM_標準_冷暖同時201006"

//if( this.m_ty.equals("0_GHP_BM_標準_冷暖切替201006") || this.m_ty.equals("3_GHP_BM_標準_冷暖同時201006")){
if( this.isStandard || this.isStandardCHFree ){
   this.GAS = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASc_S;
   //低負荷領域の計算仕分け:PPE
   if( this.Rc < this.in_run_stop && Rp > 0 ){
      switch( this.num_calcTypeLowerRangeLoad){
      case 0://  0_発停運転
         this.GAS = 0.0;
         break;

      case 1://  1_下限入力値固定
         //何もしない
         break;

      case 2://  2_下限COP値固定
         this.GAS *= ( this.Rc / Rp );
         break;

      case 3://  3_下限入力値と中間切片
         this.GAS *= ( 0.5 + this.Rc / Rp / 2. );
         break;

      default:
      }
   }else{
      //何もしない
   }
   this.GASself = 0.0;
```

```
    this.GASout = 0.0;
    if( this.isPchoerunK ){//201707
      this.PPE = this.pacDB.getPcoerunK( Rc ) * this.out_PPEc_S;//201707
    }else{
      this.PPE = this.pacDB.getPcoerun( dbOA, this.out_PPEc_S);
    }

  //}else if( this.m_ty.equals("1_GHP_BM_発電機付自己消費型201006")){
  }else if( this.isDynamo ){
    this.GAS = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASc_S_nasi;
    //低負荷領域の計算仕分け:PPE
    if( this.Rc < this.in_run_stop && Rp > 0 ){
      switch( this.num_calcTypeLowerRangeLoad){
      case 0://  0_発停運転
        this.GAS = 0.0;
        break;

      case 1://  1_下限入力値固定
        //何もしない
        break;

      case 2://  2_下限COP値固定
        this.GAS *= ( this.Rc / Rp );
        break;

      case 3://  3_下限入力値と中間切片
        this.GAS *= ( 0.5 + this.Rc / Rp / 2. );
        break;

      default:
      }
    }else{
      //何もしない
    }
    this.GASself = getEleProCool();
    this.GASout = 0.0;
    this.PPE = this.out_PPEc_S_ari;

  //}else if( this.m_ty.equals("2_GHP_BM_発電機付系統連携型201006")){
  }else if( this.isDynamoLink ){
    //ガス消費量W
    double GAS1 = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASc_S_nasi;
    //低負荷領域の計算仕分け:PPE
    if( this.Rc < this.in_run_stop && Rp > 0 ){
      switch( this.num_calcTypeLowerRangeLoad){
      case 0://  0_発停運転
        GAS1 = 0.0;
        break;

      case 1://  1_下限入力値固定
        //何もしない
        break;

      case 2://  2_下限COP値固定
        GAS1 *= ( this.Rc / Rp );
        break;

      case 3://  3_下限入力値と中間切片
        GAS1 *= ( 0.5 + this.Rc / Rp / 2. );
        break;

      default:
      }
    }else{
      //何もしない
    }

    //以下、修正
    //発電効率：(定格時の発電電力)÷(発電時の定格ｶﾞｽ入力-発電なし時の定格ｶﾞｽ入力)
    //double eleEffi = (this.out_ELEc_dy) / (this.out_GASc_S_ari - this.out_GASc_S_nasi);
    //↓
    //発電効率：冷房定格時の　(発電無し時の消費電力 - 発電時の消費電力 + 発電電力(系統出力))
```

```
        //       ÷ (発電時の燃料入力 - 発電無し時の燃料入力)
        double eleEffi = ( this.out_PPEc_S_nasi - this.out_PPEc_S_ari + this.out_ELEc_dy ) / (this.out_GASc_S_ari -
this.out_GASc_S_nasi);

        //System.out.println(eleEffi);

        //自己発電分
        double GAS2 = 0.0;
        if( this.out_PPEc_S_ari<0.5*1000.0 ){//自己消費発電ありの場合
           //発電電力 Gec1(kW)
           double eleAmount = this.out_ELEc_dy - this.pacDB.getPcoerun_nasi( dbOA, this.out_PPEc_S_nasi);
           //発電用ガス消費量 Ggc1(kW)
           GAS2 = eleAmount/eleEffi;
           //消費電力
           this.PPE = this.out_PPEc_S_ari;
        }else{//自己消費発電なしの場合
           GAS2 = 0.0;
           //PPE = this.out_PPEc_S_nasi;
           this.PPE = this.pacDB.getPcoerun_nasi( dbOA, this.out_PPEc_S_nasi);
        }

        //系統連携分
        //発電電力 Gec2(kW)
        this.PPEout = this.pacDB.getGec2( this.Tc_Loadout / this.out_Qc_S ,
this.out_ELEc_dy/1000. ,this.out_ELEc_dymax/1000.);
        this.PPEout = this.PPEout * 1000.;
        //発電用ガス消費量W  Ggc2(kW)
        double GAS3 = this.PPEout / eleEffi;

        this.GAS = GAS1;
        this.GASself = GAS2;
        this.GASout = GAS3;

     }else{
        System.out.println( this.moduleName + " cooling_cal:GHPタイプの選択ミス");
     }

     //airOutEA
     this.airOutEA.setFlowRate( this.fRate_airEA );
     this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPE ) / this.fRate_airEA * 1000. + this.airInOA.getTempDB() );


     if( this.Rc < 0.0001 ){
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.GAS = 0.0;
        this.GASself = 0.0;
        this.GASout = 0.0;
        this.PPE = 0.0;
        this.PPEout = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );

     }else if( this.Rc < this.in_run_stop ){
        switch( this.num_calcTypeLowerRangeLoad){
           case 0://  0_発停運転
              this.CAPrateC = 0.0;
              this.CAPrateH = 0.0;
              this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
              this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
              this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
              this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
              this.GAS = 0.0;
              this.GASself = 0.0;
              this.GASout = 0.0;
              this.PPE = 0.0;
              this.PPEout = 0.0;
              this.airOutEA.setFlowRate( 0. );
```

```
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );
            break;

         case 1://  1_下限入力値固定
            break;

         case 2://  2_下限COP値固定
            break;

         case 3://  3_下限入力値と中間切片
            break;

         default:
      }

   }

}

private void cooling_and_heating_cal(){
   double rCode0,rCode1,rCode2,rCode3;
   double D_fmc,D_pmc,alph,beta;
   double Rp;

   D_fmc = this.fmc;
   D_pmc = this.pmc;

   //上下限のチェック 20120821nino
   double dbOA = this.DBoa;
   double wbRA = this.WBra;

   if( dbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcmta() ) ){
      //外気乾球温度＞上限の時停止
      this.message.append( "(C)外気DB＞上限→停止");
      this.CAPrateC = 0.0;
      this.CAPrateH = 0.0;
      this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
      this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
      this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
      this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
      this.GAS = 0.0;
      this.GASself = 0.0;
      this.GASout = 0.0;
      this.PPE = 0.0;
      this.PPEout = 0.0;
      this.airOutEA.setFlowRate( 0. );
      this.airOutEA.setTempDB( this.airInOA.getTempDB() );
      return;
   }
   if( dbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcmta() ) ){
      //外気乾球温度＜下限の時　下限値の特性で運転
      this.message.append( "(C)外気DB＜下限→下限値で特性計算");
      dbOA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcmta() );
   }
   if( wbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() ) ){
      //室内湿球温度＞上限の時　上限値の特性で運転
      this.message.append( "(C)室WB＞上限→上限値で特性計算");
      wbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
   }
   if( wbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() ) ){
      //室内湿球温度＜下限の時　下限値の特性で運転
      this.message.append( "(C)室WB＜下限→下限値で特性計算");
      wbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
   }

   //室温補正
   //能力補正
   //冷房側　処理熱量
   rCode1 = this.pacDB.getKcmta( dbOA );// 外気温度補正
   rCode2 = this.pacDB.getKcti( wbRA );// 室温補正
   rCode3 = this.Lratemin;// 室内モジュールで補正(配管長、高低差補正)
```

```
this.Sc_Loadout = rCode1 * rCode2 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

//容量比
this.Rc = this.allTc_heat / this.Sc_Loadout;

this.CAPrateC = 1.0;
this.Tc_Loadout = this.allTc_heat;//20120731
if ( this.Rc > 1.0) {
   this.CAPrateC = 1.0 / this.Rc;
   this.Rc = 1.0;
   this.Tc_Loadout = this.Sc_Loadout;//21207310
}

//消費
//低負荷領域の計算仕分け:Rh
if( this.Rc < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = this.Rc;
}

alph = this.pacDB.getAlphac( Rp );
beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc ,"GHP" );

if (D_pmc * D_fmc < 0.001){
   beta = 1.0;
}

rCode0 = alph * beta;
rCode1 = this.pacDB.getKchpid( Rp );
rCode2 = this.pacDB.getKcgta( dbOA );
rCode3 = this.pacDB.getKcgti( wbRA );

// "0_GHP_BM_標準_冷暖切替201006"
// "1_GHP_BM_発電機付自己消費型201006"
// "2_GHP_BM_発電機付系統連携型201006"
// "3_GHP_BM_標準_冷暖同時201006"

//if( this.m_ty.equals("0_GHP_BM_標準_冷暖切替201006") || this.m_ty.equals("3_GHP_BM_標準_冷暖同時201006") ){
if( this.isStandard || this.isStandardCHFree ){
   this.GAS = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASc_S;
   //低負荷領域の計算仕分け:PPE
   if( this.Rc < this.in_run_stop && Rp > 0 ){
      switch( this.num_calcTypeLowerRangeLoad){
      case 0://  0_発停運転
         this.GAS = 0.0;
         break;

      case 1://  1_下限入力値固定
         //何もしない
         break;

      case 2://  2_下限COP値固定
         this.GAS *= ( this.Rc / Rp );
         break;

      case 3://  3_下限入力値と中間切片
         this.GAS *= ( 0.5 + this.Rc / Rp / 2. );
         break;
```

```
            default:
            }
        }else{
            //何もしない
        }
        this.GASself = 0.0;
        this.GASout = 0.0;
        this.PPE = this.pacDB.getPcoerun( dbOA, this.out_PPEc_S);
    //}else if( this.m_ty.equals("1_GHP_BM_発電機付自己消費型201006")){
    }else if( this.isDynamo ){
        this.GAS = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASc_S_nasi;
        //低負荷領域の計算仕分け:PPE
        if( this.Rc < this.in_run_stop && Rp > 0 ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                this.GAS = 0.0;
                break;

            case 1://  1_下限入力値固定
                //何もしない
                break;

            case 2://  2_下限COP値固定
                this.GAS *= ( this.Rc / Rp );
                break;

            case 3://  3_下限入力値と中間切片
                this.GAS *= ( 0.5 + this.Rc / Rp / 2. );
                break;

            default:
            }
        }else{
            //何もしない
        }
        this.GASself = getEleProCool();
        this.GASout = 0.0;
        this.PPE = this.out_PPEc_S_ari;
    //}else if( this.m_ty.equals("2_GHP_BM_発電機付系統連携型201006")){
    }else if( this.isDynamoLink ){
        //ガス消費量W
        double GAS1 = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASc_S_nasi;
        //低負荷領域の計算仕分け:PPE
        if( this.Rc < this.in_run_stop && Rp > 0 ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                GAS1 = 0.0;
                break;

            case 1://  1_下限入力値固定
                //何もしない
                break;

            case 2://  2_下限COP値固定
                GAS1 *= ( this.Rc / Rp );
                break;

            case 3://  3_下限入力値と中間切片
                GAS1 *= ( 0.5 + this.Rc / Rp / 2. );
                break;

            default:
            }
        }else{
            //何もしない
        }

        //以下、修正
        //発電効率：(定格時の発電電力)÷(発電時の定格ｶﾞｽ入力-発電なし時の定格ｶﾞｽ入力)
        //double eleEffi = (this.out_ELEc_dy) / (this.out_GASc_S_ari - this.out_GASc_S_nasi);
        //↓
```

```
        //発電効率：冷房定格時の　(発電無し時の消費電力 - 発電時の消費電力 + 発電電力(系統出力))
        //　　÷ (発電時の燃料入力 - 発電無し時の燃料入力)
        double eleEffi = ( this.out_PPEc_S_nasi - this.out_PPEc_S_ari + this.out_ELEc_dy) / (this.out_GASc_S_ari -
this.out_GASc_S_nasi);

        //System.out.println(eleEffi);

        //自己発電分
        double GAS2 = 0.0;
        if( this.out_PPEc_S_ari<0.5*1000.0){//自己消費発電ありの場合
           //発電電力 Gec1(kW)
           double eleAmount = this.out_ELEc_dy - this.pacDB.getPcoerun_nasi( dbOA, this.out_PPEc_S_nasi);
           //発電用ガス消費量 Ggc1(kW)
           GAS2 = eleAmount/eleEffi;
           //消費電力
           this.PPE = this.out_PPEc_S_ari;
        }else{//自己消費発電なしの場合
           GAS2 = 0.0;
           //PPE = this.out_PPEc_S_nasi;
           this.PPE = this.pacDB.getPcoerun_nasi( dbOA, this.out_PPEc_S_nasi);
        }

        //系統連携分
        //発電電力 Gec2(kW)
        this.PPEout = this.pacDB.getGec2( this.Tc_Loadout / this.out_Qc_S ,
this.out_ELEc_dy/1000. ,this.out_ELEc_dymax/1000.);
        this.PPEout = this.PPEout*1000.;
        //発電用ガス消費量W  Ggc2(kW)
        double GAS3 = this.PPEout/eleEffi;

        this.GAS = GAS1;
        this.GASself = GAS2;
        this.GASout = GAS3;

     }else{
        //System.out.println( this.moduleName + " cooling_cal:GHPタイプの選択ミス");
     }


     //暖房側　処理熱量
     rCode2 = this.pacDB.getKhti( this.DBra );
     rCode3 = this.Lratemin;

     this.Sh_Loadout = rCode2 * rCode3 * this.out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

     //double TT_Loadout = T_Loadout;

     //容量比
     this.Rh = this.allTh_heat / this.Sh_Loadout;

     this.CAPrateH = 1.0;
     this.Th_Loadout = this.allTh_heat;//20120731
     if (this.Rh > 1.0 ) {
        this.CAPrateH = 1.0 / this.Rh;
        this.Rh = 1.0;
        this.Th_Loadout = this.Sh_Loadout;//21020731
     }

     //airOutEA
     this.airOutEA.setFlowRate( this.fRate_airEA );
     this.airOutEA.setTempDB( ( this.Tc_Loadout + this.PPE ) / this.fRate_airEA * 1000. + this.airInOA.getTempDB() );



     if( this.Rc < 0.0001 ){
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.GAS = 0.0;
```

```
        this.GASself = 0.0;
        this.GASout = 0.0;
        this.PPE = 0.0;
        this.PPEout = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );

    }else if( this.Rc < this.in_run_stop ){
        switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                this.CAPrateC = 0.0;
                this.CAPrateH = 0.0;
                this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
                this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
                this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
                this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
                this.GAS = 0.0;
                this.GASself = 0.0;
                this.GASout = 0.0;
                this.PPE = 0.0;
                this.PPEout = 0.0;
                this.airOutEA.setFlowRate( 0. );
                this.airOutEA.setTempDB( this.airInOA.getTempDB() );
                break;

            case 1://  1_下限入力値固定
                break;

            case 2://  2_下限COP値固定
                break;

            case 3://  3_下限入力値と中間切片
                break;

            default:
        }

    }

    //        System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

}


/*
 * 冷房時の発電分ガス消費量の計算 自己消費型
 */
private double getEleProCool() {

    if( this.out_GASc_S_ari == this.out_GASc_S_nasi ){
        return 0;
    }

    //運転時の必要電力
    double epc = pacDB.getEpc(DBoa, this.out_PPEc_S_nasi/1000.);//kW
    epc = epc * 1000.;
    //発電効率：(発電なし時の定格電力-発電時の定格電力)÷(発電時の定格ｶﾞｽ入力-発電なし時の定格ｶﾞｽ入力)
    double eleEffi = (this.out_PPEc_S_nasi - this.out_PPEc_S_ari) / (this.out_GASc_S_ari - this.out_GASc_S_nasi);

    //発電電力W
    //double eleAmount = epc - this.pacDB.getPcoerun( DBoa, this.out_PPEc_S_ari );//this.out_PPEc_S_ari;
    double eleAmount = epc - this.pacDB.getPcoerun_ari( DBoa, this.out_PPEc_S_ari );//20130729

    //System.out.println( "epc=" + epc + "  eleAmount="+ eleAmount );

    if( eleAmount < 0 ){
        eleAmount = 0;
    }

    //発電分のガス消費量
    if( eleEffi == 0 || eleEffi == Double.NaN ){
        return 0;
```

```
    }
    double gas = eleAmount / eleEffi;

    //double hour = BestTimeManager.getDateTime().getHours();
    //if(hour>=21){
    // System.out.println(epc + "¥t" + eleAmount + "¥t" + gas);
    //}

    return gas;
}


/**
 *
 */
private void heatingOnly_cal() {
    double rCode0, rCode1, rCode2, rCode3;
    double D_fmh, D_pmh, alph, beta;
    double Rp;

    D_fmh = this.fmh;
    D_pmh = this.pmh;

    //上下限のチェック 20120821nino
    double wbOA = this.WBoa;
    double dbRA = this.DBra;

    if( wbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() ) ){
        //外気湿球温度＞上限の時　上限値の特性で運転
        this.message.append( "(C)外気WB＞上限→上限値で特性計算");
        wbOA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() );
    }
    if( wbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhta() ) ){
        //外気湿球温度＜下限の時　停止
        this.message.append( "(C)外気WB＜下限→停止");
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.GAS = 0.0;
        this.GASself = 0.0;
        this.GASout = 0.0;
        this.PPE = 0.0;
        this.PPEout = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );
        return;
    }
    if( dbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() ) ){
        //室内乾球温度＞上限の時　上限値の特性で運転
        this.message.append( "(C)室DB＞上限→上限値で特性計算");
        dbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
    }
    if( dbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() ) ){
        //室内乾球温度＜下限の時　下限値の特性で運転
        this.message.append( "(C)室DB＜下限→下限値で特性計算");
        dbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
    }


    rCode1 = this.pacDB.getKhta( wbOA );
    rCode2 = this.pacDB.getKhti( dbRA );
    rCode3 = this.Lratemin;

    this.Sh_Loadout = rCode1 * rCode2 * rCode3 * this.out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

    //double TT_Loadout = T_Loadout;

    this.Rh = this.allTh_heat / this.Sh_Loadout;
    this.CAPrateC = 0.0;
```

```
this.CAPrateH = 1.0;
this.Th_Loadout = this.allTh_heat;//20120731
if (this.Rh > 1.0 ) {
   this.CAPrateH = 1.0 / this.Rh;
   this.Rh = 1.0;
   this.Th_Loadout = this.Sh_Loadout;//20120731
}

//暖房　消費ガス　電力
//低負荷領域の計算仕分け:Rh
if( this.Rh < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = this.Rh;
}

alph = this.pacDB.getAlphah( Rp );
beta = this.pacDB.getBetah( Rp, D_fmh, D_pmh);

if (D_pmh * D_fmh < 0.001)
   beta = 1.0;// 低温入力あった場合の中間比率

rCode0 = alph * beta;
rCode1 = this.pacDB.getKhhpid( Rp );
rCode2 = this.pacDB.getKhgta( wbOA );
rCode3 = this.pacDB.getKhgti( dbRA );

// "0_GHP_BM_標準_冷暖切替201006"
// "1_GHP_BM_発電機付自己消費型201006"
// "2_GHP_BM_発電機付系統連携型201006"
// "3_GHP_BM_標準_冷暖同時201006"


//if( this.m_ty.equals("0_GHP_BM_標準_冷暖切替201006") ||  this.m_ty.equals("3_GHP_BM_標準_冷暖同時201006") ){
if( this.isStandard || this.isStandardCHFree ){
   this.GAS = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASh_S;
   //低負荷領域の計算仕分け:PPE
   if( this.Rh < this.in_run_stop && Rp > 0 ){
      switch( this.num_calcTypeLowerRangeLoad){
      case 0://  0_発停運転
         this.GAS = 0.0;
         break;

      case 1://  1_下限入力値固定
         //何もしない
         break;

      case 2://  2_下限COP値固定
         this.GAS *= ( this.Rh / Rp );
         break;

      case 3://  3_下限入力値と中間切片
         this.GAS *= ( 0.5 + this.Rh / Rp / 2. );
         break;

      default:
      }
   }else{
      //何もしない
```

```
    }
    this.GASself = 0.0;
    this.GASout = 0.0;
    if( this.isPchoerunK ){//201707
       this.PPE = this.pacDB.getPhoerunK( Rh ) * this.out_PPEh_S;//201707
    }else{
       this.PPE = this.pacDB.getPhoerun( wb0A, this.out_PPEh_S, this.Th_Loadout / this.out_Qh_S);//090713バグ修正
    }

 //}else if(m_ty.equals("1_GHP_BM_発電機付自己消費型201006")){
 }else if( this.isDynamo ){
    this.GAS = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASh_S_nasi;
    //低負荷領域の計算仕分け:PPE
    if( this.Rh < this.in_run_stop && Rp > 0 ){
       switch( this.num_calcTypeLowerRangeLoad){
       case 0://  0_発停運転
          this.GAS = 0.0;
          break;

       case 1://  1_下限入力値固定
          //何もしない
          break;

       case 2://  2_下限COP値固定
          this.GAS *= ( this.Rh / Rp );
          break;

       case 3://  3_下限入力値と中間切片
          this.GAS *= ( 0.5 + this.Rh / Rp / 2. );
          break;

       default:
       }
    }else{
       //何もしない
    }
    this.GASself = getEleProHeat();
    this.GASout = 0.0;
    this.PPE = this.out_PPEh_S_ari;

 //}else if(m_ty.equals("2_GHP_BM_発電機付系統連携型201006")){
 }else if( this.isDynamoLink ){
    //ガス消費量
    double GAS1 = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASh_S_nasi;
    //低負荷領域の計算仕分け:PPE
    if( this.Rh < this.in_run_stop && Rp > 0 ){
       switch( this.num_calcTypeLowerRangeLoad){
       case 0://  0_発停運転
          GAS1 = 0.0;
          break;

       case 1://  1_下限入力値固定
          //何もしない
          break;

       case 2://  2_下限COP値固定
          GAS1 *= ( this.Rh / Rp );
          break;

       case 3://  3_下限入力値と中間切片
          GAS1 *= ( 0.5 + this.Rh / Rp / 2. );
          break;

       default:
       }
    }else{
       //何もしない
    }

    //発電効率：(定格時の発電電力)÷(発電時の定格ｶﾞｽ入力-発電なし時の定格ｶﾞｽ入力)
    //double eleEffi = (this.out_ELEh_dy) / (this.out_GASh_S_ari - this.out_GASh_S_nasi);
    //↓
```

```
        //発電効率：暖房定格時の　(発電無し時の消費電力 - 発電時の消費電力 + 発電電力(系統出力))
        //　　÷ (発電時の燃料入力 - 発電無し時の燃料入力)
        double eleEffi = ( this.out_PPEh_S_nasi - this.out_PPEh_S_ari + this.out_ELEh_dy) / (this.out_GASh_S_ari -
this.out_GASh_S_nasi);

        //System.out.println(eleEffi);

        //自己発電分
        double GAS2 = 0.0;
        if( this.out_PPEh_S_ari<0.5*1000.0){//自己消費発電ありの場合
           //発電電力 Geh1(kW)　発電無しとの差分を取る
           double eleAmount = this.out_ELEh_dy - this.pacDB.getPhoerun_nasi( wbOA, this.out_PPEc_S_nasi, this.Th_Loadout
/ this.out_Qh_S);
           //発電用ガス消費量 Ggc1(kW)
           GAS2 = eleAmount/eleEffi;
           //消費電力
           this.PPE = this.out_PPEh_S_ari;
        }else{//自己消費発電なしの場合
           GAS2 = 0.0;
           //PPE = this.out_PPEh_S_nasi;
           this.PPE = this.pacDB.getPhoerun_nasi( wbOA, this.out_PPEh_S_nasi, this.Th_Loadout / this.out_Qh_S);
        }

        //系統連携分
        //発電電力 Geh2(kW)
        this.PPEout = this.pacDB.getGeh2( this.Th_Loadout / this.out_Qh_S,
this.out_ELEh_dy/1000. ,this.out_ELEh_dymax/1000.);
        this.PPEout = this.PPEout*1000.;
        //発電用ガス消費量 Ggh2(kW)
        double GAS3 = this.PPEout / eleEffi;

        this.GAS = GAS1;
        this.GASself = GAS2;
        this.GASout = GAS3;

     }else{
        //System.out.println(moduleName + " heating_cal:GHPタイプの選択ミス");
     }


     //airOutEA
     this.airOutEA.setFlowRate( this.fRate_airEA );
     this.airOutEA.setTempDB( ( -this.Th_Loadout + this.PPE ) / this.fRate_airEA * 1000. + this.airInOA.getTempDB() );


     if( this.Rh < 0.0001 ){
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.GAS = 0.0;
        this.GASself = 0.0;
        this.GASout = 0.0;
        this.PPE = 0.0;
        this.PPEout = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );

     }else if( this.Rh < this.in_run_stop ){
        switch( this.num_calcTypeLowerRangeLoad){
           case 0://  0_発停運転
              this.CAPrateC = 0.0;
              this.CAPrateH = 0.0;
              this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
              this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
              this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
              this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
              this.GAS = 0.0;
              this.GASself = 0.0;
              this.GASout = 0.0;
```

```
            this.PPE = 0.0;
            this.PPEout = 0.0;
            this.airOutEA.setFlowRate( 0. );
            this.airOutEA.setTempDB( this.airInOA.getTempDB() );
            break;

         case 1://  1_下限入力値固定
            break;

         case 2://  2_下限COP値固定
            break;

         case 3://  3_下限入力値と中間切片
            break;

         default:
      }

   }
}


private void heating_and_cooling_cal(){
   double rCode0,rCode1,rCode2,rCode3;
   double D_fmh,D_pmh;
   double D_out_Qc_S;
   double S,alph,beta;
   double deltaWB;
   double Rch;
   double Rp;

   D_fmh = this.fmh;
   D_pmh = this.pmh;

   //上下限のチェック 20120821nino
   double wbOA = this.WBoa;
   double dbRA = this.DBra;

   if( wbOA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() ) ){
      //外気湿球温度＞上限の時　上限値の特性で運転
      this.message.append( "(C)外気WB＞上限→上限値で特性計算");
      wbOA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() );
   }
   if( wbOA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhta() ) ){
      //外気湿球温度＜下限の時　停止
      this.message.append( "(C)外気WB＜下限→停止");
      this.CAPrateC = 0.0;
      this.CAPrateH = 0.0;
      this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
      this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
      this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
      this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
      this.GAS = 0.0;
      this.GASself = 0.0;
      this.GASout = 0.0;
      this.PPE = 0.0;
      this.PPEout = 0.0;
      this.airOutEA.setFlowRate( 0. );
      this.airOutEA.setTempDB( this.airInOA.getTempDB() );
      return;
   }
   if( dbRA > this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() ) ){
      //室内乾球温度＞上限の時　上限値の特性で運転
      this.message.append( "(C)室DB＞上限→上限値で特性計算");
      dbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
   }
   if( dbRA < this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() ) ){
      //室内乾球温度＜下限の時　下限値の特性で運転
      this.message.append( "(C)室DB＜下限→下限値で特性計算");
      dbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
   }
```

```
        //暖房　処理熱量
        //定格暖房容量
       rCode1 = this.pacDB.getKhta( wbOA );
       rCode2 = this.pacDB.getKhti( dbRA );
       rCode3 = this.Lratemin;

       this.Sh_Loadout = rCode1 * rCode2 * rCode3 * this.out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

       //double TT_Loadout = T_Loadout;
       Rch = this.allTc_heat / this.Sh_Loadout;
       deltaWB = this.pacDB.getKdwb( Rch );

       //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode1+" "+rCode2+" "+rCode3);

       //       System.out.println("allTheat="+allTheat+" T_Loadout="+T_Loadout+" Rh="+Rc+" "+rCode1+" "+rCode2+"
"+rCode3);

       //補正暖房容量
        rCode1 = this.pacDB.getKhmta( wbOA + deltaWB );
       this.Sh_Loadout  = rCode1 * rCode2 * rCode3 * this.out_Qh_S;//室外機処理可能熱量（加熱全熱）[W]

       this.Rh = this.allTh_heat / this.Sh_Loadout;
       this.CAPrateH = 1.0;
       this.Th_Loadout = this.allTh_heat;//20120731
       if (this.Rh > 1.0 ) {
          this.CAPrateH = 1.0 / this.Rh;
          this.Rh = 1.0;
          this.Th_Loadout = this.Sh_Loadout;//21020731
       }

       //暖房　消費ガス　電力
       //低負荷領域の計算仕分け:Rh
       if( this.Rh < this.in_run_stop ){
          switch( this.num_calcTypeLowerRangeLoad){
          case 0://  0_発停運転
             Rp = 0;
             break;

          case 1://  1_下限入力値固定
          case 2://  2_下限COP値固定
          case 3://  3_下限入力値と中間切片
             Rp = this.in_run_stop;
             break;

          default:
             Rp = 0;
          }
       }else{
          Rp = this.Rh;
       }

       alph = this.pacDB.getAlphah( Rp );
       beta = this.pacDB.getBetah( Rp, D_fmh, D_pmh);

       if (D_pmh * D_fmh < 0.001){
          beta = 1.0;// 低温入力あった場合の中間比率
       }

       rCode0 = alph * beta;
       rCode1 = this.pacDB.getKhhpid( Rp );
       rCode2 = this.pacDB.getKhmgta( wbOA + deltaWB );
       rCode3 = this.pacDB.getKhgti( dbRA );

       // "0_GHP_BM_標準_冷暖切替201006"
       // "1_GHP_BM_発電機付自己消費型201006"
       // "2_GHP_BM_発電機付系統連携型201006"
       // "3_GHP_BM_標準_冷暖同時201006"

       //if( this.m_ty.equals("0_GHP_BM_標準_冷暖切替201006") || this.m_ty.equals("3_GHP_BM_標準_冷暖同時201006") ){
       if( this.isStandard || this.isStandardCHFree ){
          this.GAS = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASh_S;
```

```
    //低負荷領域の計算仕分け:PPE
    if( this.Rh < this.in_run_stop && Rp > 0 ){
       switch( this.num_calcTypeLowerRangeLoad){
       case 0://  0_発停運転
          this.GAS = 0.0;
          break;

       case 1://  1_下限入力値固定
          //何もしない
          break;

       case 2://  2_下限COP値固定
          this.GAS *= ( this.Rh / Rp );
          break;

       case 3://  3_下限入力値と中間切片
          this.GAS *= ( 0.5 + this.Rh / Rp / 2. );
          break;

       default:
       }
    }else{
       //何もしない
    }
    this.GASself = 0.0;
    this.GASout = 0.0;
    this.PPE = this.pacDB.getPhoerun( wbOA, this.out_PPEh_S, this.Th_Loadout / this.out_Qh_S);//090713バグ修正

 //}else if( this.m_ty.equals("1_GHP_BM_発電機付自己消費型201006")){
 }else if( this.isDynamo ){
    this.GAS = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASh_S_nasi;
    //低負荷領域の計算仕分け:PPE
    if( this.Rh < this.in_run_stop && Rp > 0 ){
       switch( this.num_calcTypeLowerRangeLoad){
       case 0://  0_発停運転
          this.GAS = 0.0;
          break;

       case 1://  1_下限入力値固定
          //何もしない
          break;

       case 2://  2_下限COP値固定
          this.GAS *= ( this.Rh / Rp );
          break;

       case 3://  3_下限入力値と中間切片
          this.GAS *= ( 0.5 + this.Rh / Rp / 2. );
          break;

       default:
       }
    }else{
       //何もしない
    }
    this.GASself = getEleProHeat();
    this.GASout = 0.0;
    this.PPE = this.out_PPEh_S_ari;

 //}else if( this.m_ty.equals("2_GHP_BM_発電機付系統連携型201006")){
 }else if( this.isDynamoLink ){
    //ガス消費量
    double GAS1 = rCode0 * rCode1 * rCode2 * rCode3 * this.out_GASh_S_nasi;
    //低負荷領域の計算仕分け:PPE
    if( this.Rh < this.in_run_stop && Rp > 0 ){
       switch( this.num_calcTypeLowerRangeLoad){
       case 0://  0_発停運転
          GAS1 = 0.0;
          break;

       case 1://  1_下限入力値固定
          //何もしない
```

```
                break;

            case 2://  2_下限COP値固定
                GAS1 *= ( this.Rh / Rp );
                break;

            case 3://  3_下限入力値と中間切片
                GAS1 *= ( 0.5 + this.Rh / Rp / 2. );
                break;

            default:
            }
        }else{
            //何もしない
        }

        //発電効率：(定格時の発電電力)÷(発電時の定格ｶﾞｽ入力-発電なし時の定格ｶﾞｽ入力)
        //double eleEffi = (this.out_ELEh_dy) / (this.out_GASh_S_ari - this.out_GASh_S_nasi);
        //↓
        //発電効率：暖房定格時の　(発電無し時の消費電力 - 発電時の消費電力 + 発電電力(系統出力))
        //        ÷ (発電時の燃料入力 - 発電無し時の燃料入力)
        double eleEffi = ( this.out_PPEh_S_nasi - this.out_PPEh_S_ari + this.out_ELEh_dy) / (this.out_GASh_S_ari -
this.out_GASh_S_nasi);

        //System.out.println(eleEffi);

        //自己発電分
        double GAS2 = 0.0;
        if( this.out_PPEh_S_ari<0.5*1000.0){//自己消費発電ありの場合
            //発電電力 Geh1(kW)　発電無しとの差分を取る
            double eleAmount = this.out_ELEh_dy - this.pacDB.getPhoerun_nasi( wbOA, this.out_PPEc_S_nasi, this.Th_Loadout
/ this.out_Qh_S);
            //発電用ガス消費量 Ggc1(kW)
            GAS2 = eleAmount/eleEffi;
            //消費電力
            this.PPE = this.out_PPEh_S_ari;
        }else{//自己消費発電なしの場合
            GAS2 = 0.0;
            //PPE = this.out_PPEh_S_nasi;
            this.PPE = this.pacDB.getPhoerun_nasi( wbOA, this.out_PPEh_S_nasi, this.Th_Loadout / this.out_Qh_S);
        }

        //系統連携分
        //発電電力 Geh2(kW)
        this.PPEout = this.pacDB.getGeh2( this.Th_Loadout / this.out_Qh_S,
this.out_ELEh_dy/1000. ,this.out_ELEh_dymax/1000.);
        this.PPEout = this.PPEout*1000.;
        //発電用ガス消費量 Ggh2(kW)
        double GAS3 = this.PPEout/eleEffi;

        this.GAS = GAS1;
        this.GASself = GAS2;
        this.GASout = GAS3;

    }else{
        //System.out.println( this.moduleName + " heating_cal:GHPタイプの選択ミス");
    }


    //冷房　処理熱量
    rCode2 = this.pacDB.getKcti( this.WBra );// 室温補正
    rCode3 = this.Lratemin;// 室内モジュールで補正(配管長、高低差補正)

    this.Sc_Loadout = rCode2 * rCode3 * this.out_Qc_S;//室外機処理可能熱量（冷却全熱）[W]

    //容量比
    this.Rc = this.allTc_heat / this.Sc_Loadout;

    this.CAPrateC = 1.0;
    this.Tc_Loadout = this.allTc_heat;//21020731
    if ( this.Rc > 1.0) {
        this.CAPrateC = 1.0 / this.Rc;
```

```
        this.Rc = 1.0;
        this.Tc_Loadout = this.Sc_Loadout;//20120731
    }

    //airOutEA
    this.airOutEA.setFlowRate( this.fRate_airEA );
    this.airOutEA.setTempDB( ( -this.Tc_Loadout + this.PPE ) / this.fRate_airEA * 1000. + this.airInOA.getTempDB() );


    if( this.Rh < 0.0001 ){
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        this.GAS = 0.0;
        this.GASself = 0.0;
        this.GASout = 0.0;
        this.PPE = 0.0;
        this.PPEout = 0.0;
        this.airOutEA.setFlowRate( 0. );
        this.airOutEA.setTempDB( this.airInOA.getTempDB() );

    }else if( this.Rh < this.in_run_stop ){
        switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
                this.CAPrateC = 0.0;
                this.CAPrateH = 0.0;
                this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
                this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
                this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
                this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
                this.GAS = 0.0;
                this.GASself = 0.0;
                this.GASout = 0.0;
                this.PPE = 0.0;
                this.PPEout = 0.0;
                this.airOutEA.setFlowRate( 0. );
                this.airOutEA.setTempDB( this.airInOA.getTempDB() );
                break;

            case 1://  1_下限入力値固定
                break;

            case 2://  2_下限COP値固定
                break;

            case 3://  3_下限入力値と中間切片
                break;

            default:
        }

    }
        //      System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
}


/**
 * 自己消費型の  ガス消費量（自己消費）
 * @return
 */
private double getEleProHeat() {

    if( this.out_Qh_S <= 0 ||  this.out_GASc_S_ari == this.out_GASc_S_nasi  ){
        return 0;
    }

    //運転時の必要電力
    double epc = pacDB.getEph( this.WBoa, this.out_PPEh_S_nasi/1000.0 , this.Th_Loadout/ this.out_Qh_S);
```

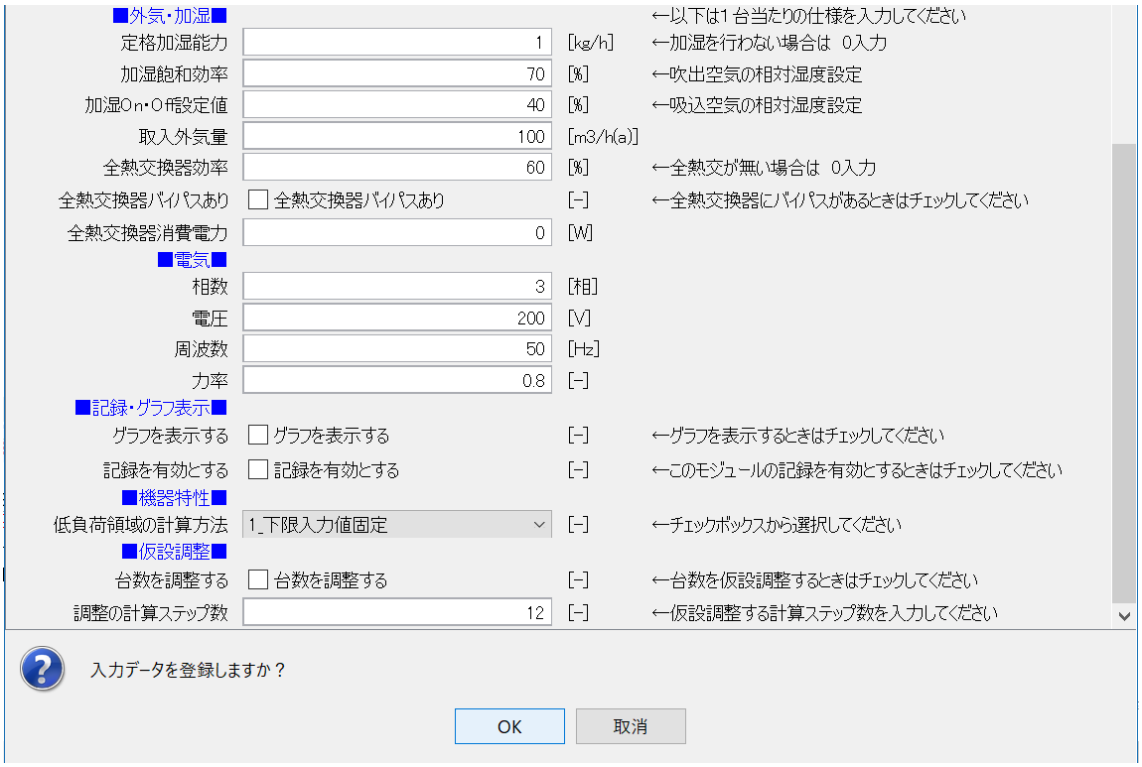
```
        epc = epc*1000.;

        //発電効率：(発電なし時の定格電力-発電時の定格電力)÷(発電時の定格ｶﾞｽ入力-発電なし時の定格ｶﾞｽ入力)
        double eleEffi = (this.out_PPEh_S_nasi - this.out_PPEh_S_ari) / (this.out_GASh_S_ari - this.out_GASh_S_nasi);


        //発電電力
        //double eleAmount = epc - this.pacDB.getPhoerun( this.WBoa, this.out_PPEh_S_ari, this.Th_Loadout /
this.out_Qh_S);//this.out_PPEh_S_ari;
        double eleAmount = epc - this.pacDB.getPhoerun_ari( this.WBoa, this.out_PPEh_S_ari );//20130729


        //System.out.println( "epc=" + epc + "  eleAmount="+ eleAmount + " eleEffi="+eleEffi );
        if( eleAmount < 0 ){
            eleAmount = 0;
        }

        //発電分のガス消費量
        if( eleEffi <= 0 || eleEffi == Double.NaN ){
            return 0;
        }

        double gas = eleAmount / eleEffi;

        return gas;
    }

    private double WB_cal(double ddb,double xx){
        double wwb,x1,de=1;
        int i,kk=-1,j=0;
        wwb=ddb;

        do{
            j++;
            x1=Psychrometrics.FNXtw(ddb,wwb);
            if(Math.abs(x1-xx)<0.000003) {
//              System.out.println(" j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
                return wwb;
            }
            if(x1<xx) {
                if(kk==-1) de=de/2.0;
                wwb=wwb+de;
                kk=1;
            }
                if(x1>=xx) {
                if(kk==1) de=de/2.0;
                wwb=wwb-de;
                kk=-1;
            }

        }while(j<1000);
//      System.out.println("*j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
        return wwb;

    }
}
```

「PAC EHP ウォールスルー2015」（場所：設備 2015／個別分散 2015／）

モジュール名	PAC EHP ウォールスルー2015
クラス	WallThroughModule2015（本体） WallThroughSpec2015（仕様） WallThroughCalc2015（計算）

（1）入力画面

・スペック



（2）モジュールの概要

ウォールスルータイプの PAC モジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

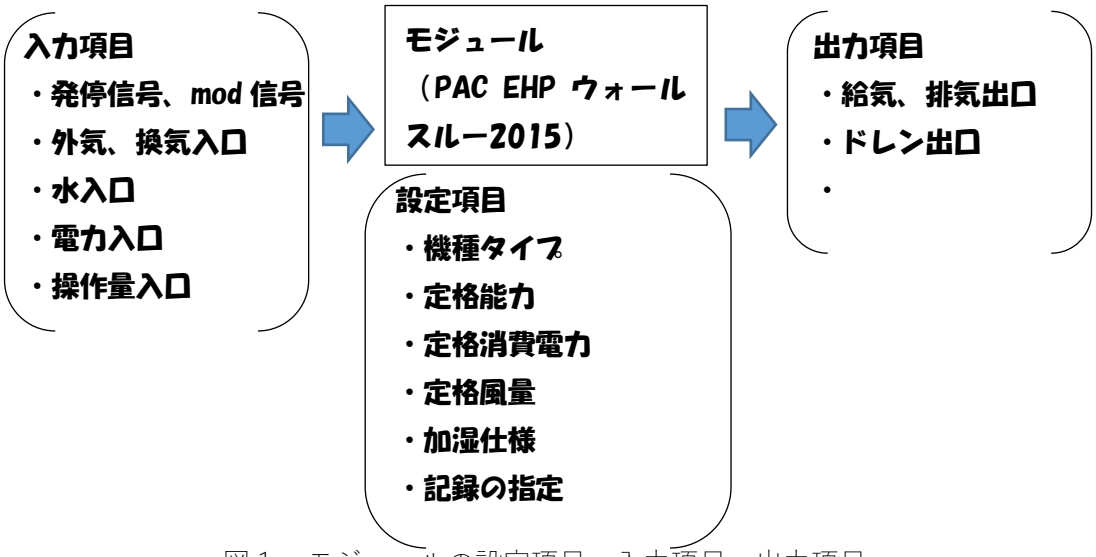


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	－	[-]	－	－		
1	室グループ／室／ゾーン	String		#RoomGroup	[-]	－	－		←室グループ／室／ゾーンを選択してください。
2	台数	double	in_dai	1	[台]	－	0		
3	機器番号	String	m_no		[-]	－	－		
4	機器種別	String	m_ty	0_201303_標準型、1_201303_インバータ型、0_標準型、1_インバータ型	[-]	－	－		←チェックボックスから選択してください
5	機器型式	String	m_kt		[-]	－	－		
6	■定格能力など■					－	－		←以下は１台当たりの仕様を入力してください
7	定格冷房能力	double	out_Qc_S	2.2	[kW]	－	0		
8	中間冷房能力	double	out_Qc_C	1.1	[kW]	－	0		←任意入力項目です
9	定格暖房能力	double	out_Qh_S	2.5	[kW]	－	0		
10	中間暖房能力	double	out_Qh_C	1.3	[kW]	－	0		←任意入力項目です
11	低温暖房能力	double	out_Qh_L	2	[kW]	－	0		
12	■定格入力など■					－	－		←以下は１台当たりの仕様を入力してください
13	定格冷房入力(電力)	double	out_PPEc_S	0.73	[kW]	－	0		
14	中間冷房入力(電力)	double	out_PPEc_C	0.33	[kW]	－	0		←任意入力項目です

15	定格暖房入力(電力)	double	out_PPEh_S	0.72	[kW]	―	0		
16	中間暖房入力(電力)	double	out_PPEh_C	0.35	[kW]	―	0		←任意入力項目です
17	低温暖房入力(電力)	double	out_PPEh_L	0.62	[kW]	―	0		
18	機器起動停止負荷率	double	in_run_stop	10	[%]	100	0		←部分負荷率を入力してください
19	定格風量	double	fRate_airEA	480	[m3/h(a)]	―	0		
20	定格ファン消費電力	double	in_Va_S	0.2	[kW]	―	0		
21	■外気・加湿■					―	―		←以下は１台当たりの仕様を入力してください
22	定格加湿能力	double	in_HUM_mx	1	[kg/h]	―	0		←加湿を行わない場合は　0 入力
23	加湿飽和効率	double	in_HUM_rt	70	[%]	―	0		←吹出空気の相対湿度設定
24	加湿 On・Off 設定値	double	in_HUM_on	40	[%]	―	0		←吸込空気の相対湿度設定
25	取入外気量	double	in_OA_S	100	[m3/h(a)]	―	0		
26	全熱交換器効率	double	in_EX_S	60	[%]	100	0		←全熱交が無い場合は　0 入力
27	全熱交換器バイパスあり	boolean	isHexbypass	FALSE	[-]	―	―		←全熱交換器にバイパスがあるときはチェックしてください
28	全熱交換器消費電力	double	in_EX_PE	0	[W]	―	0		
29	■電気■					―	―		
30	相数	double	phase	3	[相]	―	1		
31	電圧	double	voltage	200	[V]	―	100		
32	周波数	double	frequency	50	[Hz]	60	50		
33	力率	double	powerFactor	0.8	[-]	1	0		
34	■記録・グラフ表示■					―	―		
35	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてくだ

									さい
36	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください
37	■機器特性■					ー	ー		
38	低負荷領域の計算方法	String	calcTypeLowerRangeLoad	0_発停運転、1_下限入力値固定、2_下限COP値固定、3_下限入力値と中間切片	[-]	ー	ー		←チェックボックスから選択してください
39	■仮設調整■					ー	ー		
40	台数を調整する	boolean	isAdjust2012	FALSE	[-]	ー	ー		←台数を仮設調整するときはチェックしてください
41	調整の計算ステップ数	int	numAdjustSteps	12	[-]	ー	1		←仮設調整する計算ステップ数を入力してください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	外気発停信号入口	L1_swcInOA	swcInOA	-	制御	OnOff 信号	入口	
4	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
5	PID モード入口	L1_modInPID	modInPID	-	制御	制御モード	入口	
6	外気入口	L0_airInOA	airInOA	-	状態	空気	入口	
7	排気出口	L0_airOutEA	airOutEA	-	状態	空気	出口	
8	還気入口	L0_airInRA	airInRA	-	状態	空気	入口	
9	給気出口	L0_airOutSA	airOutSA	-	状態	空気	出口	
10	給水入口	L0_watInCW	watInCW	-	状態	水	入口	
11	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	
12	ドレン出口	L0_watOutD	watOutD	-	状態	水	出口	
13	操作量入口	L0_valInCtrl	valInCtrl	-	値	値	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	室内機_Message#-#-	メッセージ	—	メッセージ
2	室内機_冷却処理顕熱量合計#W#熱量	室内機_冷却処理顕熱量合計	W	My
3	室内機_加熱処理顕熱量合計#W#熱量	室内機_加熱処理顕熱量合計	W	My
4	室内機_冷却処理全熱量合計#W#熱量	室内機_冷却処理全熱量合計	W	My
5	室内機_加熱処理全熱量合計#W#熱量	室内機_加熱処理全熱量合計	W	My
6	室外機_消費電力#W#消費電力	室外機_消費電力	W	エネルギー消費
7	室外機_冷却可能熱量/最大#W#熱量	室外機_冷却可能熱量/最大	W	My
8	室外機_加熱可能熱量/最大#W#熱量	室外機_加熱可能熱量/最大	W	My
9	室外機_処理全熱量合計負荷#W#-	室外機_処理全熱量合計負荷	W	負荷
10	室内機_PID 操作量#-#-	室内機_PID 操作量	-	My
11	室内機_吹出乾球温度#℃#温度	室内機_吹出乾球温度	℃	出口
12	室内機_吹出絶対湿度#g/g#湿度	室内機_吹出絶対湿度	g/g	出口
13	室内機_吹出風量#g/s#質量流量	室内機_吹出風量#g/s	g/s	出口
14	室内機_ドレン量#g/s#質量流量	室内機_ドレン量#g/s	g/s	出口
15	室内機_加湿給水量#g/s#質量流量	室内機_加湿給水量	g/s	入口
16	室内機_還気乾球温度#℃#温度	室内機_還気乾球温度	℃	入口
17	室内機_還気絶対湿度#g/g#湿度	室内機_還気絶対湿度	g/g	入口
18	室内機_還気湿球温度#℃#温度	室内機_還気湿球温度	℃	入口
19	室内機_入口乾球温度#℃#温度	室内機_入口乾球温度	℃	入口
20	室内機_入口絶対湿度#g/g#湿度	室内機_入口絶対湿度	g/g	入口
21	室内機_入口湿球温度#℃#温度	室内機_入口湿球温度	℃	入口
22	室外機_外気乾球温度#℃#温度	室外機_外気乾球温度	℃	入口
23	室外機_外気絶対湿度#g/g#湿度	室外機_外気絶対湿度	g/g	入口
24	室外機_外気湿球温度#℃#温度	室外機_外気湿球温度	℃	入口
25	室外機_調整冷却能力#W#熱量	室外機_調整冷却能力	W	調整
26	室外機_調整加熱能力#W#熱量	室外機_調整加熱能力	W	調整
27	室内機_調整冷却台数#-#台数	室内機_調整冷却台数	-	調整
28	室内機_調整加熱台数#-#台数	室内機_調整加熱台数	-	調整
29	室内機_調整ステップ平均冷却台数#-#台数	室内機_調整ステップ平均冷却台数	-	調整
30	室内機_調整ステップ平均加熱台数#-#台数	室内機_調整ステップ平均加熱台数	-	調整

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

■WallThroughModule2015

```
package jp.or.ibec.best.domain.sample.air;

import java.util.Map;

import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;


/**
 * @author HIROSHI NINOMIYA/20101212
 *
 * ウォールスルーEHP
 *
 * 201303機器特性追加
 *
 */
public class WallThroughModule2015 extends AbstractBestModule implements
IBestMetaModule  {

   /**
    * 論理アドレス
    */
   //private final String moduleName="(WallThroughModule2015)";


   private IBestStateMessage stateNodes = null;
   private IBestControlMessage commandNodes = null;
   private IBestRecordMessage recordNodes = null;

   private WallThroughSpec2015 wtSpec = null;


   @Override
   public void setProfile(BestSpecs spec) {
//    外部定義項目取得
      if(spec == null) {
         return;
      }
      Map<String,String> map=spec.getSpec();
      if(map == null) {
         return;
      }

      this.wtSpec = new WallThroughSpec2015();

      WallThroughCalc2015.setProfile( spec, this.wtSpec );

   }

   @Override
   public void initialize(IBestStateMessage stateNodes,
         IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
         IBestDomainSchedule schedule) {
      //System.out.println(this.moduleName + " BuillMultiOutイニシャライズ");

      this.stateNodes = stateNodes;
      this.commandNodes = commandNodes;
      this.recordNodes = recordNodes;

      //状態ノードを受け取る
      super.sm = stateNodes;
```

```
        //制御ノードを受け取る
        super.cm = commandNodes;
        //記録ノードを受け取る
        super.rm = recordNodes;

        WallThroughCalc2015.initialize( super.transferMapState, this, this.stateNodes, this.commandNodes, this.recordNodes,
this.wtSpec );

    }


    /**
     * ウォールスルーEHP計算及び結果出力/各室内機よりも先に動きます。
     */
    @Override
    public void outputs() {

        if (null == super.cm || null == super.sm ){
            return;
        }

        WallThroughCalc2015.outputs_calc( this, this.stateNodes, this.commandNodes, this.recordNodes, this.wtSpec );

    }


    /**
     * ウォールスルーEHP計算/各室内機の集計後
     */

    @Override
    public void update() {

        WallThroughCalc2015.update( this.wtSpec );

    }

}
```

■WallThroughSpec2015

```
package jp.or.ibec.best.domain.sample.air;

import java.util.LinkedList;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.DO.BestWater;


/**
 * @author HIROSHI NINOMIYA/20101212
 *
 * ウォールスルーEHP
 *
 * 201303機器特性追加
 *
 */
public class WallThroughSpec2015 {

    /**
     * 論理アドレス
     */
    protected final String moduleName="(WallThroughSpec2015)";

    /**
     * 外部定義
     */
    protected StringBuffer message= new StringBuffer();

    /**
     * 外部定義
     */
    /**
     * 変数（外部定義に対応）
     */

    protected String name          ;// "機器名称";
    protected String m_no          ;//= "機器番号";
    protected String m_ty          ;//= "機器種別";
    protected String m_kt          ;//= "機器型式";
    protected double out_Qc_S      ;//= "定格冷房能力";//kW
    protected double out_Qc_C      ;//= "中間冷房能力";//kW
    protected double out_PPEc_S    ;//= "定格冷房入力(電力)";//kW
    protected double out_PPEc_C    ;//= "中間冷房入力(電力)";//kW
    protected double out_Qh_S      ;//= "定格暖房能力";//kW
    protected double out_Qh_C      ;//= "中間暖房能力";//kW
    protected double out_Qh_L      ;//= "低温暖房能力";//kW
    protected double out_PPEh_S    ;//= "定格暖房入力(電力)";//kW
    protected double out_PPEh_C    ;//= "中間暖房入力(電力)";//kW
    protected double out_PPEh_L    ;//= "低温暖房入力(電力)";//kW
/*
    private double out_CLEa_R=0.0    ;//= "クランクケースヒータ(運転時)";//W
    private double out_CLEa_S=0.0    ;//= "クランクケースヒータ(停止時)";//W
*/
    protected double out_TKEa_R =0.0  ;//= "定格待機電力(運転時)";//W
    protected double out_TKEa_T =0.0  ;//= "定格待機電力(待機時)";//W
    protected double out_TKEa_S =0.0  ;//= "定格待機電力(停止時)";//W

    protected double fRate_airEA ;//風量[g/s]

    protected double in_run_stop    ;//= "機器起動停止負荷率";//[%]
    protected int phase;       //相数[-]
    protected double voltage;     //電圧[V]
    protected double frequency;   //周波数[Hz]
    protected double powerFactor; //力率[-]

    protected boolean isGVisible = false;//このグラフを表示する=true
```

```
   protected boolean isRecord   = false;//記録を有効とする=true


   /**
    * 変数（外部定義に対応）
    */

// private String name           ;//= "機器名称";
// private String m_no           ;//= "機器番号";
// private String m_kt           ;//= "機器型式";
   protected double in_dai       ;//= "台数";
// private double in_Qc_S        ;//= "定格冷房能力";//kW
// private double in_Qh_S        ;//= "定格暖房能力";//kW
   protected double in_Va_S         ;//= "定格風量";//m3/h
   protected double in_PPEa_S    ;//= "定格ファン消費電力";//W
   protected double in_TKEa_R=0.0   ;//= "定格待機電力(運転時)";//W
   protected double in_TKEa_S=0.0   ;//= "定格待機電力(停止時)";//W
// private double in_run_stop       ;//= "機器起動停止負荷率";//[%]
// private double in_Lp          ;//= "冷媒管長";//m
// private double in_Lh          ;//= "冷媒管高低差";//m
   protected double in_HUM_mx       ;//= "定格加湿能力";//kg/h
   protected double in_HUM_rt       ;//= "加湿飽和効率";//[%]
   protected double in_HUM_on       ;//= "加湿On・Off設定値";//[%]
   protected double in_OA_S         ;//= "取入外気量";//m3/h
   protected double in_EX_S         ;//= "全熱交換器効率";//[%]
   protected boolean isHexbypass = false;//熱交バイパスあり=true
   protected double in_EX_PE     ;//= "全熱交換器消費電力[W]";

// private int phase        ;//= "相数";
// private double voltage      ;//= "電圧";
// private double frequency   ;// = "周波数";
// private double powerFactor  ;//= "力率";

// private boolean isGVisible = false;//このグラフを表示する=true
// private boolean isRecord   = false;//記録を有効とする=true

// private StringBuffer message= new StringBuffer();

   protected BestAir airInOA  = null;
   protected BestAir airOutEA = null;

   protected BestElectricity eleIn = null;   //電力
// protected Double CAPrateC = null;//室内機能力補正比率[-]冷房側
// protected Double CAPrateH = null;//室内機能力補正比率[-]暖房側
// private Double CAPrate = null;
//  private BM_EHPdata RMdata = null;  //電力
   protected PACDBManager pacDB=null;

   protected int modIn  ;
   protected int modInPID;
   protected int swcIn ;
   protected int swcInOA ;
   protected int mState;   //停止、冷房、暖房フラグ



   protected BestAir airInRA   = null;
   protected BestAir airOutSA  = null;       //給気
   protected BestWater watOutD = null;
   protected BestWater watInCW = null;
// private BestElectricity eleIn = null;  //電力
// private BM_EHPdata RMdata = new BM_EHPdata();   //電力
   protected BestValue PIDrate = null;
// private Double CAPrate = null;
// private Double LPrate = null;

// private PACDBManager pacDB=null;

// private int mode  ;
// private int onOff ;
// private int mState;  //停止、冷房、暖房フラグ
```

```
    /**
     * 出力（建物）
     */

    /**
     * 出力（建物）
     */

    protected double DBra;          //室内乾球温度[℃]
    protected double WBra;          //室内湿球温度[℃]
    protected double XGra;
    protected double IAra;
    protected double RHra;
    protected double CO2ppmra;

    protected double DBoa;          //外気乾球温度[℃]
    protected double WBoa;          //外気湿球温度[℃]
    protected double XGoa;
    protected double IAoa;
    protected double CO2ppmoa;

    protected double DBin;          //室内機吸込乾球温度[℃]
    protected double WBin;          //室内機吸込湿球温度[℃]
    protected double XGin;
    protected double IAin;
    protected double CO2ppmin;

    protected double DBout;         //乾球温度[℃]
    protected double XGout;         //絶対湿度[kg/kg.DA]
    protected double CO2ppmout;

    protected double GW;            //風量(g/s)
    protected double D_Wrate;       //ドレン水量(g/s)
    protected double CW_Wrate;      //加湿水量(g/s)

    protected double PPE_out;
    protected double E_PPE_out;
    protected double PPE_in;
    protected double E_PPE_in;
// protected double S_Load;
// protected double T_Load;
// protected double Sc_Load;
// protected double Tc_Load;
// protected double Sh_Load;
// protected double Th_Load;
// private double S_Load_RM;//室内機処理熱量（顕熱）[W]
// private double T_Load_RM;//室内機処理熱量（全熱）[W]
// protected double Sc_Load_RM;//室内機処理熱量（顕熱）[W]
// protected double Tc_Load_RM;//室内機処理熱量（全熱）[W]
// protected double Sh_Load_RM;//室内機処理熱量（顕熱）[W]
// protected double Th_Load_RM;//室内機処理熱量（全熱）[W]
// private double S_Load;
// private double T_Load;
// private double S_Load_RM;
// private double T_Load_RM;
//******************090725-s
    protected double saijohatsu;
    protected double T_C_heat;
// private int u40=0;

//******************090725-e
    protected int dbflg=0;
    protected String path="EHP";
    protected String[] filenames= new String[1];
    protected String equipmentName;

    //機器特性式の適用上下限値
    protected double maxDBOA_C_FACK;// = wtspec.pacDB.getformulaRangeMax( wtspec.pacDB.getFACKcta() );//特性/冷房/外気乾球
温度/上限値
    protected double minDBOA_C_FACK;// = wtspec.pacDB.getformulaRangeMin( wtspec.pacDB.getFACKcta() );//特性/冷房/外気乾球
温度/下限値
    protected double maxWBRA_C_FACK;// = wtspec.pacDB.getformulaRangeMax( wtspec.pacDB.getFACKcti() );//特性/冷房/室内湿球
```

```
温度/上限値
   protected double minWBRA_C_FACK;// = wtspec.pacDB.getformulaRangeMin( wtspec.pacDB.getFACKcti() );//特性/冷房/室内湿球
温度/下限値

   protected double maxWBOA_H_FACK;// = wtspec.pacDB.getformulaRangeMax( wtspec.pacDB.getFACKhta() );//特性/暖房/外気湿球
温度/上限値
   protected double minWBOA_H_FACK;// = wtspec.pacDB.getformulaRangeMin( wtspec.pacDB.getFACKhta() );//特性/暖房/外気湿球
温度/下限値
   protected double maxDBRA_H_FACK;// = wtspec.pacDB.getformulaRangeMax( wtspec.pacDB.getFACKhti() );//特性/暖房/室内乾球
温度/上限値
   protected double minDBRA_H_FACK;// = wtspec.pacDB.getformulaRangeMin( wtspec.pacDB.getFACKhti() );//特性/暖房/室内乾球
温度/下限値

   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   protected int HEXselect=1;//1=エンタルピ基準  2=温度基準
   //++++++++++++++++++++++++++++++++++++++++++++++++++++++++

   //グラフ表示など
   protected GraphJFrameBuilMultiIn_S20101212 gBMIn = null;

   //低負荷領域の計算方法
   protected String calcTypeLowerRangeLoad = null;//"低負荷領域の計算方法";
   protected int num_calcTypeLowerRangeLoad;//"低負荷領域の計算方法";

   /**
    * その他変数
    */

   protected double Rc;             //部分負荷率
   protected double Rh;             //部分負荷率

// private double DBoa;          //外気乾球温度[℃]
// private double WBoa;          //外気湿球温度[℃]
// private double XGoa;          //外気絶対湿度[g/gD.A.]

// private double DBra;          //室内機吸込乾球温度[℃]
// private double WBra;          //室内機吸込湿球温度[℃]
// private double XGra;          //室内機吸込絶対湿度[g/gD.A.]

   protected double Sc_Loadout;//室外機処理可能熱量（冷却全熱）[W]
   protected double Sh_Loadout;//室外機処理可能熱量（加熱全熱）[W]
   protected double Tc_Loadout;//室外機実処理熱量（冷却全熱）[W]
   protected double Th_Loadout;//室外機実処理熱量（加熱全熱）[W]
   protected double allSc_heat;//室内機処理熱量（冷却顕熱）[W]
   protected double allSh_heat;//室内機処理熱量（加熱顕熱）[W]
   protected double allTc_heat;//室内機処理熱量（冷却全熱）[W]
   protected double allTh_heat;//室内機処理熱量（加熱全熱）[W]
// private double S_Loadout;//室外機処理可能熱量[W]
// private double T_Loadout;//室外機実処理熱量[W]
// private double allSheat;//室内機処理熱量（顕熱）[W]
// private double allTheat;//室内機処理熱量（全熱）[W]
// private double PPE;
// private double E_PPE;

// private int num;
   protected String kiki_file;
// private String path="EHP";
// private String[] filenames= new String[1];
// private String equipmentName;

// private double DBra_max;
// private double WBra_min;
// private double Lratemin;
   protected double fmc;
   protected double pmc;
   protected double fmh;
   protected double pmh;

   protected int iType = 0;
   protected final int iType_STANDARD = 0;
   protected final int iType_INVERTER = 1;
```

```
    //グラフ表示など
    protected GraphJFrameBuilMultiOut_S20101212 gBMout = null;

    //仮設調整モード2012

    protected boolean isAdjust2012  = false;//true="台数を調整する";//
    //private boolean isAdjust2012 = false;//20120406nino 建築学会大会空衛学会OS論文検討用＝true
    protected int numAdjustSteps;//"調整の計算ステップ数";
    protected LinkedList<Double> daiListc = null;//必要台数
    protected LinkedList<Double> daiListh = null;//必要台数
    protected double maxcAdjustdai = 1;
    protected double maxhAdjustdai = 1;
    protected double avecdaiAdjust = 0;
    protected double avehdaiAdjust = 0;
    protected double in_daicAdjust = 1;//= "調整台数";
    protected double in_daihAdjust = 1;//= "調整台数";
    protected double in_run_stop_Num;//= "機器起動停止負荷率"台数補正;

    protected double out_Qc_S_1      ;//= "定格冷房能力";//kW
    protected double out_Qc_C_1      ;//= "中間冷房能力";//kW
    protected double out_PPEc_S_1    ;//= "定格冷房入力(電力)";//kW
    protected double out_PPEc_C_1    ;//= "中間冷房入力(電力)";//kW
    protected double out_Qh_S_1      ;//= "定格暖房能力";//kW
    protected double out_Qh_C_1      ;//= "中間暖房能力";//kW
    protected double out_Qh_L_1      ;//= "低温暖房能力";//kW
    protected double out_PPEh_S_1    ;//= "定格暖房入力(電力)";//kW
    protected double out_PPEh_C_1    ;//= "中間暖房入力(電力)";//kW
    protected double out_PPEh_L_1    ;//= "低温暖房入力(電力)";//kW

// private double fRate_airEAc ;//風量[g/s]
// private double fRate_airEAh ;//風量[g/s]

    protected double in_Va_S_1    ;//= "定格風量";//m3/h
    protected double in_PPEa_S_1     ;//= "定格ファン消費電力";//W

    protected double in_HUM_mx_1     ;//= "定格加湿能力";//kg/h

    protected ZoneAirforSSinsideModule201502 zoneAirforSSinside = null;
    protected String grzName = null;
    protected boolean isUseZoneAir = false;


/* static double dbOA;
    static double wbIn;
    static double maxCapacity;
    static double opeCapacity;

    static double rCode0;
    static double rCode1;
    static double rCode2;
    static double rCode3;
    static double IAout1;

    static double rc;//負荷率
    static double s_Load;//室内機処理負荷顕熱
    static double t_Load;//室内機処理負荷全熱

    static double alph;
    static double beta;

    static double ppe_out;


    static double wbOA;
    static double dbIn;

    static double rh;
*/
}
```

■WallThroughCalc2015

```
package jp.or.ibec.best.domain.sample.air;

import java.util.LinkedList;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.essentials.utility.Psychrometrics;
import jp.or.ibec.best.message.BestEngineMessageHandler;
import jp.or.ibec.best.message.BestEngineMessageParam;
import jp.or.ibec.best.message.constants.SystemMessageConstants;

/**
 * @author HIROSHI NINOMIYA/20101212
 *
 * ウォールスルーEHP
 *
 * 201303機器特性追加
 *
 */
public class WallThroughCalc2015  {

   /**
    * 論理アドレス
    */
// private final String moduleName="(WallThroughCalc2015)";

   private static final String S_NODE_airInOA  = "L0_airInOA";//外気
   private static final String S_NODE_airOutEA = "L0_airOutEA";//外気

   private static final String S_NODE_airInRA  = "L0_airInRA";//室内吸込口
   private static final String S_NODE_airOutSA = "L0_airOutSA";//室内吹出し

   private static final String S_NODE_watOutD  = "L0_watOutD";//コイル出口ドレン
   private static final String S_NODE_watInCW  = "L0_watInCW";//watInCW---090407追加**************

   private static final String S_NODE_valInPID = "L0_valInCtrl";//供給熱量 from PID          : in

   private static final String S_NODE_eleIn    = "L0_eleIn";//電力

   private static final String C_NODE_swcIn    = "L1_swcIn";//運転状態：off/on
   private static final String C_NODE_swcInOA  = "L1_swcInOA";//OA状態：off/on
   private static final String C_NODE_modIn    = "L1_modIn";//空調モード：停止/冷却/加熱
   private static final String C_NODE_modInPID = "L1_modInPID";//PID空調モード：停止/冷却/加熱

   private static final String R_NODE = "L2_recOut";



   /**
    * 外部定義
    */
   private static final String SPEC_name         = "名称";

   private static final String SPEC_grzName  = "室グループ／室／ゾーン";

   private static final String SPEC_m_no         = "機器番号[-]";
   private static final String SPEC_m_ty         = "機器種別[-]";
   private static final String SPEC_m_kt         = "機器型式[-]";
   private static final String SPEC_out_Qc_S     = "定格冷房能力[W]";//kW
```

```
    private static final String SPEC_out_Qc_C      = "中間冷房能力[W]";//kW
    private static final String SPEC_out_Qh_S      = "定格暖房能力[W]";//kW
    private static final String SPEC_out_Qh_C      = "中間暖房能力[W]";//kW
    private static final String SPEC_out_Qh_L      = "低温暖房能力[W]";//kW
    private static final String SPEC_out_PPEc_S    = "定格冷房入力(電力)[W]";//kW
    private static final String SPEC_out_PPEc_C    = "中間冷房入力(電力)[W]";//kW
    private static final String SPEC_out_PPEh_S    = "定格暖房入力(電力)[W]";//kW
    private static final String SPEC_out_PPEh_C    = "中間暖房入力(電力)[W]";//kW
    private static final String SPEC_out_PPEh_L    = "低温暖房入力(電力)[W]";//kW
/*
    private static final String SPEC_out_CLEa_R    = "クランクケースヒータ(運転時)";//W
    private static final String SPEC_out_CLEa_S    = "クランクケースヒータ(停止時)";//W
    private static final String SPEC_out_TKEa_R    = "定格待機電力(運転時)";//W
    private static final String SPEC_out_TKEa_T    = "定格待機電力(待機時)";//W
    private static final String SPEC_out_TKEa_S    = "定格待機電力(停止時)";//W
*/
// private static final String SPEC_fRate_airEA   = "風量[g/s]";

    private static final String SPEC_in_run_stop    = "機器起動停止負荷率[-]";//[%]

    private static final String SPEC_Phase       = "相数[-]";
    private static final String SPEC_Voltage     = "電圧[V]";
    private static final String SPEC_Frequency   = "周波数[Hz]";
    private static final String SPEC_PowerFactor = "力率[-]";
    //
    private static final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
    private static final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

    private static final String SPEC_CalcTypeLowerRangeLoad = "低負荷領域の計算方法";

    private static final String SPEC_isAdjust2012   = "台数を調整する";
    private static final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";


    /**
     * 外部定義
     */

    private static final String SPEC_in_dai      = "台数[-]";
// private static final String SPEC_in_Qc_S        = "定格冷房能力";//kW
// private static final String SPEC_in_Qh_S     = "定格暖房能力";//kW
    private static final String SPEC_in_Va_S     = "定格風量[g/s]";//m3/h
    private static final String SPEC_in_PPEa_S   = "定格ファン消費電力[W]";//W
// private static final String SPEC_in_TKEa_R       = "定格待機電力(運転時)";//W
// private static final String SPEC_in_THEa_S       = "定格待機電力(停止時)";//W
// private static final String SPEC_in_run_stop   = "機器起動停止負荷率";//[%]
// private static final String SPEC_in_Lp          = "冷媒管長";//m
// private static final String SPEC_in_Lh          = "冷媒管高低差";//m
    private static final String SPEC_in_HUM_mx      = "定格加湿能力[g/s]";//kg/h
    private static final String SPEC_in_HUM_rt      = "加湿飽和効率[-]";//[%]
    private static final String SPEC_in_HUM_on      = "加湿On・Off設定値[-]";//[%]
    private static final String SPEC_in_OA_S     = "取入外気量[g/s]";//m3/h
    private static final String SPEC_in_EX_S     = "全熱交換器効率[-]";//[%]
    private static final String SPEC_isHexbypass    = "全熱交換器バイパスあり[-]";
    private static final String SPEC_in_EX_PE    = "全熱交換器消費電力[W]";


    /**
     * 変数（外部定義に対応）
     */


    /**
     * 変数（外部定義に対応）
     */


    /**
     * 出力（建物）
     */
    private static final String RECORD_message = "室内機_Message#-#-";
// private static final String RECORD_ID1  = "室内機_処理顕熱量合計#W#熱量";
```

```
   private static final String RECORD_ID1c = "室内機_冷却処理顕熱量合計#W#熱量";
   private static final String RECORD_ID1h = "室内機_加熱処理顕熱量合計#W#熱量";
// private static final String RECORD_ID2  = "室内機_処理全熱量合計#W#熱量";
   private static final String RECORD_ID2c = "室内機_冷却処理全熱量合計#W#熱量";
   private static final String RECORD_ID2h = "室内機_加熱処理全熱量合計#W#熱量";
// private static final String RECORD_ID3c = "室内機_冷却能力補正比率";
// private static final String RECORD_ID3h = "室内機_加熱能力補正比率";
   private static final String RECORD_ID4  = "室外機_消費電力#W#消費電力";
   private static final String RECORD_ID5c = "室外機_冷却可能熱量/最大#W#熱量";
   private static final String RECORD_ID5h = "室外機_加熱可能熱量/最大#W#熱量";
// private static final String RECORD_ID6c = "室外機冷却実処理熱量/現状#W#熱量";
// private static final String RECORD_ID6h = "室外機加熱実処理熱量/現状#W#熱量";
   private static final String RECORD_qT   = "室外機_処理全熱量合計負荷#W#-";
   private static final String RECORD_pidVal  = "室内機_PID操作量#-#-";

// private static final String RECORD_ID1 = "室内機処理顕熱量合計#W#熱量";
// private static final String RECORD_ID2 = "室内機処理全熱量合計#W#熱量";
// private static final String RECORD_ID3 = "室内機能力補正比率";
// private static final String RECORD_ID4 = "室外機消費電力#W#消費電力";
// private static final String RECORD_ID5 = "室外機処理可能熱量/最大#W#熱量";
// private static final String RECORD_ID6 = "室外機実処理熱量/現状#W#熱量";

   /**
    * 出力（建物）
    */

// private static final String RECORD_ID1 = "室内機処理顕熱量#W#熱量";
// private static final String RECORD_ID2 = "室内機処理全熱量#W#熱量";
// private static final String RECORD_ID3 = "室内機消費電力#W#消費電力";
   private static final String RECORD_ID4i = "室内機_吹出乾球温度#℃#温度";
   private static final String RECORD_ID5i = "室内機_吹出絶対湿度#g/g#湿度";
   private static final String RECORD_ID6i = "室内機_吹出風量#g/s#質量流量";
   private static final String RECORD_ID7i = "室内機_ドレン量#g/s#質量流量";
   private static final String RECORD_ID8i = "室内機_加湿給水量#g/s#質量流量";

   private static final String RECORD_DBra = "室内機_還気乾球温度#℃#温度";
   private static final String RECORD_Xra  = "室内機_還気絶対湿度#g/g#湿度";
   private static final String RECORD_WBra = "室内機_還気湿球温度#℃#温度";

   private static final String RECORD_DBin = "室内機_入口乾球温度#℃#温度";
   private static final String RECORD_Xin  = "室内機_入口絶対湿度#g/g#湿度";
   private static final String RECORD_WBin = "室内機_入口湿球温度#℃#温度";

   private static final String RECORD_DBoa = "室外機_外気乾球温度#℃#温度";
   private static final String RECORD_Xoa  = "室外機_外気絶対湿度#g/g#湿度";
   private static final String RECORD_WBoa = "室外機_外気湿球温度#℃#温度";

   private static final String RECORD_out_Qc_Adjust = "室外機_調整冷却能力#W#熱量";
   private static final String RECORD_out_Qh_Adjust = "室外機_調整加熱能力#W#熱量";
   private static final String RECORD_in_daicAdjust = "室内機_調整冷却台数#-#台数";
   private static final String RECORD_in_daihAdjust = "室内機_調整加熱台数#-#台数";
   private static final String RECORD_avecdaiAdjust = "室内機_調整ステップ平均冷却台数#-#台数";
   private static final String RECORD_avehdaiAdjust = "室内機_調整ステップ平均加熱台数#-#台数";


   //@Override
   public static void setProfile( BestSpecs spec, WallThroughSpec2015 wtspec ) {
//    外部定義項目取得
      if(spec == null) {
         return;
      }
      Map<String,String> map=spec.getSpec();
      if(map == null) {
         return;
      }

      // 機器名称
      wtspec.name = spec.getSpecValue( SPEC_name, wtspec.moduleName );

      wtspec.grzName = spec.getSpecValue( SPEC_grzName, "" );
      if( !wtspec.grzName.equals( "" )){
         wtspec.isUseZoneAir = true;
```

```
}

if( wtspec.isUseZoneAir ){
   wtspec.zoneAirforSSinside = new ZoneAirforSSinsideModule201502();
   wtspec.zoneAirforSSinside._setProfile( wtspec.name, wtspec.grzName );
}

// 機器番号
wtspec.m_no = spec.getSpecValue( SPEC_m_no, "" );

//  機器種別  0_標準型、1_インバータ
wtspec.m_ty = spec.getSpecValue( SPEC_m_ty, "0_201303_標準型" );

//
if( wtspec.m_ty.equals( "0_201303_標準型")){//0_201303_標準型
   wtspec.iType = wtspec.iType_STANDARD;
   wtspec.kiki_file ="WallThroughEHP201303";
   wtspec.out_TKEa_R = 0;//待機電力（運転中）
   wtspec.out_TKEa_S = 5;//待機電力（停止時）
   wtspec.out_TKEa_T = 5;//待機電力（待機時）
}else if( wtspec.m_ty.equals( "1_201303_インバータ型")){
   wtspec.iType = wtspec.iType_INVERTER;
   wtspec.kiki_file ="WallThroughInvEHP201303";
   wtspec.out_TKEa_R = 0;//待機電力（運転中）
   wtspec.out_TKEa_S = 11;//待機電力（停止時）
   wtspec.out_TKEa_T = 11;//待機電力（待機時）
}
//
else if( wtspec.m_ty.equals( "0_標準型")){
   wtspec.iType = wtspec.iType_STANDARD;
   wtspec.kiki_file ="WallThroughEHP20100625";
}else if( wtspec.m_ty.equals( "1_インバータ型")){
   wtspec.iType = wtspec.iType_INVERTER;
   wtspec.kiki_file ="WallThroughInvEHP20100625";
}else{
   wtspec.iType = wtspec.iType_STANDARD;
   wtspec.kiki_file ="WallThroughEHP201303";
}

// 機器型式
wtspec.m_kt = spec.getSpecValue( SPEC_m_kt, "" );

//    台数
wtspec.in_dai = spec.getSpecValue( SPEC_in_dai, 1 );

// 定格冷房能力
wtspec.out_Qc_S   = spec.getSpecValue( SPEC_out_Qc_S, 0. ) * wtspec.in_dai;
wtspec.out_Qc_S_1 = spec.getSpecValue( SPEC_out_Qc_S, 0. );

// 中間冷房能力
wtspec.out_Qc_C   = spec.getSpecValue( SPEC_out_Qc_C, 0. ) * wtspec.in_dai;
wtspec.out_Qc_C_1 = spec.getSpecValue( SPEC_out_Qc_C, 0. );

// 定格冷房入力(電力)
wtspec.out_PPEc_S   = spec.getSpecValue( SPEC_out_PPEc_S, 0. ) * wtspec.in_dai;
wtspec.out_PPEc_S_1 = spec.getSpecValue( SPEC_out_PPEc_S, 0. );

// 中間冷房入力(電力)
wtspec.out_PPEc_C   = spec.getSpecValue( SPEC_out_PPEc_C, 0. ) * wtspec.in_dai;
wtspec.out_PPEc_C_1 = spec.getSpecValue( SPEC_out_PPEc_C, 0. );

// 定格暖房能力
wtspec.out_Qh_S   = spec.getSpecValue( SPEC_out_Qh_S, 0. ) * wtspec.in_dai;
wtspec.out_Qh_S_1 = spec.getSpecValue( SPEC_out_Qh_S, 0. );

// 中間暖房能力
wtspec.out_Qh_C   = spec.getSpecValue( SPEC_out_Qh_C, 0. ) * wtspec.in_dai;
wtspec.out_Qh_C_1 = spec.getSpecValue( SPEC_out_Qh_C, 0. );

// 低温暖房能力
wtspec.out_Qh_L   = spec.getSpecValue( SPEC_out_Qh_L, 0. ) * wtspec.in_dai;
wtspec.out_Qh_L_1 = spec.getSpecValue( SPEC_out_Qh_L, 0. );
```

```
      if( wtspec.out_Qh_L <= 0 && wtspec.out_Qh_S>0 ){
         BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
               SystemMessageConstants.CAL_ERROR2 ,
               new String[]{ "( "+wtspec.out_Qh_L+"/"+wtspec.moduleName+" )[ ウォールスルー ]", "低温暖房能力を設定してください"} ));

      }

      // 定格暖房入力(電力)
      wtspec.out_PPEh_S   = spec.getSpecValue( SPEC_out_PPEh_S, 0. ) * wtspec.in_dai;
      wtspec.out_PPEh_S_1 = spec.getSpecValue( SPEC_out_PPEh_S, 0. );

      // 中間暖房入力( 電力)
      wtspec.out_PPEh_C   = spec.getSpecValue( SPEC_out_PPEh_C, 0. ) * wtspec.in_dai;
      wtspec.out_PPEh_C_1 = spec.getSpecValue( SPEC_out_PPEh_C, 0. );

      // 低温暖房入力(電力)
      wtspec.out_PPEh_L   = spec.getSpecValue( SPEC_out_PPEh_L, 0. ) * wtspec.in_dai;
      wtspec.out_PPEh_L_1 = spec.getSpecValue( SPEC_out_PPEh_L, 0. );

      if( wtspec.out_PPEh_L <= 0 && wtspec.out_PPEh_S>0 ){
         BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
               SystemMessageConstants.CAL_ERROR2 ,
               new String[]{ "( "+wtspec.out_PPEh_L+"/"+wtspec.moduleName+" )[ ウォールスルー ]", "低温暖房入力(電力)を設定してください"} ));

      }

/*
      // クランクケースヒータ(運転時)
      wtspec.out_CLEa_R = spec.getSpecValue( this.SPEC_out_CLEa_R, 0. )*wtspec.in_dai;

      // クランクケースヒータ(停止時)
      wtspec.out_CLEa_S = spec.getSpecValue( this.SPEC_out_CLEa_S, 0. )*wtspec.in_dai;

      // 待機電力(運転時)
      wtspec.out_TKEa_R = spec.getSpecValue( this.SPEC_out_TKEa_R, 0. )*wtspec.in_dai;

      // 待機電力(待機時)
      wtspec.out_TKEa_T = spec.getSpecValue( this.SPEC_out_TKEa_T, 0. )*wtspec.in_dai;

      // 待機電力(停止時)
      wtspec.out_TKEa_S = spec.getSpecValue( this.SPEC_out_TKEa_S, 0. )*wtspec.in_dai;

      //SPEC_fRate_airEA   = "風量[g/s]";
      wtspec.fRate_airEA = spec.getSpecValue( this.SPEC_fRate_airEA, 0. )*wtspec.in_dai;
*/
      // 機器起動停止負荷率
      wtspec.in_run_stop = spec.getSpecValue( SPEC_in_run_stop, 0.3 );

      // 相数
      wtspec.phase = spec.getSpecValue( SPEC_Phase, 3 );

      // 電圧
      wtspec.voltage = spec.getSpecValue( SPEC_Voltage, 200. );

      // 周波数
      wtspec.frequency = spec.getSpecValue( SPEC_Frequency, 50. );

      // 力率
      wtspec.powerFactor = spec.getSpecValue( SPEC_PowerFactor, 0.8 );



      if( wtspec.out_Qc_S < 0.001 ){
         wtspec.fmc = 0.0;
      }else{
         wtspec.fmc = wtspec.out_Qc_C / wtspec.out_Qc_S;
      }
      if( wtspec.out_PPEc_S < 0.001 ){
```

```
            wtspec.pmc = 0.0;
        }else{
            wtspec.pmc = wtspec.out_PPEc_C / wtspec.out_PPEc_S;
        }

        if( wtspec.out_Qh_S < 0.001 ){
            wtspec.fmh = 0.0;
        }else{
            wtspec.fmh = wtspec.out_Qh_C / wtspec.out_Qh_S;
        }
        if( wtspec.out_PPEh_S < 0.001 ){
            wtspec.pmh = 0.0;
        }else{
            wtspec.pmh = wtspec.out_PPEh_C / wtspec.out_PPEh_S;
        }

        if( wtspec.pmc > 0.9999 ) {
            wtspec.pmc = 0.0;
            wtspec.fmc = 0.0;
        }
        if( wtspec.pmh > 0.9999 ) {
            wtspec.pmh = 0.0;
            wtspec.fmh = 0.0;
        }

        //    機器名称
        //wtspec.name = spec.getSpecValue( this.SPEC_name, wtspec.moduleName );

/*      // 機器番号
        wtspec.m_no = spec.getSpecValue( this.SPEC_m_no, "" );
            try{
                if(m_no.substring(0 , 1).equals("+")==true){
                    saijohatsu=Double.parseDouble(m_no.substring(1 , 3));
                }else{
                    saijohatsu=50.0;
                }
            }
            catch(NumberFormatException e){
                saijohatsu=50.0;
            }

        // 機器型式
        wtspec.m_kt = spec.getSpecValue( this.SPEC_m_kt, "" );
*/

        // 定格風量
        wtspec.in_Va_S   = spec.getSpecValue( SPEC_in_Va_S, 0. ) * wtspec.in_dai;
        wtspec.in_Va_S_1 = spec.getSpecValue( SPEC_in_Va_S, 0. );

        if( wtspec.in_Va_S <= 0 ){
            BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                    SystemMessageConstants.CAL_ERROR2 ,
                    new String[]{ "( "+wtspec.in_Va_S+"/"+wtspec.moduleName+" )[ ウォールスルー ]", "風量を設定してください
"} ));

        }
        // 定格ファン消費電力
        wtspec.in_PPEa_S   = spec.getSpecValue( SPEC_in_PPEa_S, 0. ) * wtspec.in_dai;
        wtspec.in_PPEa_S_1 = spec.getSpecValue( SPEC_in_PPEa_S, 0. );

/*
        // 待機電力(運転時)
        wtspec.in_TKEa_R = spec.getSpecValue( this.SPEC_in_TKEa_R, 0. )*in_dai;

        // 待機電力(停止時)
        wtspec.in_TKEa_S = spec.getSpecValue( this.SPEC_in_THEa_S, 0. )*in_dai;

        // 機器起動停止負荷率
        wtspec.in_run_stop = spec.getSpecValue( this.SPEC_in_run_stop, 0.3 )*100.0;
*/

        // 定格加湿能力
```

```
    wtspec.in_HUM_mx   = spec.getSpecValue( SPEC_in_HUM_mx, 0. ) * wtspec.in_dai;
    wtspec.in_HUM_mx_1 = spec.getSpecValue( SPEC_in_HUM_mx, 0. );

    // 加湿飽和効率
    wtspec.in_HUM_rt = spec.getSpecValue( SPEC_in_HUM_rt, 0.7 )*100.0;

    // 加湿器ONOFF湿度
    wtspec.in_HUM_on = spec.getSpecValue( SPEC_in_HUM_on, 0.4 )*100.0;

    // 取入外気量
    wtspec.in_OA_S = spec.getSpecValue( SPEC_in_OA_S, 0. ) * wtspec.in_dai;

    // 全熱交換器効率
    wtspec.in_EX_S = spec.getSpecValue( SPEC_in_EX_S, 0.5 );

    //SPEC_isHexbypass = "熱交バイパスあり";
    wtspec.isHexbypass = spec.getSpecValue( SPEC_isHexbypass, false );

    // 全熱交換器消費電力[W]
    wtspec.in_EX_PE = spec.getSpecValue( SPEC_in_EX_PE, 0. ) * wtspec.in_dai;


    //isGVisibleを取得
    wtspec.isGVisible = spec.getSpecValue( SPEC_isGVisible, false );

    //isRecordを取得
    wtspec.isRecord = spec.getSpecValue( SPEC_isRecord, false );

    //低負荷領域の計算方法
    wtspec.calcTypeLowerRangeLoad = spec.getSpecValue( SPEC_CalcTypeLowerRangeLoad, "0_発停運転" );
    //calcTypeLowerRangeLoad
    //  0_発停運転
    //  1_下限入力値固定
    //  2_下限COP値固定
    //  3_下限入力値と中間切片
    if( wtspec.calcTypeLowerRangeLoad.equals( "0_発停運転" )){
       wtspec.num_calcTypeLowerRangeLoad = 0;

    }else if(wtspec.calcTypeLowerRangeLoad.equals( "1_下限入力値固定" )){
       wtspec.num_calcTypeLowerRangeLoad = 1;

    }else if(wtspec.calcTypeLowerRangeLoad.equals( "2_下限COP値固定" )){
       wtspec.num_calcTypeLowerRangeLoad = 2;

    }else if(wtspec.calcTypeLowerRangeLoad.equals( "3_下限入力値と中間切片" )){
       wtspec.num_calcTypeLowerRangeLoad = 3;

    }else{
       wtspec.num_calcTypeLowerRangeLoad = 0;
    }

    //isAdjust2012 = "台数を調整する";//
    wtspec.isAdjust2012 = spec.getSpecValue( SPEC_isAdjust2012, false );

    // 調整の計算ステップ数
    wtspec.numAdjustSteps = spec.getSpecValue( SPEC_NumAdjustSteps, 18 );

    wtspec.daiListc = new LinkedList<Double>();
    wtspec.daiListh = new LinkedList<Double>();
    for( int i=0; i<wtspec.numAdjustSteps; i++ ){
       wtspec.daiListc.add( 0. );
       wtspec.daiListh.add( 0. );
    }

}


//@Override
public static void initialize( Map<String, String> mapState, AbstractBestModule abs, IBestStateMessage stateNodes,
       IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
       WallThroughSpec2015 wtspec ) {
    //System.out.println(wtspec.moduleName + " BuillMultiOutイニシャライズ");
```

```
        //状態ノードを受け取る
        //super.sm = stateNodes;
        //制御ノードを受け取る
        //super.cm = commandNodes;
        //記録ノードを受け取る
        //super.rm = recordNodes;

        //eleIn
        wtspec.eleIn = BestElectricity.bindnode( mapState, stateNodes, S_NODE_eleIn );

        //airInOA
        wtspec.airInOA = BestAir.bindnode( mapState, stateNodes, S_NODE_airInOA );
        wtspec.airInOA.setMaxFlowRate( wtspec.in_OA_S );//20150423

        //airOutEA
        wtspec.airOutEA = BestAir.bindnode( mapState, stateNodes, S_NODE_airOutEA );
        wtspec.airOutEA.setMaxFlowRate( wtspec.fRate_airEA );//20150423

        //グラフ表示の準備
        if( wtspec.isGVisible ){
           wtspec.gBMout = new GraphJFrameBuilMultiOut_S20101212( wtspec.name, 300, wtspec.out_Qc_S * 1.5, 0 );
        }



        //airOutRM
        wtspec.airOutSA = BestAir.bindnode( mapState, stateNodes, S_NODE_airOutSA );
        wtspec.airOutSA.setMaxFlowRate( wtspec.in_Va_S );//20150423
        if( wtspec.isUseZoneAir ){
           wtspec.zoneAirforSSinside.checkNumberAirChange( wtspec.in_Va_S, "in_Va_S", wtspec.moduleName, "initialize()",
wtspec.name, wtspec.isRecord, wtspec.message);
           wtspec.zoneAirforSSinside._initialize( );
        }

        //watInCW
        wtspec.watInCW = BestWater.bindnode( mapState, stateNodes, S_NODE_watInCW );

        //watOutD
        wtspec.watOutD = BestWater.bindnode( mapState, stateNodes, S_NODE_watOutD );


        wtspec.airInRA = BestAir.bindnode( mapState, stateNodes, S_NODE_airInRA );
        wtspec.airInRA.setMaxFlowRate( wtspec.in_Va_S );//20150423

        //S_NODE_valInPID
        wtspec.PIDrate = BestValue.bindnode( mapState, stateNodes, S_NODE_valInPID );


        //機器特性ファイル
        wtspec.filenames[0]  = wtspec.kiki_file;
        wtspec.equipmentName = wtspec.kiki_file;
        wtspec.pacDB = new PACDBManager( wtspec.path, wtspec.filenames );
        wtspec.pacDB.setEquipmentName( wtspec.equipmentName );

        //機器特性式の適用上下限値
        wtspec.maxDBOA_C_FACK = wtspec.pacDB.getformulaRangeMax( wtspec.pacDB.getFACKcta() );//特性/冷房/外気乾球温度/上限
値
        wtspec.minDBOA_C_FACK = wtspec.pacDB.getformulaRangeMin( wtspec.pacDB.getFACKcta() );//特性/冷房/外気乾球温度/下限
値
        wtspec.maxWBRA_C_FACK = wtspec.pacDB.getformulaRangeMax( wtspec.pacDB.getFACKcti() );//特性/冷房/室内湿球温度/上限
値
        wtspec.minWBRA_C_FACK = wtspec.pacDB.getformulaRangeMin( wtspec.pacDB.getFACKcti() );//特性/冷房/室内湿球温度/下限
値

        wtspec.maxWBOA_H_FACK = wtspec.pacDB.getformulaRangeMax( wtspec.pacDB.getFACKhta() );//特性/暖房/外気湿球温度/上限
値
        wtspec.minWBOA_H_FACK = wtspec.pacDB.getformulaRangeMin( wtspec.pacDB.getFACKhta() );//特性/暖房/外気湿球温度/下限
値
        wtspec.maxDBRA_H_FACK = wtspec.pacDB.getformulaRangeMax( wtspec.pacDB.getFACKhti() );//特性/暖房/室内乾球温度/上限
値
        wtspec.minDBRA_H_FACK = wtspec.pacDB.getformulaRangeMin( wtspec.pacDB.getFACKhti() );//特性/暖房/室内乾球温度/下限
値
```

```
        //グラフ表示の準備
        if( wtspec.isGVisible ){
            wtspec.gBMIn = new GraphJFrameBuilMultiIn_S20101212( wtspec.name, 300, wtspec.out_Qc_S * 1.5, wtspec.in_Va_S *
1.5 );
        }

    }




    /**
     * ウォールスルーEHP計算及び結果出力/各室内機よりも先に動きます。
     */
    //@Override
    public static void outputs_calc( AbstractBestModule abs, IBestStateMessage stateNodes,
            IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
            WallThroughSpec2015 wtspec ) {

        if (null == commandNodes || null == stateNodes ){
            return;
        }

        /**
         * 入力
         */
        //---20060211-----------------------------------------------
        wtspec.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
        wtspec.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
        wtspec.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
        wtspec.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
//      allSheat = 0.0;
//      allTheat = 0.0;
//      DBra_max = -100.0;
//      WBra_min = 1000.0;
//      Lratemin = 100.0;

        if(wtspec.dbflg==0){
            //filenames[0]=RMdata.get_kiki();
            //equipmentName=RMdata.get_kiki();
            //pacDB=new PACDBManager(path,filenames);
            //pacDB.setEquipmentName(equipmentName);
        }
        wtspec.dbflg = 1;

        //
        if( wtspec.isUseZoneAir ){
            wtspec.zoneAirforSSinside._outputSetRA( wtspec.airInRA );
        }else{
            wtspec.airInRA = (BestAir)stateNodes.getState(abs.getConnectionNode( S_NODE_airInRA ));
        }

        wtspec.airInOA = (BestAir)stateNodes.getState(abs.getConnectionNode( S_NODE_airInOA ));

        wtspec.swcIn    = commandNodes.getCommand(abs.getConnectionNode( C_NODE_swcIn ));
        wtspec.swcInOA  = commandNodes.getCommand(abs.getConnectionNode( C_NODE_swcInOA ));
        wtspec.modIn    = commandNodes.getCommand(abs.getConnectionNode( C_NODE_modIn ));
        wtspec.modInPID = commandNodes.getCommand(abs.getConnectionNode( C_NODE_modInPID ));

        /**
         * 入力
         */

        //状態ノードから室内空気を設定
        //室内乾球温度を設定
        wtspec.DBra = wtspec.airInRA.getTempDB();

        //室内絶対湿度を設定
        wtspec.XGra = wtspec.airInRA.getHumi();
```

```
//室内湿球温度を設定
wtspec.WBra = wtspec.airInRA.getTempWB();

//室内CO2濃度を設定
wtspec.CO2ppmra = wtspec.airInRA.getCO2ppm();

//    System.out.println("DBin="+DBin+" "+XGin+"  "+WBin+"  ");

//外気乾球温度設定
wtspec.DBoa = wtspec.airInOA.getTempDB();

//外気絶対湿度を設定
wtspec.XGoa = wtspec.airInOA.getHumi();

//外気湿球温度を設定
wtspec.WBoa = wtspec.airInOA.getTempWB();
if( wtspec.WBoa < -9000.0 ) wtspec.WBoa = WB_cal(wtspec.DBoa, wtspec.XGoa);

//外気CO2濃度を設定
wtspec.CO2ppmoa = wtspec.airInOA.getCO2ppm();

//PID操作量
wtspec.PIDrate = (BestValue)stateNodes.getState( abs.getConnectionNode( S_NODE_valInPID ));

//    System.out.println("mState="+mState+" "+onOff+"  "+mode);
//    System.out.println("DBoa="+DBoa+" "+XGoa+"  "+WBoa);


//
if( Airswc.isOFF( wtspec.swcIn )){
   //停止
   wtspec.mState = 0;
   if( wtspec.isRecord ){
      wtspec.message.append( "(C)停止");
   }
}else if( Airswc.isON( wtspec.swcIn )){
   //運転
   if( Airmod.isCOOL( wtspec.modIn ) && Airmod.isHEAT( wtspec.modIn ) ){
      //冷暖同時
      if( Airmod.isCOOL( wtspec.modInPID ) ){
         wtspec.mState = 1;
         if( wtspec.isRecord ){
            wtspec.message.append( "(C)PID冷暖同時の冷房");
         }
      }else if(  Airmod.isHEAT( wtspec.modInPID ) ){
         wtspec.mState = 2;
         if( wtspec.isRecord ){
            wtspec.message.append( "(C)PID冷暖同時の暖房");
         }
      }else{
         wtspec.mState = 1;
      }
   }else if( Airmod.isCOOL( wtspec.modIn )){
      //冷房運転
      wtspec.mState = 1;
      if( wtspec.isRecord ){
         wtspec.message.append( "(C)冷房");
      }
   }else if( Airmod.isHEAT( wtspec.modIn )){
      //暖房運転
      wtspec.mState = 2;
      if( wtspec.isRecord ){
         wtspec.message.append( "(C)暖房");
      }
   }else if( Airmod.isVENTILATE( wtspec.modIn )){
      //換気モード
      wtspec.mState = 3;
      if( wtspec.isRecord ){
         wtspec.message.append( "(C)換気");
      }
   }else{
```

```
            //停止
            wtspec.mState = 0;
            if( wtspec.isRecord ){
               wtspec.message.append( "(C)停止");
            }
         }
         BestAir.checkOpeData( wtspec.airInRA, "airInRA", wtspec.moduleName, "outputs()", wtspec.name, wtspec.isRecord,
wtspec.message );
      }



      //計算の切替

      //室内機側の計算
      switch(wtspec.mState){
      case 0:
         //停止
         if( wtspec.isRecord ){
            wtspec.message.append("(C)停止");
         }
         calc_Stop_In( wtspec );
         //
         break;
      case 1:
         //冷房
         if( wtspec.out_Qc_S_1 > 0 ){
            if( wtspec.isRecord ){
               wtspec.message.append("(C)冷房運転");
            }
            cooling_cal( wtspec );
            wtspec.PPE_in = wtspec.in_PPEa_S + wtspec.in_TKEa_R + wtspec.in_EX_PE;//20130717
//          wtspec.Sh_Load = 0.0;
//          wtspec.Th_Load = 0.0;
         }else{
            if( wtspec.isRecord ){
               wtspec.message.append("(C)冷能力=0停止");//20130625
            }
            calc_Stop_In( wtspec );
            wtspec.mState = 0;
         }
         break;
      case 2:
         //暖房
         if( wtspec.out_Qh_S_1 > 0 ){
            if( wtspec.isRecord ){
               wtspec.message.append("(C)暖房運転");
            }
            heating_cal( wtspec );
            wtspec.PPE_in = wtspec.in_PPEa_S + wtspec.in_TKEa_R + wtspec.in_EX_PE;//20130717
//          wtspec.Sc_Load = 0.0;
//          wtspec.Tc_Load = 0.0;
         }else{
            if( wtspec.isRecord ){
               wtspec.message.append("(C)暖能力=0停止");//20130625
            }
            calc_Stop_In( wtspec );
            wtspec.mState = 0;
         }
         break;
      case 3:
         //換気
         if( wtspec.isRecord ){
            wtspec.message.append("(C)換気運転");
         }
         ventilation_cal( wtspec );
         wtspec.PPE_in = wtspec.in_PPEa_S + wtspec.in_TKEa_R + wtspec.in_EX_PE;//20130717
//       wtspec.S_Load = 0.0;
//       wtspec.T_Load = 0.0;
//       wtspec.Sc_Load = 0.0;
//       wtspec.Sh_Load = 0.0;
//       wtspec.Tc_Load = 0.0;
```

```
//        wtspec.Th_Load = 0.0;
          break;
       default:
          if( wtspec.isRecord ){
             wtspec.message.append("(C)運転モード？停止");//20130625
          }
          calc_Stop_In( wtspec );
          wtspec.mState = 0;
          //System.out.println(wtspec.moduleName + ">>Error<< onOff*modeが範囲外");
       }

//     wtspec.allSc_heat = wtspec.Sc_Load;
//     wtspec.allSh_heat = wtspec.Sh_Load;
//     wtspec.allTc_heat = wtspec.Tc_Load;
//     wtspec.allTh_heat = wtspec.Th_Load;

       //室外機側の計算
       switch(wtspec.mState){
       case 0:
          //停止
          if( wtspec.isRecord ){
             wtspec.message.append("(C)停止");
          }
          calc_Stop_Out( wtspec );
          break;
       case 1:
          //冷房
//        if( wtspec.out_Qc_S_1 > 0 ){
//           if( wtspec.isRecord ){
//              wtspec.message.append("(C)冷房運転");
//           }
//           cooling_cal_Out( wtspec );
//        }else{
//           if( wtspec.isRecord ){
//              wtspec.message.append("(C)冷能力=0停止");//20130625
//           }
//           calc_Stop_Out( wtspec );
//           wtspec.mState = 0;
//        }

          break;
       case 2:
          //暖房
//        if( wtspec.out_Qh_S_1 > 0 ){
//           if( wtspec.isRecord ){
//              wtspec.message.append("(C)暖房運転");
//           }
//           heating_cal_Out( wtspec );
//        }else{
//           if( wtspec.isRecord ){
//              wtspec.message.append("(C)暖能力=0停止");//20130625
//           }
//           calc_Stop_Out( wtspec );
//           wtspec.mState = 0;
//        }

          break;
       case 3:
          //換気
          if( wtspec.isRecord ){
             wtspec.message.append("(C)換気運転");
          }
          wtspec.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
          wtspec.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
          wtspec.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
          wtspec.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
          wtspec.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
          wtspec.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
          wtspec.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
          wtspec.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
//        wtspec.CAPrateC = 0.0;
//        wtspec.CAPrateH = 0.0;
```

```
            break;
        default:
            if( wtspec.isRecord ){
                wtspec.message.append("(C)運転モード？停止");//20130625
            }
            calc_Stop_Out( wtspec );
            wtspec.mState = 0;
            //System.out.println(wtspec.moduleName + ">>Error<< onOff*modeが範囲外");
        }


        stateNodes.setState( abs.getConnectionNode( S_NODE_airOutEA ), wtspec.airOutEA );

        wtspec.E_PPE_in = wtspec.PPE_in * Math.pow( 1/wtspec.powerFactor/ wtspec.powerFactor - 1, 0.5 );
        wtspec.E_PPE_out= wtspec.PPE_out* Math.pow( 1/wtspec.powerFactor/ wtspec.powerFactor - 1, 0.5 );
        wtspec.eleIn.setActivePower( wtspec.PPE_out + wtspec.PPE_in );
        wtspec.eleIn.setReactivePower(wtspec.E_PPE_out + wtspec.E_PPE_in );
        wtspec.eleIn.setPhase( wtspec.phase );
        wtspec.eleIn.setVoltage( wtspec.voltage );
        wtspec.eleIn.setFrequency( wtspec.frequency );

        //wtspec.CAPrate = wtspec.RMdata.get_CAPrate();
//      wtspec.CAPrateC = 1.;
//      wtspec.CAPrateH = 1.;

//      wtspec.Sc_Load_RM = wtspec.Sc_Load;
//      wtspec.Tc_Load_RM = wtspec.Tc_Load;
//      wtspec.Sh_Load_RM = wtspec.Sh_Load;
//      wtspec.Th_Load_RM = wtspec.Th_Load;
        //    System.out.println("CAPrate="+CAPrate+" "+S_Load_RM+" "+S_Load+ " "+mState);

        //室内機側の再計算
        switch( wtspec.mState ){
        case 0:
            //停止
            wtspec.DBout = wtspec.DBin;//乾球温度
            wtspec.XGout = wtspec.XGin;//絶対湿度
            wtspec.GW = 0.0;
            wtspec.D_Wrate = 0.0;
            wtspec.CW_Wrate = 0.0;
            break;
        case 1:
            //冷房
            wtspec.GW = wtspec.in_Va_S;
//          cooling_supply( wtspec );
            wtspec.D_Wrate = ( wtspec.XGin - wtspec.XGout ) * wtspec.GW;//g/s
            if( wtspec.D_Wrate < 0.0 ){
                wtspec.D_Wrate = 0.0;//追加090619
            }
            wtspec.CW_Wrate = 0.0;
            break;
        case 2:
            //暖房
            wtspec.GW = wtspec.in_Va_S;
//          heating_supply( wtspec );
            wtspec.D_Wrate = 0.0;
            break;
        case 3:
            //換気
            wtspec.GW = wtspec.in_Va_S;
            ventilation_supply( wtspec );
            wtspec.D_Wrate = 0.0;
            //wtspec.S_Load_RM = 0.0;
            //wtspec.T_Load_RM = 0.0;
            break;
        default:
            System.out.println( wtspec.moduleName + ">>Error<< onOff*modeが範囲外");
        }
        //    System.out.println("DBout="+DBout);

        wtspec.airOutSA.setTempDB( wtspec.DBout );
        wtspec.airOutSA.setHumi( wtspec.XGout );
```

```
wtspec.airOutSA.setFlowRate( wtspec.GW );
wtspec.airOutSA.setCO2ppm( wtspec.CO2ppmout );//20131031

wtspec.watOutD.setFlowRate( wtspec.D_Wrate );
wtspec.watOutD.setTemp( wtspec.DBout );
wtspec.watInCW.setFlowRate( wtspec.CW_Wrate );
//    System.out.println("DBout="+DBout+" "+XGout+"  "+airOutRM.getFlowRate());

//出力側ノード
stateNodes.setState( abs.getConnectionNode( S_NODE_airOutSA ), wtspec.airOutSA);
stateNodes.setState( abs.getConnectionNode( S_NODE_eleIn ), wtspec.eleIn);
stateNodes.setState( abs.getConnectionNode( S_NODE_watOutD ), wtspec.watOutD);
stateNodes.setState( abs.getConnectionNode( S_NODE_watInCW ), wtspec.watInCW);//090407追加

//グラフデータ追加
if( wtspec.isGVisible ){
   wtspec.gBMIn.addData( BestTimeManager.getDateWeatherTime(),
         wtspec.allSc_heat,
         wtspec.allTc_heat,
         wtspec.allSh_heat,
         wtspec.allTh_heat,
         wtspec.PPE_in,
         wtspec.DBoa,
         wtspec.DBra,
         wtspec.DBin,
         wtspec.DBout,
         wtspec.XGoa,
         wtspec.XGra,
         wtspec.XGin,
         wtspec.XGout,
         wtspec.GW,
         wtspec.D_Wrate );
}


//出力設定

//グラフデータ追加
if( wtspec.isGVisible ){
   wtspec.gBMout.addData( BestTimeManager.getDateWeatherTime(),
   wtspec.Sc_Loadout,//室外機処理可能熱量（冷却）[W]
   wtspec.Sh_Loadout,//室外機処理可能熱量（加熱）[W]
   wtspec.allTc_heat,//室内機処理要求熱量（冷却）[W]
   wtspec.allTh_heat,//室内機処理要求熱量（加熱）[W]
   wtspec.PPE_out,
   wtspec.DBoa,
   wtspec.DBin,
   wtspec.XGoa,
   wtspec.XGin,
   1. );
}

//記録ノード
if( wtspec.isRecord && recordNodes != null ){
   record( abs, recordNodes, wtspec );
}

wtspec.message.setLength(0);



//************************----------------
if( wtspec.isAdjust2012 ){
   //移動平均の最大値で調整していく
   if( true ){
      wtspec.daiListc.removeLast();
      wtspec.daiListc.addFirst( wtspec.in_daicAdjust * wtspec.PIDrate.getValue() );
      //
      double sum_in_dai = 0;
      for( int i=0; i<wtspec.numAdjustSteps; i++ ){
         sum_in_dai += wtspec.daiListc.get( i );
      }
```

```
            wtspec.avecdaiAdjust = sum_in_dai / wtspec.numAdjustSteps;
            //
            if( sum_in_dai > wtspec.maxcAdjustdai * wtspec.numAdjustSteps ){
               wtspec.maxcAdjustdai = wtspec.avecdaiAdjust;

               wtspec.in_daicAdjust = wtspec.maxcAdjustdai;
               wtspec.in_run_stop_Num = wtspec.in_run_stop / wtspec.in_daicAdjust;

               wtspec.out_Qc_S   = wtspec.out_Qc_S_1   * wtspec.in_daicAdjust;//= "定格冷房能力";//kW
               wtspec.out_Qc_C   = wtspec.out_Qc_C_1   * wtspec.in_daicAdjust;//= "中間冷房能力";//kW
               wtspec.out_PPEc_S = wtspec.out_PPEc_S_1 * wtspec.in_daicAdjust;//= "定格冷房入力(電力)";//kW
               wtspec.out_PPEc_C = wtspec.out_PPEc_C_1 * wtspec.in_daicAdjust;//= "中間冷房入力(電力)";//kW

//             private double fRate_airEAc ;//風量[g/s]
//             private double fRate_airEAh ;//風量[g/s]

               wtspec.in_Va_S   = wtspec.in_Va_S_1   * wtspec.in_daicAdjust;//= "定格風量";//m3/h
               wtspec.in_PPEa_S = wtspec.in_PPEa_S_1 * wtspec.in_daicAdjust;//= "定格ファン消費電力";//W

               wtspec.in_HUM_mx = wtspec.in_HUM_mx_1 * wtspec.in_daicAdjust;//= "定格加湿能力";//kg/h

               wtspec.airInRA.setFlowRate(wtspec.in_Va_S);
               wtspec.airOutSA.setFlowRate(wtspec.in_Va_S);
            }
         }


         if( true ){
            wtspec.daiListh.removeLast();
            wtspec.daiListh.addFirst( wtspec.in_daihAdjust * wtspec.PIDrate.getValue() );
            //
            double sum_in_dai = 0;
            for( int i=0; i<wtspec.numAdjustSteps; i++ ){
               sum_in_dai += wtspec.daiListh.get( i );
            }
            wtspec.avehdaiAdjust = sum_in_dai / wtspec.numAdjustSteps;
            //
            if( sum_in_dai > wtspec.maxhAdjustdai * wtspec.numAdjustSteps ){
               wtspec.maxhAdjustdai = wtspec.avehdaiAdjust;

               wtspec.in_daihAdjust = wtspec.maxhAdjustdai;
               wtspec.in_run_stop_Num = wtspec.in_run_stop / wtspec.in_daihAdjust;

               wtspec.out_Qh_S   = wtspec.out_Qh_S_1   * wtspec.in_daihAdjust;//= "定格暖房能力";//kW
               wtspec.out_Qh_C   = wtspec.out_Qh_C_1   * wtspec.in_daihAdjust;//= "中間暖房能力";//kW
               wtspec.out_Qh_L   = wtspec.out_Qh_L_1   * wtspec.in_daihAdjust;//= "低温暖房能力";//kW
               wtspec.out_PPEh_S = wtspec.out_PPEh_S_1 * wtspec.in_daihAdjust;//= "定格暖房入力(電力)";//kW
               wtspec.out_PPEh_C = wtspec.out_PPEh_C_1 * wtspec.in_daihAdjust;//= "中間暖房入力(電力)";//kW
               wtspec.out_PPEh_L = wtspec.out_PPEh_L_1 * wtspec.in_daihAdjust;//= "低温暖房入力(電力)";//kW

//             private double fRate_airEAc ;//風量[g/s]
//             private double fRate_airEAh ;//風量[g/s]

               wtspec.in_Va_S   = wtspec.in_Va_S_1   * wtspec.in_daihAdjust;//= "定格風量";//m3/h
               wtspec.in_PPEa_S = wtspec.in_PPEa_S_1 * wtspec.in_daihAdjust;//= "定格ファン消費電力";//W

               wtspec.in_HUM_mx = wtspec.in_HUM_mx_1 * wtspec.in_daihAdjust;//= "定格加湿能力";//kg/h

               wtspec.airInRA.setFlowRate(wtspec.in_Va_S);
               wtspec.airOutSA.setFlowRate(wtspec.in_Va_S);
            }

         }
      }
      //*************************----------------

   }



   private static void calc_Stop_In( WallThroughSpec2015 wtspec ){
      //出口空気の状態は入口空気の状態とする
```

```
        wtspec.airOutSA.copyAllvalState( wtspec.airInRA );
        wtspec.airOutSA.setFlowRate(0.0);
//      wtspec.S_Load = 0.0;
//      wtspec.T_Load = 0.0;
//      wtspec.Sc_Load = 0.0;
//      wtspec.Sh_Load = 0.0;
//      wtspec.Tc_Load = 0.0;
//      wtspec.Th_Load = 0.0;
        //LPrate=1.0;
        wtspec.PPE_in = wtspec.in_TKEa_S;
        wtspec.DBin = wtspec.DBra;
        wtspec.XGin = wtspec.XGra;
        wtspec.WBin = wtspec.WBra;
        wtspec.CO2ppmin = wtspec.CO2ppmra;//20131031
    }

    private static void calc_Stop_Out( WallThroughSpec2015 wtspec ){
        wtspec.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
        wtspec.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
        wtspec.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
        wtspec.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
        wtspec.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
        wtspec.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
        wtspec.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
        wtspec.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
        wtspec.PPE_out = wtspec.out_TKEa_S;
    }


    /**
     * 記録
     */
    private static void record( AbstractBestModule abs, IBestRecordMessage recordNodes, WallThroughSpec2015 wtspec ){

        if( recordNodes != null ){
            if( CheckPrintModule.isPrintMessage ){
                //message
                recordNodes.setRecord(abs.getConnectionNode(R_NODE),
                        RECORD_message, wtspec.name, wtspec.message.toString() );
            }

            if( CheckPrintModule.isPrintEnergy ){
                //記録ノードに室外機消費電力を設定
                recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID4, wtspec.name,
AirFormat.df_2(wtspec.PPE_out));
            }

            if( CheckPrintModule.isPrintLoad ){
                //記録ノードに室外機処理熱量（全熱）を設定
                if( wtspec.Tc_Loadout > 0 ){
                    recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_qT, wtspec.name, AirFormat.df_2(-
wtspec.Tc_Loadout ));//室外機処理全熱量合計#W#熱量
                }else{
                    recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_qT, wtspec.name,
AirFormat.df_2(wtspec.Th_Loadout ));//室外機処理全熱量合計#W#熱量
                }
            }

            if( CheckPrintModule.isPrintStateOut ){
                //記録ノードに吹出口乾球温度を設定
                recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID4i, wtspec.name, AirFormat.df_2(wtspec.DBout));
                //記録ノードに吹出口絶対湿度を設定
                recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID5i, wtspec.name, AirFormat.df_5(wtspec.XGout));
                //記録ノードに吹出口風量を設定
                recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID6i, wtspec.name, AirFormat.df_2(wtspec.GW));
                //記録ノードにドレン量を設定
                recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID7i, wtspec.name,
AirFormat.df_2(wtspec.D_Wrate));
            }

            if( CheckPrintModule.isPrintStateMy ){
                //記録ノードに室内機加熱能力補正比率を設定
```

```
//          recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID3c, wtspec.name,
AirFormat.df_2(wtspec.CAPrateC));
//          recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID3h, wtspec.name,
AirFormat.df_2(wtspec.CAPrateH));
            //記録ノードに室内機処理熱量（顕熱）を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID1c, wtspec.name,
AirFormat.df_2(wtspec.allSc_heat));//室内機冷却処理顕熱量合計#W#熱量
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID1h, wtspec.name,
AirFormat.df_2(wtspec.allSh_heat));//室内機加熱処理顕熱量合計#W#熱量
            //記録ノードに室内機処理熱量（全熱）を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID2c, wtspec.name,
AirFormat.df_2(wtspec.allTc_heat));//室内機冷却処理全熱量合計#W#熱量
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID2h, wtspec.name,
AirFormat.df_2(wtspec.allTh_heat));//室内機加熱処理全熱量合計#W#熱量
            //記録ノードに室外機処理可能熱量（全熱）を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID5c, wtspec.name,
AirFormat.df_2(wtspec.Sc_Loadout));//室外機冷却可能熱量/最大#W#熱量
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID5h, wtspec.name,
AirFormat.df_2(wtspec.Sh_Loadout));//室外機加熱可能熱量/最大#W#熱量
            //記録ノードに室内機処理熱量（顕熱）を設定
            //recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID1,  wtspec.name,
AirFormat.df_2(wtspec.S_Load_RM));//室内機処理顕熱量合計#W#熱量
            //記録ノードに室内機処理熱量（全熱）を設定
            //recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID2,  wtspec.name,
AirFormat.df_2(wtspec.T_Load_RM));//室内機処理全熱量合計#W#熱量
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_pidVal, wtspec.name,
AirFormat.df_4(wtspec.PIDrate.getValue()));

        }


        if( CheckPrintModule.isPrintStateIn ){
            //記録ノードに加湿給水量を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_ID8i, wtspec.name,
AirFormat.df_2(wtspec.CW_Wrate));
            //記録ノードに還気乾球温度を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_DBra, wtspec.name, AirFormat.df_2(wtspec.DBra));
            //記録ノードに還気絶対湿度を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_Xra, wtspec.name, AirFormat.df_5(wtspec.XGra));
            //記録ノードに還気湿球温度を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_WBra, wtspec.name, AirFormat.df_2(wtspec.WBra));
            //記録ノードに入口乾球温度を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_DBin, wtspec.name, AirFormat.df_2(wtspec.DBin));
            //記録ノードに入口絶対湿度を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_Xin, wtspec.name, AirFormat.df_5(wtspec.XGin));
            //記録ノードに入口湿球温度を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_WBin, wtspec.name, AirFormat.df_2(wtspec.WBin));
            //記録ノードに入口乾球温度を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_DBoa, wtspec.name, AirFormat.df_2(wtspec.DBoa));
            //記録ノードに入口絶対湿度を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_Xoa, wtspec.name, AirFormat.df_5(wtspec.XGoa));
            //記録ノードに入口湿球温度を設定
            recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_WBoa, wtspec.name, AirFormat.df_2(wtspec.WBoa));
        }

        if( CheckPrintModule.isPrintAdjust ){
            //
            if( wtspec.isAdjust2012 ){
                //記録ノードに室外機調整能力を設定
                recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_out_Qc_Adjust, wtspec.name,
wtspec.out_Qc_S );//室外機調整能力[W]
                recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_out_Qh_Adjust, wtspec.name,
wtspec.out_Qh_S );//室外機調整能力[W]
                //記録ノードに室外機調整能力を設定
                //recordNodes.setRecord(abs.getConnectionNode(R_NODE), "out_rcAdjust", wtspec.name,
wtspec.out_rcAdjust );//室外機調整率[-]
                //recordNodes.setRecord(abs.getConnectionNode(R_NODE), "out_rhAdjust", wtspec.name,
wtspec.out_rhAdjust );//室外機調整率[-]
                //記録ノードに調整台数を設定
                recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_in_daicAdjust, wtspec.name,
AirFormat.df_2(wtspec.in_daicAdjust) );
                recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_in_daihAdjust, wtspec.name,
```

```
AirFormat.df_2(wtspec.in_daihAdjust) );
                recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_avecdaiAdjust, wtspec.name,
AirFormat.df_2(wtspec.avecdaiAdjust) );
                recordNodes.setRecord(abs.getConnectionNode(R_NODE), RECORD_avehdaiAdjust, wtspec.name,
AirFormat.df_2(wtspec.avehdaiAdjust) );
             }
          }
       }
    }

    /**
     * ウォールスルーEHP計算/各室内機の集計後
     */

    //@Override
    public static void update(  WallThroughSpec2015 wtspec ) {
       if( wtspec.isUseZoneAir ){
          wtspec.zoneAirforSSinside._update( wtspec.airOutSA);
       }
    }

    private static void cooling_cal( WallThroughSpec2015 wtspec ){

       double D_fmc,D_pmc,alph,beta;
       double Rp;

       D_fmc    = wtspec.fmc;
       D_pmc    = wtspec.pmc;
       ////////////////////
       //室側の冷房計算
       double heatCapacity=0.0;
       double rCode0 = 0.0, rCode3 = 0.0;
       double rCode1 = 0.0, rCode2 = 0.0;
       double IAout1,DBout1,XGout1;
       double S_heat,T_heat;
       double HEXrate;

       HEXrate = wtspec.in_EX_S;

       //バイパス制御 追加 エンタルピで判断
       if( wtspec.isHexbypass == true ){   //20101212nino
          //全熱交換器バイパス有
          wtspec.IAra = Psychrometrics.FNH( wtspec.DBra, wtspec.XGra );//20130109
          wtspec.IAoa = Psychrometrics.FNH( wtspec.DBoa, wtspec.XGoa );

          if( wtspec.IAra > wtspec.IAoa && wtspec.PIDrate.getValue() > 0.001 && wtspec.HEXselect==1 ){//20130109
             //エンタルピ基準  でバイパス
             HEXrate = 0.0;
          }
          if( wtspec.DBra > wtspec.DBoa && wtspec.PIDrate.getValue() > 0.001 && wtspec.HEXselect==2 ){//20130109
             //温度/顕熱基準  でバイパス
             HEXrate = 0.0;
          }
       }

       //吸込み状態
       if( Airswc.isON( wtspec.swcInOA )){//20101212nino
          //外気導入時
          //吸込み状態を求める  RAとOAとHEX
          wtspec.DBin = (( wtspec.in_Va_S - wtspec.in_OA_S ) * wtspec.DBra + (( wtspec.DBoa - wtspec.DBra ) * ( 1.0 -
HEXrate ) + wtspec.DBra ) * wtspec.in_OA_S ) / wtspec.in_Va_S;//20130109
          wtspec.XGin = (( wtspec.in_Va_S - wtspec.in_OA_S ) * wtspec.XGra + (( wtspec.XGoa - wtspec.XGra ) * ( 1.0 -
HEXrate ) + wtspec.XGra ) * wtspec.in_OA_S ) / wtspec.in_Va_S;//20130109
          wtspec.CO2ppmin = (( wtspec.in_Va_S - wtspec.in_OA_S ) * wtspec.CO2ppmra + wtspec.in_OA_S * wtspec.CO2ppmoa ) /
wtspec.in_Va_S;//20131031
       }else{//20101212nino
          //外気導入時でない時はRAが吸込み状態
          wtspec.DBin = wtspec.DBra;//20130109
          wtspec.XGin = wtspec.XGra;//20130109
          wtspec.CO2ppmin = wtspec.CO2ppmra;//20131031
       }
       wtspec.WBin = Psychrometrics.FNWbtx( wtspec.DBin, wtspec.XGin );
```

```
        // System.out.println("C/DBin="+DBin+" "+XGin+"  ");
        // System.out.println("C/DBoa="+DBoa+" "+XGoa+"  ");
        // System.out.println("C/DBra="+DBra+" "+XGra+"  ");
        // System.out.println("  ");
        // System.out.println("DBra="+DBra+" "+XGra+"  ");

        //上下限のチェック 20120821nino
        double dbOA = wtspec.DBoa;
        double wbIn = wtspec.WBin;

        //特性式の適用範囲のチェックと上下限時の値の設定
        if( dbOA > wtspec.maxDBOA_C_FACK ){
           //外気乾球温度＞上限の時停止
           if( wtspec.isRecord ){
              wtspec.message.append( "(C)外気DB＞上限→停止");
           }
//            wtspec.S_Load = 0.0;
//            wtspec.T_Load = 0.0;
//            wtspec.Sc_Load = 0.0;
//            wtspec.Tc_Load = 0.0;
//            wtspec.Sh_Load = 0.0;
//            wtspec.Th_Load = 0.0;
           wtspec.Tc_Loadout = 0.0;//室外機実処理熱量[W]
           wtspec.Sc_Loadout = 0.0;//室外機処理可能熱量[W]
           wtspec.Th_Loadout = 0.0;//室外機実処理熱量[W]
           wtspec.Sh_Loadout = 0.0;//室外機処理可能熱量[W]
           wtspec.PPE_out = wtspec.out_TKEa_S;
           wtspec.airOutEA.setFlowRate( 0. );
           wtspec.airOutEA.setTempDB( wtspec.airInOA.getTempDB() );
           return;
        }
        if( dbOA < wtspec.minDBOA_C_FACK ){
           //外気乾球温度＜下限の時　下限値の特性で運転
           if( wtspec.isRecord ){
              wtspec.message.append( "(C)外気DB＜下限→下限値で特性計算");
           }
           dbOA = wtspec.minDBOA_C_FACK;
        }
        if( wbIn > wtspec.maxWBRA_C_FACK ){
           //室内湿球温度＞上限の時　上限値の特性で運転
           if( wtspec.isRecord ){
              wtspec.message.append( "(C)室WB＞上限→上限値で特性計算");
           }
           wbIn = wtspec.maxWBRA_C_FACK;
        }
        if( wbIn < wtspec.minWBRA_C_FACK ){
           //室内湿球温度＜下限の時　下限値の特性で運転
           if( wtspec.isRecord ){
              wtspec.message.append( "(C)室WB＜下限→下限値で特性計算");
           }
           wbIn = wtspec.minWBRA_C_FACK;
        }

        ////////////////////////////out
        //室温補正
        //能力補正
        rCode1 = wtspec.pacDB.getKcta( dbOA );//外気補正
        rCode2 = wtspec.pacDB.getKcti( wbIn );//室温補正

        heatCapacity = rCode1 * rCode2 * wtspec.out_Qc_S;//室外機処理可能熱量[W]
        //heatCapacity = rCode1 * rCode2 * wtspec.out_Qc_S;

        if( heatCapacity < 0 ){
           heatCapacity = 0;
        }


        wtspec.Rc = wtspec.pacDB.getKchp( wtspec.PIDrate.getValue() );
        if( wtspec.Rc > 1.0 ) {
           wtspec.Rc = 1.0;
        }
```

```
T_heat = heatCapacity * wtspec.Rc;//室外機実処理熱量[W]


//////////////////////////
// System.out.println( " Rc = "+ Rc );

//低負荷領域の計算仕分け:Rh
if( wtspec.Rc < wtspec.in_run_stop ){
   switch( wtspec.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = wtspec.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = wtspec.Rc;
}

alph = wtspec.pacDB.getAlphac( Rp );
if( D_pmc * D_fmc < 0.001 || wtspec.iType == wtspec.iType_STANDARD ){//標準（定速）はbeta=1
   beta = 1.0;
}else{
   beta = wtspec.pacDB.getBetac( Rp, D_fmc, D_pmc);
}
//       System.out.println("Rc="+Rc+" "+D_fmc+" "+D_pmc);

rCode0 = alph * beta;
rCode1 = wtspec.pacDB.getKchpid( Rp );
rCode2 = wtspec.pacDB.getKcwta( dbOA );

//       System.out.println("WBra="+WBra);

rCode3 = wtspec.pacDB.getKcwti( wbIn );

wtspec.PPE_out = rCode0 * rCode1 * rCode2 * rCode3 * wtspec.out_PPEc_S;

if( wtspec.iType == wtspec.iType_STANDARD ){
   wtspec.PPE_out /= ( 1.- 0.25 * ( 1. - Rp ));
}

//低負荷領域の計算仕分け:PPE
if( wtspec.Rc < wtspec.in_run_stop && Rp > 0 ){
   switch( wtspec.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      wtspec.PPE_out = wtspec.out_TKEa_S;;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      wtspec.PPE_out *= ( wtspec.Rc / Rp );
      break;

   case 3://  3_下限入力値と中間切片
      wtspec.PPE_out *= ( 0.5 + wtspec.Rc / Rp / 2. );
      break;

   default:
   }
}else{
   //何もしない
```

```
      }

      //airOutEA
      wtspec.airOutEA.setFlowRate( wtspec.fRate_airEA );
      wtspec.airOutEA.setTempDB( ( wtspec.Tc_Loadout + wtspec.PPE_out ) / wtspec.fRate_airEA * 1000. +
wtspec.airInOA.getTempDB() );

      if( wtspec.Rc < 0.0001 ){
//       wtspec.CAPrateC = 0.0;
//       wtspec.CAPrateH = 0.0;
         wtspec.Tc_Loadout = 0.0;//室外機実処理熱量[W]
         wtspec.Sc_Loadout = 0.0;//室外機処理可能熱量[W]
         wtspec.Th_Loadout = 0.0;//室外機実処理熱量[W]
         wtspec.Sh_Loadout = 0.0;//室外機処理可能熱量[W]
         wtspec.PPE_out = wtspec.out_TKEa_S;
         wtspec.airOutEA.setFlowRate( 0. );
         wtspec.airOutEA.setTempDB( wtspec.airInOA.getTempDB() );
      }else if( wtspec.Rc < wtspec.in_run_stop ){
         switch( wtspec.num_calcTypeLowerRangeLoad){
         case 0://  0_発停運転
//          wtspec.CAPrateC = 0.0;
//          wtspec.CAPrateH = 0.0;
            wtspec.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            wtspec.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            wtspec.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            wtspec.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            wtspec.PPE_out = wtspec.out_TKEa_S;;
            wtspec.airOutEA.setFlowRate( 0. );
            wtspec.airOutEA.setTempDB( wtspec.airInOA.getTempDB() );
            break;

         case 1://  1_下限入力値固定
            break;

         case 2://  2_下限COP値固定
            break;

         case 3://  3_下限入力値と中間切片
            break;

         default:
         }

      }

      //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

      ////////////////////////////////in
   // rCode1 = wtspec.pacDB.getKcta( dbOA );
   // rCode2 = wtspec.pacDB.getKcti( wbIn );

   // heatCapacity = rCode1 * rCode2 * wtspec.out_Qc_S;

   // if( heatCapacity < 0 ){
   //    heatCapacity = 0;
   // }

      // System.out.println("heatCapacity ="+rCode1 * rCode2 +" "+heatCapacity+"  ");
   // double rc = wtspec.pacDB.getKchp( wtspec.PIDrate.getValue() );

   // T_heat = heatCapacity * rc;

      wtspec.IAin   = Psychrometrics.FNH( wtspec.DBin, wtspec.XGin );
      IAout1 = wtspec.IAin - T_heat / wtspec.in_Va_S * 1000.0;
      DBout1 = Psychrometrics.FNDbrh( 90.0, IAout1 );//乾球温度

      S_heat = ( wtspec.DBin - DBout1 ) * wtspec.in_Va_S;

      if( S_heat > T_heat ){
         S_heat = T_heat;
         DBout1 = wtspec.DBin - T_heat / wtspec.in_Va_S;
         XGout1 = wtspec.XGin;
```

```
    }else{
        XGout1 = Psychrometrics.FNxtr( DBout1, 90.0);//絶対湿度
        if( wtspec.XGin < XGout1 ){
            XGout1 = wtspec.XGin;
            DBout1 = wtspec.DBin - T_heat / wtspec.in_Va_S;
        }
    }

    // System.out.println("IAra="+IAra+" "+IAout1+"  "+MHP_in_Vc_S);

    wtspec.DBout = DBout1;
    wtspec.XGout = XGout1;

    wtspec.T_C_heat = T_heat;

    wtspec.Sc_Loadout = heatCapacity;
    wtspec.Tc_Loadout = T_heat;
    wtspec.allTc_heat = T_heat;
    wtspec.allSc_heat = S_heat;

    if( wtspec.PIDrate.getValue() < wtspec.in_run_stop ){
        switch( wtspec.num_calcTypeLowerRangeLoad){
        case 0://  0_発停運転
            wtspec.Tc_Loadout = 0.0;//室外機実処理熱量[W]
            wtspec.Sc_Loadout = 0.0;//室外機処理可能熱量[W]
            wtspec.Th_Loadout = 0.0;//室外機実処理熱量[W]
            wtspec.Sh_Loadout = 0.0;//室外機処理可能熱量[W]
            wtspec.allTc_heat = 0;
            wtspec.allSc_heat = 0;
            wtspec.allTh_heat = 0;
            wtspec.allSh_heat = 0;
            break;

        case 1://  1_下限入力値固定
            break;

        case 2://  2_下限COP値固定
            break;

        case 3://  3_下限入力値と中間切片
            break;

        default:
        }
    }
    // System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate);

}



private static void heating_cal( WallThroughSpec2015 wtspec ){

    //室側の暖房計算
    double heatCapacity=0.0;
    double rCode0 = 0.0, rCode3 = 0.0;
    double rCode1 = 0.0, rCode2 = 0.0;
    //double IAout1,DBout1,XGout1;
    double S_heat;
    double HEXrate;

    double IAout;

    HEXrate = wtspec.in_EX_S;
    //バイパス制御 追加 20101212nino
    if( wtspec.isHexbypass == true ){
        wtspec.IAra = Psychrometrics.FNH( wtspec.DBra, wtspec.XGra );//20130109
        wtspec.IAoa = Psychrometrics.FNH( wtspec.DBoa, wtspec.XGoa );
        if( wtspec.IAra < wtspec.IAoa && wtspec.PIDrate.getValue() > 0.001 && wtspec.HEXselect == 1 ){//20130109
            HEXrate = 0.0;
        }
        if( wtspec.DBra < wtspec.DBoa && wtspec.PIDrate.getValue() > 0.001 && wtspec.HEXselect == 2 ){//20130109
```

```
            HEXrate = 0.0;
         }
         if( wtspec.IAra > wtspec.IAoa && wtspec.PIDrate.getValue() < 0.001 && wtspec.HEXselect == 1 ){//20130109
            HEXrate = 0.0;
         }
         if( wtspec.DBra > wtspec.DBoa && wtspec.PIDrate.getValue() < 0.001 && wtspec.HEXselect == 2 ){//20130109
            HEXrate = 0.0;
         }
      }

      if( Airswc.isON( wtspec.swcInOA )){//20101212nino
         wtspec.DBin = (( wtspec.in_Va_S - wtspec.in_OA_S ) * wtspec.DBra + ( wtspec.DBra - ( wtspec.DBra - wtspec.DBoa )
* ( 1.0 - HEXrate )) * wtspec.in_OA_S ) / wtspec.in_Va_S;//20130109
         wtspec.XGin = (( wtspec.in_Va_S - wtspec.in_OA_S ) * wtspec.XGra + ( wtspec.XGra - ( wtspec.XGra - wtspec.XGoa )
* ( 1.0 - HEXrate )) * wtspec.in_OA_S ) / wtspec.in_Va_S;//20130109
         wtspec.CO2ppmin = (( wtspec.in_Va_S - wtspec.in_OA_S ) * wtspec.CO2ppmra + wtspec.in_OA_S * wtspec.CO2ppmoa ) /
wtspec.in_Va_S;//20131031
      }else{//20101212nino
         wtspec.DBin = wtspec.DBra;//20130109
         wtspec.XGin = wtspec.XGra;//20130109
         wtspec.CO2ppmin = wtspec.CO2ppmra;//20131031
      }
      wtspec.WBin = Psychrometrics.FNWbtx(wtspec.DBin,wtspec.XGin);//20130109


      // System.out.println("H/DBin="+DBin+" "+XGin+"  ");
      // System.out.println("H/DBoa="+DBoa+" "+XGoa+"  ");
      // System.out.println("H/DBra="+DBra+" "+XGra+"  ");
      // System.out.println("  ");

      //上下限のチェック 20120821nino
      double wbOA = wtspec.WBoa;
      double dbIn = wtspec.DBin;

      if( wbOA > wtspec.maxWBOA_H_FACK ){
         //外気湿球温度＞上限の時　上限値の特性で運転
         if( wtspec.isRecord ){
            wtspec.message.append( "(C)外気WB＞上限→上限値で特性計算");
         }
         wbOA = wtspec.maxWBOA_H_FACK;
      }
      if( wbOA < wtspec.minWBOA_H_FACK ){
         //外気湿球温度＜下限の時　停止
         if( wtspec.isRecord ){
            wtspec.message.append( "(C)外気WB＜下限→停止");
         }

         wtspec.Tc_Loadout = 0.0;//室外機実処理熱量[W]
         wtspec.Sc_Loadout = 0.0;//室外機処理可能熱量[W]
         wtspec.Th_Loadout = 0.0;//室外機実処理熱量[W]
         wtspec.Sh_Loadout = 0.0;//室外機処理可能熱量[W]
         wtspec.PPE_out = wtspec.out_TKEa_S;;
         wtspec.airOutEA.setFlowRate( 0. );
         wtspec.airOutEA.setTempDB( wtspec.airInOA.getTempDB() );
         return;
      }
      if( dbIn > wtspec.maxDBRA_H_FACK ){
         //室内乾球温度＞上限の時　上限値の特性で運転
         if( wtspec.isRecord ){
            wtspec.message.append( "(C)室DB＞上限→上限値で特性計算");
         }
         dbIn = wtspec.maxDBRA_H_FACK;
      }
      if( dbIn < wtspec.minDBRA_H_FACK ){
         //室内乾球温度＜下限の時　下限値の特性で運転
         if( wtspec.isRecord ){
            wtspec.message.append( "(C)室DB＜下限→下限値で特性計算");
         }
         dbIn = wtspec.minDBRA_H_FACK;
      }
```

```
        ////////////////////////////////////////////out
        double D_fmh,D_pmh,D_out_Qh_S,D_out_PPEh_S,alph,beta;
        double Rp;

        D_fmh    = wtspec.fmh;
        D_pmh    = wtspec.pmh;
        D_out_Qh_S  = wtspec.out_Qh_S;
        D_out_PPEh_S= wtspec.out_PPEh_S;

        if( wtspec.WBoa < 4.5 && wtspec.out_Qh_L > 0.0001 ) { //低温入力の場合
              D_fmh    = wtspec.fmh;
            D_pmh      = wtspec.pmh;
            D_out_Qh_S = wtspec.out_Qh_L;
            D_out_PPEh_S= wtspec.out_PPEh_L;
        }


        ////////////////////////////////////////////in
        rCode1 = wtspec.pacDB.getKhta( wbOA );
        rCode2 = wtspec.pacDB.getKhti( dbIn );

        heatCapacity = rCode1 * rCode2 * D_out_Qh_S;

        if( heatCapacity < 0 ){
           heatCapacity = 0;
        }
        // System.out.println("heatCapacity ="+rCode1+" "+rCode2+" "+heatCapacity+"  ");
        wtspec.Rh = wtspec.pacDB.getKhhp( wtspec.PIDrate.getValue() );//負荷率をここで適用することに変更20130702
        if( wtspec.Rh > 1.0 ) {
           wtspec.Rh = 1.0;
//         wtspec.Th_Loadout = wtspec.Sh_Loadout;//20120731//室外機実処理熱量[W]
        }

    // IAin   = Psychrometrics.FNH( wtspec.DBin, wtspec.XGin );
    // IAout1 = IAin + heatCapacity * rh / wtspec.in_Va_S * 1000.0;//負荷率をここで適用することに変更20130702
    //  XGout1 = wtspec.XGin;
    // DBout1 = Psychrometrics.FNDbxh( XGout1, IAout1 );
       S_heat = heatCapacity  * wtspec.Rh;

       //       System.out.println("Rh="+Rh);

        wtspec.Sh_Loadout = heatCapacity;
        wtspec.Th_Loadout = heatCapacity;
        wtspec.allSh_heat = S_heat;
        wtspec.allTh_heat = S_heat;

        //低負荷領域の計算仕分け:Rh
        if( wtspec.Rh < wtspec.in_run_stop ){
           switch( wtspec.num_calcTypeLowerRangeLoad){
           case 0://  0_発停運転
              Rp = 0;
              break;

           case 1://  1_下限入力値固定
           case 2://  2_下限COP値固定
           case 3://  3_下限入力値と中間切片
              Rp = wtspec.in_run_stop;
              break;

           default:
              Rp = 0;
           }
        }else{
           Rp = wtspec.Rh;
        }

        alph = wtspec.pacDB.getAlphah( Rp );
        beta = wtspec.pacDB.getBetah( Rp, D_fmh, D_pmh);

        if( D_pmh * D_fmh < 0.001 || wtspec.iType == wtspec.iType_STANDARD ){//標準（定速）はbeta=1
           beta = 1.0;
        }
```

```
        rCode0 = alph * beta;
        rCode1 = wtspec.pacDB.getKhhpid( Rp );
        rCode2 = wtspec.pacDB.getKhwta( wbOA );

        rCode3 = wtspec.pacDB.getKhwti( dbIn );

        wtspec.PPE_out = rCode0 * rCode1 * rCode2 * rCode3 * D_out_PPEh_S;

        if( wtspec.iType == wtspec.iType_STANDARD ){
           wtspec.PPE_out /= ( 1.- 0.25 * ( 1. - Rp ));
        }

        //低負荷領域の計算仕分け:PPE
        if( wtspec.Rh < wtspec.in_run_stop && Rp > 0 ){
           switch( wtspec.num_calcTypeLowerRangeLoad){
           case 0://  0_発停運転
              wtspec.PPE_out = wtspec.out_TKEa_S;
              break;

           case 1://  1_下限入力値固定
              //何もしない
              break;

           case 2://  2_下限COP値固定
              wtspec.PPE_out *= ( wtspec.Rh / Rp );
              break;

           case 3://  3_下限入力値と中間切片
              wtspec.PPE_out *= ( 0.5 + wtspec.Rh / Rp / 2. );
              break;

           default:
           }
        }else{
           //何もしない
        }

        //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);

        //airOutEA
        wtspec.airOutEA.setFlowRate( wtspec.fRate_airEA );
        wtspec.airOutEA.setTempDB( ( -wtspec.Th_Loadout + wtspec.PPE_out ) / wtspec.fRate_airEA * 1000. +
wtspec.airInOA.getTempDB() );

        if( wtspec.Rh < 0.0001 ){
//         wtspec.CAPrateC = 0.0;
//         wtspec.CAPrateH = 0.0;
           wtspec.Tc_Loadout = 0.0;//室外機実処理熱量[W]
           wtspec.Sc_Loadout = 0.0;//室外機処理可能熱量[W]
           wtspec.Th_Loadout = 0.0;//室外機実処理熱量[W]
           wtspec.Sh_Loadout = 0.0;//室外機処理可能熱量[W]
           wtspec.PPE_out = wtspec.out_TKEa_S;
           wtspec.airOutEA.setFlowRate( 0. );
           wtspec.airOutEA.setTempDB( wtspec.airInOA.getTempDB() );

        }else if( wtspec.Rh < wtspec.in_run_stop ){
           switch( wtspec.num_calcTypeLowerRangeLoad){
           case 0://  0_発停運転
//            wtspec.CAPrateC = 0.0;
//            wtspec.CAPrateH = 0.0;
              wtspec.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
              wtspec.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
              wtspec.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
              wtspec.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
              wtspec.PPE_out = wtspec.out_TKEa_S;
              wtspec.airOutEA.setFlowRate( 0. );
              wtspec.airOutEA.setTempDB( wtspec.airInOA.getTempDB() );
              break;

           case 1://  1_下限入力値固定
              break;
```

```
            case 2://  2_下限COP値固定
               break;

            case 3://  3_下限入力値と中間切片
               break;

            default:
            }
         }

         if( wtspec.PIDrate.getValue() < wtspec.in_run_stop ){
            switch( wtspec.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
               wtspec.Tc_Loadout = 0.0;//室外機実処理熱量[W]
               wtspec.Sc_Loadout = 0.0;//室外機処理可能熱量[W]
               wtspec.Th_Loadout = 0.0;//室外機実処理熱量[W]
               wtspec.Sh_Loadout = 0.0;//室外機処理可能熱量[W]
               wtspec.allTc_heat = 0;
               wtspec.allSc_heat = 0;
               wtspec.allTh_heat = 0;
               wtspec.allSh_heat = 0;
               break;

            case 1://  1_下限入力値固定
               break;

            case 2://  2_下限COP値固定
               break;

            case 3://  3_下限入力値と中間切片
               break;

            default:
            }
         }

         // System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate+" DBout="+DBout1);

         wtspec.RHra = Psychrometrics.FNRhtx( wtspec.DBra, wtspec.XGra );
//       wtspec.Sh_Load_RM = wtspec.Sh_Load;
//       wtspec.Th_Load_RM = wtspec.Th_Load;

         if( wtspec.in_HUM_on <= wtspec.RHra || wtspec.in_HUM_mx < 0.00001 ){
            wtspec.IAin  = Psychrometrics.FNH( wtspec.DBin, wtspec.XGin );
            IAout = wtspec.IAin +wtspec.allTh_heat / wtspec.in_Va_S * 1000.0;
            wtspec.DBout = wtspec.DBin + wtspec.allSh_heat / wtspec.in_Va_S;//乾球温度
            wtspec.XGout = Psychrometrics.FNXth( wtspec.DBout, IAout );//絶対湿度
            wtspec.CW_Wrate = 0.0;
            //    System.out.println("1 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
            //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
         } else {
            wtspec.IAin  = Psychrometrics.FNH( wtspec.DBin, wtspec.XGin );
            IAout = wtspec.IAin + wtspec.allTh_heat / wtspec.in_Va_S*1000.0;
            wtspec.DBout = Psychrometrics.FNDbrh( wtspec.in_HUM_rt, IAout);
            wtspec.XGout = Psychrometrics.FNxtr( wtspec.DBout, wtspec.in_HUM_rt);
            wtspec.CW_Wrate = ( wtspec.XGout - wtspec.XGin ) * wtspec.GW;

            //定格加湿量をオーバーするときの処理
            if( wtspec.CW_Wrate > wtspec.in_HUM_mx ){
               wtspec.XGout = wtspec.in_HUM_mx / wtspec.GW+ wtspec.XGin;
               wtspec.DBout =Psychrometrics.FNDbxh( wtspec.XGout, IAout);
               wtspec.CW_Wrate = wtspec.in_HUM_mx;
            }
            if( wtspec.XGout < wtspec.XGin){
               wtspec.IAin  = Psychrometrics.FNH( wtspec.DBin, wtspec.XGin );
               IAout = wtspec.IAin + wtspec.allTh_heat / wtspec.in_Va_S * 1000.0;
               wtspec.DBout = wtspec.DBin + wtspec.allSh_heat / wtspec.in_Va_S;//乾球温度
               wtspec.XGout = Psychrometrics.FNXth( wtspec.DBout, IAout);//絶対湿度
               wtspec.CW_Wrate = 0.0;
            }
            //    System.out.println("2 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
```

```
        //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
      }
      //    wtspec.watInCW.setFlowRate(CW_Wrate);
   }


   private static double WB_cal(double ddb,double xx){
      double wwb,x1,de=1;
      int kk=-1,j=0;
      wwb=ddb;

      do{
         j++;
         x1=Psychrometrics.FNXtw(ddb,wwb);
         if(Math.abs(x1-xx)<0.000003) {
            //          System.out.println(" j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
            return wwb;
         }
         if(x1<xx) {
            if(kk==-1) de=de/2.0;
            wwb=wwb+de;
            kk=1;
         }
            if(x1>=xx) {
            if(kk==1) de=de/2.0;
            wwb=wwb-de;
            kk=-1;
         }

      }while(j<1000);
      //    System.out.println("*j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
      return wwb;

   }




   private static void ventilation_cal( WallThroughSpec2015 wtspec ){
      //double heatCapacity=0.0,rCode1 = 0.0,rCode2 = 0.0,rCode3 = 0.0,rCode4 = 0.0,rCode5 = 0.0,rCode6 = 0.0,rCode7 =
0.0,rCode8 = 0.0;
      //double IAra,IAout1,DBout1,XGout1,S_heat,T_heat;
      double HEXrate;

      HEXrate = wtspec.in_EX_S;

      //換気の時は全熱交換器は停止とする
      HEXrate = 0.0;

      //外気と室空気の混合空気の計算
      if( Airswc.isON( wtspec.swcInOA )){//20101212nino
         wtspec.DBin = (( wtspec.in_Va_S - wtspec.in_OA_S ) * wtspec.DBra + (( wtspec.DBoa - wtspec.DBra ) * ( 1.0 -
HEXrate ) + wtspec.DBra ) * wtspec.in_OA_S ) / wtspec.in_Va_S;
         wtspec.XGin = (( wtspec.in_Va_S - wtspec.in_OA_S ) * wtspec.XGra + (( wtspec.XGoa - wtspec.XGra ) * ( 1.0 -
HEXrate ) + wtspec.XGra ) * wtspec.in_OA_S ) / wtspec.in_Va_S;
         wtspec.CO2ppmin = (( wtspec.in_Va_S - wtspec.in_OA_S ) * wtspec.CO2ppmra + wtspec.in_OA_S * wtspec.CO2ppmoa ) /
wtspec.in_Va_S;//20131031
      }else{//20101212nino
         wtspec.DBin = wtspec.DBra;//20130109
         wtspec.XGin = wtspec.XGra;//20130109
         wtspec.CO2ppmin = wtspec.CO2ppmra;//20131031
      }
      wtspec.WBin = Psychrometrics.FNWbtx( wtspec.DBin, wtspec.XGin );
      //    System.out.println("C/DBin="+DBin+" "+XGin+"  ");
      //    System.out.println("C/DBoa="+DBoa+" "+XGoa+"  ");
      //    System.out.println("C/DBra="+DBra+" "+XGra+"  ");
      //    System.out.println("  ");

      //    System.out.println("DBra="+DBra+" "+XGra+"  ");

//    wtspec.LPrate = 1.0;
```

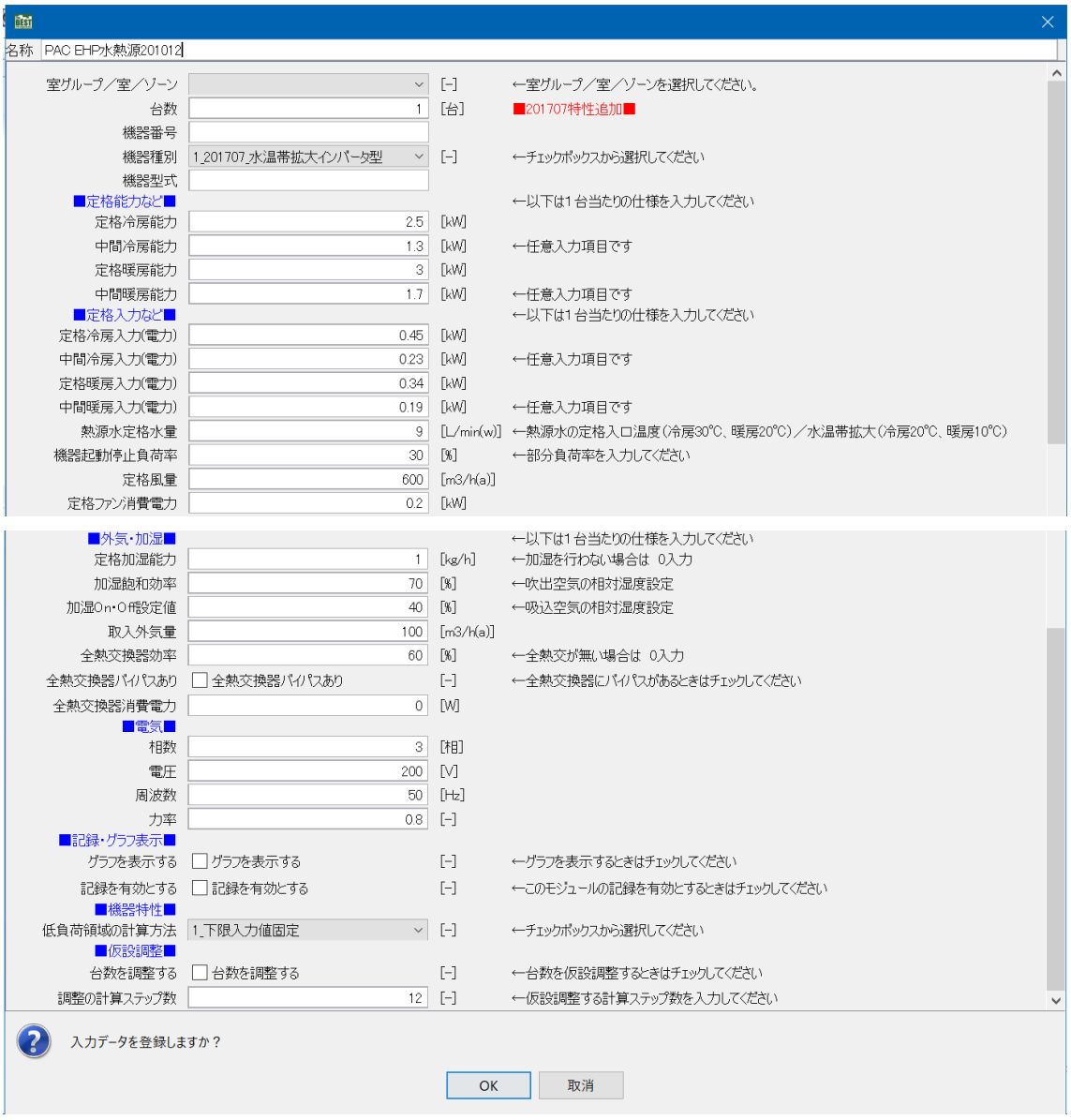
```
        wtspec.T_C_heat = 0.0;

//      wtspec.S_Load = 0.0;
//      wtspec.T_Load = 0.0;
//      wtspec.Sc_Load = 0.0;
//      wtspec.Tc_Load = 0.0;
//      wtspec.Sh_Load = 0.0;
//      wtspec.Th_Load = 0.0;

        // System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate.getValue());
    }


    private static void ventilation_supply( WallThroughSpec2015 wtspec ){

//      wtspec.Sh_Load_RM = 0.0;
//      wtspec.Th_Load_RM = 0.0;

        wtspec.DBout = wtspec.DBin;//乾球温度
        wtspec.XGout = wtspec.XGin;//絶対湿度
        wtspec.CO2ppmout = wtspec.CO2ppmin;//20131031
        wtspec.CW_Wrate = 0.0;

    }

}
```

「PAC EHP 水熱源 201012」（場所：設備 2015／個別分散 2015／）

モジュール名	PAC EHP 水熱源 201012
クラス	WaterCooledEHPModule20101212

（1）入力画面

・スペック



（2）モジュールの概要

水熱源タイプの PAC モジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	－	[-]	－	－		
1	室グループ／室／ゾーン	String	zoneAirforSSinside	#RoomGroup	[-]	－	－		←室グループ／室／ゾーンを選択してください。
2	台数	double	in_dai	1	[台]	－	0		
3	機器番号	String	m_no			－	－		
4	機器種別	String	m_ty	1_201707_水温帯拡大インバータ型、0_201303_定速型、1_201303_インバータ型、0_定速型、1_インバータ型	[-]	－	－		←チェックボックスから選択してください
5	機器型式	String	m_kt			－	－		
6	■定格能力など■					－	－		←以下は１台当たりの仕様を入力してください
7	定格冷房能力	double	out_Qc_S	2.5	[kW]	－	0		
8	中間冷房能力	double	out_Qc_C	1.3	[kW]	－	0		←任意入力項目です
9	定格暖房能力	double	out_Qh_S	3	[kW]	－	0		
10	中間暖房能力	double	out_Qh_C	1.7	[kW]	－	0		←任意入力項目です
11	■定格入力など■					－	－		←以下は１台当たりの仕様を入力してください
12	定格冷房入力(電力)	double	out_PPEc_S	0.45	[kW]	－	0		

13	中間冷房入力(電力)	double	out_PPEc_C	0.23	[kW]	ー	0		←任意入力項目です
14	定格暖房入力(電力)	double	out_PPEh_S	0.34	[kW]	ー	0		
15	中間暖房入力(電力)	double	out_PPEh_C	0.19	[kW]	ー	0		←任意入力項目です
16	熱源水定格水量	double	de_fRate_watInHS	9	[L/min(w)]	ー	0		←熱源水の定格入口温度（冷房 30℃、暖房 20℃）／水温帯拡大（冷房 20℃、暖房 10℃）
17	機器起動停止負荷率	double	in_run_stop	30	[%]	100	0		←部分負荷率を入力してください
18	定格風量	double	in_Va_S	600	[m3/h(a)]	ー	0		
19	定格ファン消費電力	double	in_PPEa_S	0.2	[kW]	ー	0		
20	■外気・加湿■					ー	ー		←以下は１台当たりの仕様を入力してください
21	定格加湿能力	double	in_HUM_mx	1	[kg/h]	ー	0		←加湿を行わない場合は　0 入力
22	加湿飽和効率	double	in_HUM_rt	70	[%]	ー	0		←吹出空気の相対湿度設定
23	加湿 On・Off 設定値	double	in_HUM_on	40	[%]	ー	0		←吸込空気の相対湿度設定
24	取入外気量	double	in_OA_S	100	[m3/h(a)]	ー	0		
25	全熱交換器効率	double	in_EX_S	60	[%]	100	0		←全熱交が無い場合は　0 入力
26	全熱交換器バイパスあり	boolean	isHexbypass	FALSE	[-]	ー	ー		←全熱交換器にバイパスがあるときはチェックしてください
27	全熱交換器消費電力	double	in_EX_PE	0	[W]	ー	0		
28	■電気■					ー	ー		
29	相数	int	phase	3	[相]	ー	1		
30	電圧	double	voltage	200	[V]	ー	100		
31	周波数	double	frequency	50	[Hz]	60	50		
32	力率	double	powerFactor	0.8	[-]	1	0		

33	■記録・グラフ表示■					―	―		
34	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
35	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください
36	■機器特性■					―	―		
37	低負荷領域の計算方法	String	calcTypeLowerRangeLoad	0_発停運転、1_下限入力値固定、2_下限COP値固定、3_下限入力値と中間切片	[-]	―	―		←チェックボックスから選択してください
38	■仮設調整■					―	―		
39	台数を調整する	boolean	isAdjust2012	FALSE	[-]	―	―		←台数を仮設調整するときはチェックしてください
40	調整の計算ステップ数	int	numAdjustSteps	12	[-]	―	1		←仮設調整する計算ステップ数を入力してください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	外気導入信号入口	L1_swcInOA	swcInOA	-	制御	OnOff 信号	入口	
4	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
5	PID モード信号入口	L1_modInPID	modInPID	-	制御	制御モード	入口	
6	還気入口	L0_airInRA	airInRA	-	状態	空気	入口	
7	給気出口	L0_airOutSA	airOutSA	-	状態	空気	出口	
8	外気入口	L0_airInOA	airInOA	-	状態	空気	入口	
9	排気出口	L0_airOutEA	airOutEA	-	状態	空気	出口	
10	熱源水入口	L0_watInHS	watInHS	-	状態	水	入口	
11	熱源水出口	L0_watOutHS	watOutHS	-	状態	水	出口	
12	給水入口	L0_watInCW	watInCW	-	状態	水	入口	
13	ドレン出口	L0_watOutD	watOutD	-	状態	水	出口	
14	操作量入口	L0_valInCtrl	valInCtrl	-	値	値	入口	
15	出力入口	L0_eleIn	eleIn	-	状態	電気	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	_Message#-#-	メッセージ		メッセージ
2	処理顕熱量合計#W#熱量	処理顕熱量合計	W	My
3	冷却要求顕熱量合計#W#熱量	冷却要求顕熱量合計	W	My
4	加熱要求顕熱量合計#W#熱量	加熱要求顕熱量合計	W	My
5	処理全熱量合計#W#熱量	処理全熱量合計	W	My
6	冷却要求全熱量合計#W#熱量	冷却要求全熱量合計	W	My
7	加熱要求全熱量合計#W#熱量	加熱要求全熱量合計	W	My
8	冷却能力補正比率	冷却能力補正比率	—	My
9	加熱能力補正比率	加熱能力補正比率	—	My
10	消費電力#W#消費電力	消費電力#W#消費電力	W	エネルギー消費
11	冷却可能熱量/最大#W#熱量	冷却可能熱量/最大	W	My
12	加熱可能熱量/最大#W#熱量	加熱可能熱量/最大	W	My
13	室外機_処理全熱量合計負荷#W#-	室外機_処理全熱量合計負荷	W	負荷
14	室内機_PID 操作量#-#-	室内機_PID 操作量	—	入口
15	熱源水入口流量#g/s#質量流量	熱源水入口流量	g/s	入口
16	熱源水入口温度#℃#温度	熱源水入口温度	℃	入口
17	熱源水出口流量#g/s#質量流量	熱源水出口流量	g/s	出口
18	熱源水出口温度#℃#温度	熱源水出口温度	℃	出口
19	COP#-#COP	COP	—	
20	室吹出乾球温度#℃#温度	室吹出乾球温度	℃	出口
21	室吹出絶対湿度#g/g#湿度	室吹出絶対湿度	g/s	出口
22	室吹出風量#g/s#質量流量	室吹出風量	g/s	出口
23	ドレン量#g/s#質量流量	ドレン量	g/s	出口
24	加湿給水量#g/s#質量流量	加湿給水量	g/s	入口
25	室吸込乾球温度#℃#温度	室吸込乾球温度	℃	入口
26	室吸込絶対湿度#g/g#湿度	室吸込絶対湿度	g/s	入口
27	室外機調整冷却能力#W#熱量	室外機調整冷却能力	W	調整
28	室外機調整加熱能力#W#熱量	室外機調整加熱能力	W	調整
29	室内機調整冷却台数#-#台数	室内機調整冷却台数	—	調整
30	室内機調整加熱台数#-#台数	室内機調整加熱台数	—	調整
31	室内機調整ステップ平均冷却台数#-#台数	室内機調整ステップ平均冷却台数	—	調整

32	室内機調整ステップ平均加熱台数#-#台数	室内機調整ステップ平均加熱台数	—	調整
33				

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.essentials.utility.Psychrometrics;
import jp.or.ibec.best.log.BestLogger;
import jp.or.ibec.best.message.BestEngineMessageHandler;
import jp.or.ibec.best.message.BestEngineMessageParam;
import jp.or.ibec.best.message.constants.SystemMessageConstants;

/**
 * @author HIROSHI NINOMIYA/20101212
 *
 * 水冷EHP
 *
 * 201303機器特性追加
 *
 */
public class WaterCooledEHPModule20101212 extends AbstractBestModule implements
IBestMetaModule  {

   /**
    * 論理アドレス
    */
   private final String moduleName="(WaterCooledEHPModule20101212)";

   private final String S_NODE_airInOA  = "L0_airInOA";//外気
   private final String S_NODE_airOutEA = "L0_airOutEA";//外気
   private final String S_NODE_watInHS  = "L0_watInHS";//熱源水入口
   private final String S_NODE_watOutHS = "L0_watOutHS";//熱源水出口

   private final String S_NODE_airInRA  = "L0_airInRA";//室内吸込口
   private final String S_NODE_airOutSA = "L0_airOutSA";//室内吹出し

   private final String S_NODE_watOutD  = "L0_watOutD";//コイル出口ドレン
   private final String S_NODE_watInCW  = "L0_watInCW";//watInCW---090407追加**************

   private final String S_NODE_valInPID = "L0_valInCtrl";//供給熱量 from PID           : in

   private final String S_NODE_eleIn    = "L0_eleIn";//電力

   private final String C_NODE_swcIn = "L1_swcIn";//運転状態：off/on
   private final String C_NODE_swcInOA = "L1_swcInOA";//外気運転状態：off/on
   private final String C_NODE_modIn = "L1_modIn";//空調モード：停止/冷却/加熱
   private final String C_NODE_modInPID = "L1_modInPID";//空調モード：停止/冷却/加熱

   private final String R_NODE = "L2_recOut";



   /**
    * 外部定義
    */
```

```
    private final String SPEC_name          = "名称";

    private final String SPEC_grzName  = "室グループ／室／ゾーン";

    private final String SPEC_m_no          = "機器番号";
    private final String SPEC_m_ty          = "機器種別";
    private final String SPEC_m_kt          = "機器型式";
    private final String SPEC_out_Qc_S      = "定格冷房能力[W]";//kW
    private final String SPEC_out_Qc_C      = "中間冷房能力[W]";//kW
    private final String SPEC_out_Qh_S      = "定格暖房能力[W]";//kW
    private final String SPEC_out_Qh_C      = "中間暖房能力[W]";//kW
//  private final String SPEC_out_Qh_L      = "低温暖房能力[W]";//kW
    private final String SPEC_out_PPEc_S    = "定格冷房入力(電力)[W]";//kW
    private final String SPEC_out_PPEc_C    = "中間冷房入力(電力)[W]";//kW
    private final String SPEC_out_PPEh_S    = "定格暖房入力(電力)[W]";//kW
    private final String SPEC_out_PPEh_C    = "中間暖房入力(電力)[W]";//kW
//  private final String SPEC_out_PPEh_L    = "低温暖房入力(電力)[W]";//kW
/*
    private final String SPEC_out_CLEa_R    = "クランクケースヒータ(運転時)";//W
    private final String SPEC_out_CLEa_S    = "クランクケースヒータ(停止時)";//W
    private final String SPEC_out_TKEa_R    = "定格待機電力(運転時)";//W
    private final String SPEC_out_TKEa_T    = "定格待機電力(待機時)";//W
    private final String SPEC_out_TKEa_S    = "定格待機電力(停止時)";//W
*/
    private final String SPEC_de_fRate_watInHS   = "熱源水定格水量[g/s]";

    private final String SPEC_in_run_stop  = "機器起動停止負荷率[-]";//[%]
    private final String SPEC_Phase       = "相数[-]";
    private final String SPEC_Voltage     = "電圧[V]";
    private final String SPEC_Frequency   = "周波数[Hz]";
    private final String SPEC_PowerFactor = "力率[-]";
    //
    private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
    private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

    private final String SPEC_CalcTypeLowerRangeLoad = "低負荷領域の計算方法";

    private final String SPEC_isAdjust2012   = "台数を調整する";
    private final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";

    private StringBuffer message= new StringBuffer();

    /**
     * 外部定義
     */

    private final String SPEC_in_dai       = "台数[-]";
//  private final String SPEC_in_Qc_S      = "定格冷房能力";//kW
//  private final String SPEC_in_Qh_S         = "定格暖房能力";//kW
    private final String SPEC_in_Va_S         = "定格風量[g/s]";//m3/h
    private final String SPEC_in_PPEa_S    = "定格ファン消費電力[W]";//W
//  private final String SPEC_in_TKEa_R     = "定格待機電力(運転時)";//W
//  private final String SPEC_in_THEa_S     = "定格待機電力(停止時)";//W
//  private final String SPEC_in_run_stop     = "機器起動停止負荷率";//[%]
//  private final String SPEC_in_Lp        = "冷媒管長";//m
//  private final String SPEC_in_Lh        = "冷媒管高低差";//m
    private final String SPEC_in_HUM_mx       = "定格加湿能力[g/s]";//kg/h
    private final String SPEC_in_HUM_rt       = "加湿飽和効率[-]";//[%]
    private final String SPEC_in_HUM_on       = "加湿On・Off設定値[-]";//[%]
    private final String SPEC_in_OA_S         = "取入外気量[g/s]";//m3/h
    private final String SPEC_in_EX_S         = "全熱交換器効率[-]";//[%]
    private final String SPEC_isHexbypass      = "全熱交換器バイパスあり[-]";
    private final String SPEC_in_EX_PE        = "全熱交換器消費電力[W]";


    /**
     * 変数（外部定義に対応）
     */

    private String name             ;// "機器名称";
    private String m_no             ;//= "機器番号";
    private String m_ty             ;//= "機器種別";
```

```
    private String m_kt              ;//= "機器型式";
    private double out_Qc_S      ;//= "定格冷房能力";//kW
    private double out_Qc_C      ;//= "中間冷房能力";//kW
    private double out_PPEc_S    ;//= "定格冷房入力(電力)";//kW
    private double out_PPEc_C    ;//= "中間冷房入力(電力)";//kW
    private double out_Qh_S      ;//= "定格暖房能力";//kW
    private double out_Qh_C      ;//= "中間暖房能力";//kW
    private double out_Qh_L      ;//= "低温暖房能力";//kW
    private double out_PPEh_S    ;//= "定格暖房入力(電力)";//kW
    private double out_PPEh_C    ;//= "中間暖房入力(電力)";//kW
    private double out_PPEh_L    ;//= "低温暖房入力(電力)";//kW
/*
    private double out_CLEa_R=0.0    ;//= "クランクケースヒータ(運転時)";//W
    private double out_CLEa_S=0.0    ;//= "クランクケースヒータ(停止時)";//W
*/
    private double out_TKEa_R =0.0  ;//= "定格待機電力(運転時)";//W
    private double out_TKEa_T =0.0  ;//= "定格待機電力(待機時)";//W
    private double out_TKEa_S =0.0  ;//= "定格待機電力(停止時)";//W

    private double t_watInHS;// = this.watInHS.getTemp();
// private double t_watOutHS;// = this.watOutHS.getTemp();
    private double fRate_watInHS ;//熱源水水量[g/s]
    private double de_fRate_watInHS ;//熱源水定格水量[g/s]

    private double in_run_stop    ;//= "機器起動停止負荷率";//[%]
    private int phase;         //相数[-]
    private double voltage;    //電圧[V]
    private double frequency;  //周波数[Hz]
    private double powerFactor;   //力率[-]

    private boolean isGVisible = false;//このグラフを表示する=true
    private boolean isRecord   = false;//記録を有効とする=true


    /**
     * 変数（外部定義に対応）
     */

// private String name           ;//= "機器名称";
// private String m_no           ;//= "機器番号";
// private String m_kt           ;//= "機器型式";
    private double in_dai         ;//= "台数";
    private double in_Qc_S        ;//= "定格冷房能力";//kW
    private double in_Qh_S        ;//= "定格暖房能力";//kW
    private double in_Va_S        ;//= "定格風量";//m3/h
    private double in_PPEa_S      ;//= "定格ファン消費電力";//W
    private double in_TKEa_R=0.0  ;//= "定格待機電力(運転時)";//W
    private double in_TKEa_S=0.0  ;//= "定格待機電力(停止時)";//W
// private double in_run_stop    ;//= "機器起動停止負荷率";//[%]
// private double in_Lp          ;//= "冷媒管長";//m
// private double in_Lh          ;//= "冷媒管高低差";//m
    private double in_HUM_mx         ;//= "定格加湿能力";//kg/h
    private double in_HUM_rt         ;//= "加湿飽和効率";//[%]
    private double in_HUM_on         ;//= "加湿On・Off設定値";//[%]
    private double in_OA_S        ;//= "取入外気量";//m3/h
    private double in_EX_S        ;//= "全熱交換器効率";//[%]
    private boolean isHexbypass = false;//熱交バイパスあり=true
    private double in_EX_PE    ;//= "全熱交換器消費電力[W]";
// private int phase        ;//= "相数";
// private double voltage      ;//= "電圧";
// private double frequency   ;// = "周波数";
// private double powerFactor  ;//= "力率";

// private boolean isGVisible = false;//このグラフを表示する=true
// private boolean isRecord   = false;//記録を有効とする=true

// private StringBuffer message= new StringBuffer();

    private BestAir airInOA  = null;
    private BestAir airOutEA   = null;

    private BestWater watInHS = null;
```

```
    private BestWater watOutHS = null;

    private BestElectricity eleIn = null;  //電力
// private Double CAPrate = null;
    private double CAPrateC = 0;//null;
    private double CAPrateH = 0;//null;
//  private BM_EHPdata RMdata = null;  //電力
    private PACDBManager pacDB=null;

    private double rangeMaxCOOLING_T_watIn;//熱源水入口温度上限
    private double rangeMinCOOLING_T_watIn;//熱源水入口温度下限
    private double rangeMaxCOOLING_wbIn;//室WB上限
    private double rangeMinCOOLING_wbIn;//室WB下限

    private double rangeMaxHEATING_T_watIn;//熱源水入口温度上限
    private double rangeMinHEATING_T_watIn;//熱源水入口温度下限
    private double rangeMaxHEATING_dbRA;//室DB上限
    private double rangeMinHEATING_dbRA;//室DB下限



    private int modIn  ;
    private int modInPID  ;
    private int swcIn ;
    private int swcInOA ;
    private int mState;  //停止、冷房、暖房フラグ



    private BestAir airInRA   = null;
    private BestAir airOutSA  = null;         //給気
    private BestWater watOutD = null;
    private BestWater watInCW = null;
// private BestElectricity eleIn = null;  //電力
// private BM_EHPdata RMdata = new BM_EHPdata();   //電力
    private BestValue PIDrate = null;
// private Double CAPrate = null;
// private Double LPrate = null;

// private PACDBManager pacDB=null;

// private int mode  ;
// private int onOff ;
// private int mState;  //停止、冷房、暖房フラグ

    /**
     * 出力（建物）
     */
    private final String RECORD_message = "_Message#-#-";
    private final String RECORD_ID1  = "処理顕熱量合計#W#熱量";
    private final String RECORD_ID1c = "冷却要求顕熱量合計#W#熱量";
    private final String RECORD_ID1h = "加熱要求顕熱量合計#W#熱量";
    private final String RECORD_ID2  = "処理全熱量合計#W#熱量";
    private final String RECORD_ID2c = "冷却要求全熱量合計#W#熱量";
    private final String RECORD_ID2h = "加熱要求全熱量合計#W#熱量";
    private final String RECORD_ID3c = "冷却能力補正比率";
    private final String RECORD_ID3h = "加熱能力補正比率";
    private final String RECORD_ID4  = "消費電力#W#消費電力";
    private final String RECORD_ID5c = "冷却可能熱量/最大#W#熱量";
    private final String RECORD_ID5h = "加熱可能熱量/最大#W#熱量";
// private final String RECORD_ID6c = "室外機冷却実処理熱量/現状#W#熱量";
// private final String RECORD_ID6h = "室外機加熱実処理熱量/現状#W#熱量";
    private final String RECORD_qT   = "室外機_処理全熱量合計負荷#W#-";
    private final String RECORD_pidVal  = "室内機_PID操作量#-#-";

    private final String RECORD_M_watInHS = "熱源水入口流量#g/s#質量流量";
    private final String RECORD_T_watInHS = "熱源水入口温度#℃#温度";
    private final String RECORD_M_watOutHS= "熱源水出口流量#g/s#質量流量";
    private final String RECORD_T_watOutHS= "熱源水出口温度#℃#温度";

    private final String RECORD_COP = "COP#-#COP";
```

```
    /**
     * 出力（建物）
     */

// private final String RECORD_ID1 = "室内機処理顕熱量#W#熱量";
// private final String RECORD_ID2 = "室内機処理全熱量#W#熱量";
// private final String RECORD_ID3 = "室内機消費電力#W#消費電力";
   private final String RECORD_ID4i = "室吹出乾球温度#℃#温度";
   private final String RECORD_ID5i = "室吹出絶対湿度#g/g#湿度";
   private final String RECORD_ID6i = "室吹出風量#g/s#質量流量";
   private final String RECORD_ID7i = "ドレン量#g/s#質量流量";
   private final String RECORD_ID8i = "加湿給水量#g/s#質量流量";

   private final String RECORD_DBairInRM = "室吸込乾球温度#℃#温度";
   private final String RECORD_XairInRM  = "室吸込絶対湿度#g/g#湿度";



   private final String RECORD_out_Qc_Adjust = "室外機調整冷却能力#W#熱量";
   private final String RECORD_out_Qh_Adjust = "室外機調整加熱能力#W#熱量";
   private final String RECORD_in_daicAdjust = "室内機調整冷却台数#-#台数";
   private final String RECORD_in_daihAdjust = "室内機調整加熱台数#-#台数";
   private final String RECORD_avecdaiAdjust = "室内機調整ステップ平均冷却台数#-#台数";
   private final String RECORD_avehdaiAdjust = "室内機調整ステップ平均加熱台数#-#台数";

   private double DBin;          //室内乾球温度[℃]
   private double WBin;          //室内湿球温度[℃]
   private double XGin;
   private double IAin;
   private double RHin;

   private double DBoa;          //外気乾球温度[℃]
   private double WBoa;          //外気湿球温度[℃]
   private double XGoa;
   private double IAoa;

   private double DBra;          //室内機吸込乾球温度[℃]
   private double WBra;          //室内機吸込湿球温度[℃]
   private double XGra;
   private double IAra;

   private double DBout;            //乾球温度[℃]
   private double XGout;            //絶対湿度[kg/kg.DA]

   private double GW;               //風量(g/s)
   private double D_Wrate;       //ドレン水量(g/s)
   private double CW_Wrate;         //加湿水量(g/s)

   private double PPE_out;
   private double E_PPE_out;
   private double PPE_in;
   private double E_PPE_in;
   private double S_Load;//室要求顕熱負荷
   private double T_Load;//室要求全熱負荷
   private double Sc_Load;//室要求冷房顕熱負荷
   private double Tc_Load;//室要求冷房全熱負荷
   private double Sh_Load;//室要求暖房顕熱負荷
   private double Th_Load;//室要求暖房全熱負荷
   private double S_Load_RM;//処理熱量（顕熱）[W]
   private double T_Load_RM;//処理熱量（全熱）[W]
   private double Sc_Load_RM;//処理熱量（顕熱）[W]
   private double Tc_Load_RM;//処理熱量（全熱）[W]
   private double Sh_Load_RM;//処理熱量（顕熱）[W]
   private double Th_Load_RM;//処理熱量（全熱）[W]
// private double S_Load_RM;
// private double T_Load_RM;
//******************090725-s
   private double saijohatsu;
   private double T_C_heat;
// private int u40=0;
```

```
//*****************090725-e
   private int dbflg=0;
   private String path="EHP";
   private String[] filenames= new String[1];
   private String equipmentName;
   //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   private int HEXselect=1;//1=エンタルピ基準  2=温度基準
   //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

   //グラフ表示など
   private GraphJFrameBuilMultiIn_S20101212 gBMIn = null;


   //低負荷領域の計算方法
   private String calcTypeLowerRangeLoad = null;//"低負荷領域の計算方法";
   private int num_calcTypeLowerRangeLoad;//"低負荷領域の計算方法";

   /**
    * その他変数
    */

   private double Rc;               //部分負荷率
   private double Rh;               //部分負荷率

   // private double DBoa;          //外気乾球温度[℃]
// private double WBoa;          //外気湿球温度[℃]
// private double XGoa;          //外気絶対湿度[g/gD.A.]

// private double DBra;          //室内機吸込乾球温度[℃]
// private double WBra;          //室内機吸込湿球温度[℃]
// private double XGra;          //室内機吸込絶対湿度[g/gD.A.]

   private double Sc_Loadout;//室外機処理可能熱量（冷却全熱）[W]
   private double Sh_Loadout;//室外機処理可能熱量（加熱全熱）[W]
   private double Tc_Loadout;//室外機実処理熱量（冷却全熱）[W]
   private double Th_Loadout;//室外機実処理熱量（加熱全熱）[W]
   private double allSc_heat;//室内機処理熱量（冷却顕熱）[W]
   private double allSh_heat;//室内機処理熱量（加熱顕熱）[W]
   private double allTc_heat;//室内機処理熱量（冷却全熱）[W]
   private double allTh_heat;//室内機処理熱量（加熱全熱）[W]
// private double S_Loadout;//室外機処理可能熱量[W]
// private double T_Loadout;//室外機実処理熱量[W]
// private double allSheat;//室内機処理熱量（顕熱）[W]
// private double allTheat;//室内機処理熱量（全熱）[W]
// private double PPE;
// private double E_PPE;

// private int num;
   private String kiki_file;
// private String path="EHP";
// private String[] filenames= new String[1];
// private String equipmentName;

// private double DBra_max;
// private double WBra_min;
// private double Lratemin;
   private double fmc;
   private double pmc;
   private double fmh;
   private double pmh;

   private double COP;  //成績係数(-)
// private double QEx;  //放熱(W)

   //グラフ表示など
   private GraphJFrameBuilMultiOut_S20101212 gBMout = null;


   //仮設調整モード2012

   private boolean isAdjust2012  = false;//true="台数を調整する";//
   //private boolean isAdjust2012 = false;//20120406nino 建築学会大会空衛学会OS論文検討用＝true
```

```
    private int numAdjustSteps;//"調整の計算ステップ数";
    private LinkedList<Double> daiListc = null;//必要台数
    private LinkedList<Double> daiListh = null;//必要台数
    private LinkedList<Double> rcList = null;//必要台数
    private LinkedList<Double> rhList = null;//必要台数
    private double maxcAdjustdai = 1;
    private double maxhAdjustdai = 1;
    private double avecdaiAdjust = 0;
    private double avehdaiAdjust = 0;
    private double in_daicAdjust = 1;//= "調整台数";
    private double in_daihAdjust = 1;//= "調整台数";
    private double in_run_stop_Num;//= "機器起動停止負荷率"台数補正;

// private double max_rcAdjust = 1.;
// private double max_rhAdjust = 1.;
// private double ave_rcAdjust = 0;
// private double ave_rhAdjust = 0;
// private double out_rcAdjust = 1.;//= "調整率冷房";
// private double out_rhAdjust = 1.;//= "調整率暖房";

    private double out_Qc_S_1      ;//= "定格冷房能力";//kW
    private double out_Qc_C_1      ;//= "中間冷房能力";//kW
    private double out_PPEc_S_1    ;//= "定格冷房入力(電力)";//kW
    private double out_PPEc_C_1    ;//= "中間冷房入力(電力)";//kW
    private double out_Qh_S_1      ;//= "定格暖房能力";//kW
    private double out_Qh_C_1      ;//= "中間暖房能力";//kW
    private double out_Qh_L_1      ;//= "低温暖房能力";//kW
    private double out_PPEh_S_1    ;//= "定格暖房入力(電力)";//kW
    private double out_PPEh_C_1    ;//= "中間暖房入力(電力)";//kW
    private double out_PPEh_L_1    ;//= "低温暖房入力(電力)";//kW

// private double out_Qc_Adjust;//= "調整冷却能力";
// private double out_Qh_Adjust;//= "調整加熱能力";

    private double fRate_watInHSc ;//水量[g/s]
    private double fRate_watInHSh ;//水量[g/s]

    private double de_fRate_watInHS_1 ;//熱源水定格水量[g/s]

    private double in_Va_S_1      ;//= "定格風量";//m3/h
    private double in_PPEa_S_1    ;//= "定格ファン消費電力";//W

    private double in_HUM_mx_1    ;//= "定格加湿能力";//kg/h

    private ZoneAirforSSinsideModule201502 zoneAirforSSinside = null;
    private String grzName = null;
    private boolean isUseZoneAir = false;


    @Override
    public void setProfile(BestSpecs spec) {
//    外部定義項目取得
      if(spec == null) {
         return;
      }
      Map<String,String> map=spec.getSpec();
      if(map == null) {
         return;
      }

      // 機器名称
      this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

      this.grzName = spec.getSpecValue( this.SPEC_grzName, "" );
      if( !this.grzName.equals( "" )){
         this.isUseZoneAir = true;
      }

      if( this.isUseZoneAir ){
         this.zoneAirforSSinside = new ZoneAirforSSinsideModule201502();
         this.zoneAirforSSinside._setProfile( this.name, this.grzName );
      }
```

```
    // 機器番号
// this.m_no = spec.getSpecValue( this.SPEC_m_no, "" );
    if(null!=map.get(this.SPEC_m_no)){
       this.m_no = (String)map.get(this.SPEC_m_no);
       //******************090725-s
       try{
          if(m_no.substring(0 , 1).equals("+")==true){
             saijohatsu=Double.parseDouble(m_no.substring(1 , 3));
          }else{
             saijohatsu=50.0;
          }
       }
       catch(NumberFormatException e){
          saijohatsu=50.0;
       }
       //******************090725-s
    } else {
       ///System.out.println("(W)SPEC_機器番号がありません");
       saijohatsu=50.0;  //******************090725
    }

    //  機器種別  0_定速型、1_インバータ型
    //this.m_ty = spec.getSpecValue( this.SPEC_m_ty, "0_201303_定速型" );
    this.m_ty = spec.getSpecValue( this.SPEC_m_ty, "1_201707_水温帯拡大インバータ型" );//201707

    //kiki_file="EHP_WaterCooled20100625";
    //this.kiki_file="EHP_WaterCooled201303";
    this.kiki_file="EHP_WaterCooledInv201707";//201707
    //
    if(m_ty.equals("1_201707_水温帯拡大インバータ型")){
       this.kiki_file="EHP_WaterCooledInv201707";
       this.out_TKEa_R =0.0  ;//= "定格待機電力(運転時)";//W
       this.out_TKEa_T =11.0  ;//= "定格待機電力(待機時)";//W
       this.out_TKEa_S =11.0  ;//= "定格待機電力(停止時)";//W
    }
    //
    if(m_ty.equals("0_201303_定速型")){
       this.kiki_file="EHP_WaterCooled201303";
       this.out_TKEa_R =0.0  ;//= "定格待機電力(運転時)";//W
       this.out_TKEa_T =5.0  ;//= "定格待機電力(待機時)";//W
       this.out_TKEa_S =5.0  ;//= "定格待機電力(停止時)";//W
    }
    if(m_ty.equals("1_201303_インバータ型")){
       this.kiki_file="EHP_WaterCooledInv201303";
       this.out_TKEa_R =0.0  ;//= "定格待機電力(運転時)";//W
       this.out_TKEa_T =11.0  ;//= "定格待機電力(待機時)";//W
       this.out_TKEa_S =11.0  ;//= "定格待機電力(停止時)";//W
    }

    //
    if(m_ty.equals("0_定速型")){
       this.kiki_file="EHP_WaterCooled20100625";
    }
    if(m_ty.equals("1_インバータ型")){
       this.kiki_file="EHP_WaterCooledInv20100625";
    }


    // 機器型式
    this.m_kt = spec.getSpecValue( this.SPEC_m_kt, "" );

    // 台数
    this.in_dai = spec.getSpecValue( this.SPEC_in_dai, 1. );

    // 定格冷房能力
    this.out_Qc_S   = spec.getSpecValue( this.SPEC_out_Qc_S, 0. ) * in_dai;
    this.out_Qc_S_1 = spec.getSpecValue( this.SPEC_out_Qc_S, 0. );

    // 中間冷房能力
    this.out_Qc_C   = spec.getSpecValue( this.SPEC_out_Qc_C, 0. ) * in_dai;
    this.out_Qc_C_1 = spec.getSpecValue( this.SPEC_out_Qc_C, 0. );
```

```
        // 定格冷房入力(電力)
        this.out_PPEc_S   = spec.getSpecValue( this.SPEC_out_PPEc_S, 0. ) * in_dai;
        this.out_PPEc_S_1 = spec.getSpecValue( this.SPEC_out_PPEc_S, 0. );

        // 中間冷房入力(電力)
        this.out_PPEc_C   = spec.getSpecValue( this.SPEC_out_PPEc_C, 0. ) * in_dai;
        this.out_PPEc_C_1 = spec.getSpecValue( this.SPEC_out_PPEc_C, 0. );

        // 定格暖房能力
        this.out_Qh_S   = spec.getSpecValue( this.SPEC_out_Qh_S, 0. ) * in_dai;
        this.out_Qh_S_1 = spec.getSpecValue( this.SPEC_out_Qh_S, 0. );

        // 中間暖房能力
        this.out_Qh_C   = spec.getSpecValue( this.SPEC_out_Qh_C, 0. ) * in_dai;
        this.out_Qh_C_1 = spec.getSpecValue( this.SPEC_out_Qh_C, 0. );

/*      // 低温暖房能力
        this.out_Qh_L   = spec.getSpecValue( this.SPEC_out_Qh_L, 0. ) * in_dai;
        this.out_Qh_L_1 = spec.getSpecValue( this.SPEC_out_Qh_L, 0. ) * in_dai;
*/
        // 定格暖房入力(電力)
        this.out_PPEh_S   = spec.getSpecValue( this.SPEC_out_PPEh_S, 0. ) * in_dai;
        this.out_PPEh_S_1 = spec.getSpecValue( this.SPEC_out_PPEh_S, 0. );

        // 中間暖房入力(電力)
        this.out_PPEh_C   = spec.getSpecValue( this.SPEC_out_PPEh_C, 0. ) * in_dai;
        this.out_PPEh_C_1 = spec.getSpecValue( this.SPEC_out_PPEh_C, 0. );

/*      // 低温暖房入力(電力)
        this.out_PPEout_PPEh_Lh_C   = spec.getSpecValue( this.SPEC_out_PPEh_L, 0. ) * in_dai;
        this.out_PPEout_PPEh_Lh_C_1 = spec.getSpecValue( this.SPEC_out_PPEh_L, 0. );
*/
/*
        // クランクケースヒータ(運転時)
        this.out_CLEa_R   = spec.getSpecValue( this.SPEC_out_CLEa_R, 0. ) * in_dai;
        this.out_CLEa_R_1 = spec.getSpecValue( this.SPEC_out_CLEa_R, 0. );

        // クランクケースヒータ(停止時)
        this.out_CLEa_S   = spec.getSpecValue( this.SPEC_out_CLEa_S, 0. ) * in_dai;
        this.out_CLEa_S_1 = spec.getSpecValue( this.SPEC_out_CLEa_S, 0. );

        // 待機電力(運転時)
        this.out_TKEa_R   = spec.getSpecValue( this.SPEC_out_TKEa_R, 0. ) * in_dai;
        this.out_TKEa_R_1 = spec.getSpecValue( this.SPEC_out_TKEa_R, 0. );

        // 待機電力(待機時)
        this.out_TKEa_T   = spec.getSpecValue( this.SPEC_out_TKEa_T, 0. ) * in_dai;
        this.out_TKEa_T_1 = spec.getSpecValue( this.SPEC_out_TKEa_T, 0. );

        // 待機電力(停止時)
        this.out_TKEa_S   = spec.getSpecValue( this.SPEC_out_TKEa_S, 0. ) * in_dai;
        this.out_TKEa_S_1 = spec.getSpecValue( this.SPEC_out_TKEa_S, 0. );
*/

        //SPEC_de_fRate_watInHS   = "熱源水定格水量[g/s]";
        this.de_fRate_watInHS   = spec.getSpecValue( this.SPEC_de_fRate_watInHS, 0. ) * in_dai;//20160601
        this.de_fRate_watInHS_1 = spec.getSpecValue( this.SPEC_de_fRate_watInHS, 0. );//20160601

        if( this.de_fRate_watInHS <= 0 ){
           BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                 SystemMessageConstants.CAL_ERROR2 ,
                 new String[]{ "( "+de_fRate_watInHS+"/"+moduleName+" )[ PAC EHP水熱源 ]", "熱源水定格水量を設定してくださ
い"} ));
           this.de_fRate_watInHS = this.out_Qc_S * 0.11;//20110505nino
        }

/*      // 機器起動停止負荷率
        if(null!=map.get(this.SPEC_in_run_stop)){
           this.in_run_stop = Double.parseDouble((String)map.get(this.SPEC_in_run_stop))*100.0;
        } else {
           System.out.println(this.moduleName +"(E)SPEC_機器起動停止負荷率がありません");
```

```
        in_run_stop=30.0;
    }
*/
    // 相数
    this.phase = spec.getSpecValue( this.SPEC_Phase, 3 );

    // 電圧
    this.voltage = spec.getSpecValue( this.SPEC_Voltage, 200. );

    // 周波数
    this.frequency = spec.getSpecValue( this.SPEC_Frequency, 50. );

    // 力率
    this.powerFactor = spec.getSpecValue( this.SPEC_PowerFactor, 0.8 );


    if(out_Qc_S<0.001)      fmc= 0.0;
       else           fmc= out_Qc_C/out_Qc_S;
    if(out_PPEc_S<0.001) pmc= 0.0;
       else           pmc= out_PPEc_C/out_PPEc_S;

    if(out_Qh_S<0.001)      fmh= 0.0;
       else           fmh= out_Qh_C/out_Qh_S;
    if(out_PPEh_S<0.001) pmh= 0.0;
       else           pmh= out_PPEh_C/out_PPEh_S;

    if(pmc>0.9999) {pmc=0.0;fmc=0.0;}
    if(pmh>0.9999) {pmh=0.0;fmh=0.0;}



    // 定格風量
    this.in_Va_S   = spec.getSpecValue( this.SPEC_in_Va_S, 0. ) * in_dai;
    this.in_Va_S_1 = spec.getSpecValue( this.SPEC_in_Va_S, 0. );

    if( this.in_Va_S <= 0 ){
       BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
             SystemMessageConstants.CAL_ERROR2 ,
             new String[]{ "( "+in_Va_S+"/"+moduleName+" )[ PAC EHP水熱源 ]", "風量を設定してください"} ));

    }

    // 定格ファン消費電力
    this.in_PPEa_S   = spec.getSpecValue( this.SPEC_in_PPEa_S, 0. ) * in_dai;
    this.in_PPEa_S_1 = spec.getSpecValue( this.SPEC_in_PPEa_S, 0. );

/*
    // 待機電力(運転時)
    this.in_TKEa_R   = spec.getSpecValue( this.SPEC_in_TKEa_R, 0. ) * in_dai;
    this.in_TKEa_R_1 = spec.getSpecValue( this.SPEC_in_TKEa_R, 0. );

    // 待機電力(停止時)
    this.in_TKEa_S   = spec.getSpecValue( this.SPEC_in_THEa_S, 0. ) * in_dai;
    this.in_TKEa_S_1 = spec.getSpecValue( this.SPEC_in_THEa_S, 0. );
*/

    // 機器起動停止負荷率
    this.in_run_stop = spec.getSpecValue( this.SPEC_in_run_stop, 0.3 );

    this.in_run_stop_Num = this.in_run_stop;

    // 定格加湿能力
    this.in_HUM_mx   = spec.getSpecValue( this.SPEC_in_HUM_mx, 0. ) * in_dai;
    this.in_HUM_mx_1 = spec.getSpecValue( this.SPEC_in_HUM_mx, 0. );

    // 加湿飽和効率
    this.in_HUM_rt   = spec.getSpecValue( this.SPEC_in_HUM_rt, 0.7 ) * 100.;

    // 加湿器ONOFF湿度
    this.in_HUM_on   = spec.getSpecValue( this.SPEC_in_HUM_on, 0.4 ) * 100.;

    // 取入外気量
```

```
    this.in_OA_S    = spec.getSpecValue( this.SPEC_in_OA_S, 0. ) * in_dai;
    //this.in_OA_S_1 = spec.getSpecValue( this.SPEC_in_OA_S, 0. );

    // 全熱交換器効率
    this.in_EX_S = spec.getSpecValue( this.SPEC_in_EX_S, 0.6 );

    //SPEC_isHexbypass = "熱交バイパスあり";
    this.isHexbypass = spec.getSpecValue( this.SPEC_isHexbypass, false );

    // 全熱交換器消費電力[W]
    this.in_EX_PE    = spec.getSpecValue( this.SPEC_in_EX_PE, 0. ) * in_dai;
    //this.in_EX_PE_1 = spec.getSpecValue( this.SPEC_in_EX_PE, 0. );


    //isGVisibleを取得
    this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

    //isRecordを取得
    this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );


    //低負荷領域の計算方法
    this.calcTypeLowerRangeLoad = spec.getSpecValue( this.SPEC_CalcTypeLowerRangeLoad, "0_発停運転" );
    //calcTypeLowerRangeLoad
    //  0_発停運転
    //  1_下限入力値固定
    //  2_下限COP値固定
    //  3_下限入力値と中間切片
    if( this.calcTypeLowerRangeLoad.equals( "0_発停運転" )){
       this.num_calcTypeLowerRangeLoad = 0;
    }else if(this.calcTypeLowerRangeLoad.equals( "1_下限入力値固定" )){
       this.num_calcTypeLowerRangeLoad = 1;
    }else if(this.calcTypeLowerRangeLoad.equals( "2_下限COP値固定" )){
       this.num_calcTypeLowerRangeLoad = 2;
    }else if(this.calcTypeLowerRangeLoad.equals( "3_下限入力値と中間切片" )){
       this.num_calcTypeLowerRangeLoad = 3;
    }else{
       this.num_calcTypeLowerRangeLoad = 0;
    }

    //isAdjust2012 = "台数を調整する";//
    this.isAdjust2012 = spec.getSpecValue( this.SPEC_isAdjust2012, false );

    // 調整の計算ステップ数
    this.numAdjustSteps = spec.getSpecValue( this.SPEC_NumAdjustSteps, 12 );


    this.daiListc = new LinkedList<Double>();
    this.daiListh = new LinkedList<Double>();
    for( int i=0; i<this.numAdjustSteps; i++ ){
       this.daiListc.add( 0. );
       this.daiListh.add( 0. );
    }
    this.rcList = new LinkedList<Double>();
    this.rhList = new LinkedList<Double>();
    for( int i=0; i<this.numAdjustSteps; i++ ){
       this.rcList.add( 0. );
       this.rhList.add( 0. );
    }

    this.fRate_watInHSc = this.de_fRate_watInHS;
    this.fRate_watInHSh = this.de_fRate_watInHS;

}



@Override
public void initialize(IBestStateMessage stateNodes,
      IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
      IBestDomainSchedule schedule) {
    //System.out.println(this.moduleName + " BuillMultiOutイニシャライズ");
```

```
        //状態ノードを受け取る
        super.sm = stateNodes;
        //制御ノードを受け取る
        super.cm = commandNodes;
        //記録ノードを受け取る
        super.rm = recordNodes;

        //eleIn
        this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

        //airInOA
        this.airInOA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInOA );
        this.airInOA.setMaxFlowRate( this.in_OA_S );//20150423

        //airOutEA
        this.airOutEA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutEA );
        this.airOutEA.setMaxFlowRate( this.in_OA_S );//20150423

        //watInHS
        this.watInHS = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInHS );

        //watOutHS
        this.watOutHS = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutHS );

        //グラフ表示の準備
        if( this.isGVisible ){
           gBMout = new GraphJFrameBuilMultiOut_S20101212( this.name, 300,  out_Qc_S * 1.5, 0 );
        }


//    rmlinemap= (Map<String,BM_EHPdata>)super.sm.getState(super.getConnectionNode(this.S_NODE_valInLine));
//    if(null == rmlinemap) {
//       rmlinemap = new HashMap<String,BM_EHPdata>();
//       System.out.println("out=NULL ");
//    } else {
//       num = 0;
//       Set<String> LineEntry = this.rmlinemap.keySet();
//       for(Iterator i = LineEntry.iterator();i.hasNext();){
//          String key = (String) i.next();
////           System.out.println(" name="+name+" key="+key);
//          RMdata = rmlinemap.get(key);
//          rmnm[num]=key;
////           System.out.println("key="+key+" 0="+rmdata[0]+" 1="+rmdata[1]+" 2="+rmdata[2]+" 3="+rmdata[3]+"
4="+rmdata[4]+" 5="+rmdata[5]+" 6="+rmdata[6]);
//          num++;
//       }
//       //出力設定
//       for(int i=0;i<num;i++){
//          RMdata = rmlinemap.get(rmnm[i]);
//          RMdata.set_kiki(kiki_file);
//          rmlinemap.put(rmnm[i],RMdata);
//       }
//       super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine), rmlinemap);
//    }


        filenames[0]=kiki_file;
        equipmentName=kiki_file;
        pacDB=new PACDBManager(path,filenames);
        pacDB.setEquipmentName(equipmentName);

        //20160601 rangeMax Min
        this.rangeMaxCOOLING_T_watIn = this.pacDB.getformulaRangeMax( this.pacDB.getFACKctw() );//熱源水入口温度上限
        this.rangeMinCOOLING_T_watIn = this.pacDB.getformulaRangeMin( this.pacDB.getFACKctw() );//熱源水入口温度下限
        this.rangeMaxCOOLING_wbIn = this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );//室WB上限
        this.rangeMinCOOLING_wbIn = this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );//室WB下限
        this.rangeMaxHEATING_T_watIn = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhtw() );
        this.rangeMinHEATING_T_watIn = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhtw() );
        this.rangeMaxHEATING_dbRA = this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );//室DB上限
        this.rangeMinHEATING_dbRA = this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );//室DB下限

        //airOutRM
```

```
      this.airOutSA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutSA );
      this.airOutSA.setMaxFlowRate( this.in_Va_S );//20150423
      if( this.isUseZoneAir ){
         this.zoneAirforSSinside.checkNumberAirChange( this.in_Va_S, "in_Va_S", this.moduleName, "initialize()",
this.name, this.isRecord, this.message);
         this.zoneAirforSSinside._initialize( );
      }

      //**************090407追加**************
      //watInCW
      this.watInCW = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInCW );

      //****************************************


      //watOutD
      this.watOutD = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutD );

      //出口


      //接続ノード　入口
      //S_NODE_airInRA
      this.airInRA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInRA );
      this.airInRA.setTempDB(24.0);
      this.airInRA.setHumi(0.01);
      this.airInRA.setMaxFlowRate( this.in_Va_S );//20150423

      //S_NODE_valInPID
      this.PIDrate = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInPID );


      this.in_Qc_S = this.out_Qc_S;
      this.in_Qh_S = this.out_Qh_S;

      //グラフ表示の準備
      if( this.isGVisible ){
         gBMIn = new GraphJFrameBuilMultiIn_S20101212( this.name, 300, in_Qc_S * 1.5, in_Va_S * 1.5 );
      }

   }


   /**
    * 水冷EHP計算及び結果出力/各室内機よりも先に動きます。
    */
   @Override
   public void outputs() {

      if (null == super.cm || null == super.sm ){
         return;
      }

      /**
       * 入力
       */
      //---20060211-----------------------------------------------
      this.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
      this.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
      this.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
      this.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]
//    allSheat = 0.0;
//    allTheat = 0.0;
//    DBra_max = -100.0;
//    WBra_min = 1000.0;
//    Lratemin = 100.0;

      if(dbflg==0){
         //filenames[0]=RMdata.get_kiki();
         //equipmentName=RMdata.get_kiki();
         //pacDB=new PACDBManager(path,filenames);
         //pacDB.setEquipmentName(equipmentName);
```

```
        }
        dbflg = 1;

        //
        this.airInRA = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airInRA));
        if( this.isUseZoneAir ){
           this.zoneAirforSSinside._outputSetRA( this.airInRA );
        }

        this.airInOA = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airInOA));
        this.watInHS = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watInHS ));
        if( Double.isNaN( watInHS.getTemp() ) ){//20150305
           BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                 SystemMessageConstants.WARNING ,
                 new String[]{ "(E)入口水温が異常値", this.name }));
           BestLogger.info( "(E)入口水温が異常値"
                 + "__"+this.name+"__("+moduleName+"_outputs_watInHS)" + AirSystemControl.getTimeStr() );
        }


        //停止中も熱源水は入口の状態で出ていくものとする
        this.watOutHS.copyAllState( this.watInHS );//20130110

        this.swcIn    = super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcIn));
        this.swcInOA  = super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcInOA));
        this.modIn    = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modIn));
        this.modInPID = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modInPID));//20160601


        //
        if( Airswc.isOFF( this.swcIn )){
           //停止
           this.mState = 0;
        }else if( this.watInHS.getFlowRate() == 0 && !this.isAdjust2012 ){
           //停止    //20110622nino
           this.mState = 0;
           this.message.append( "(W)熱源水流量=0→停止" );
        }else if( Airswc.isON( this.swcIn )){
           //運転
           if( Airmod.isCOOL( this.modIn ) && Airmod.isHEAT( this.modIn ) ){
              //冷暖房運転
              if( Airmod.isCOOL( this.modInPID ) ){
                 //冷房運転
                 this.mState = 1;
              }else if( Airmod.isHEAT( this.modInPID ) ){
                 //暖房運転
                 this.mState = 2;
              }else if( Airmod.isVENTILATE( this.modInPID )){
                 //換気モード
                 this.mState = 3;
              }else{
                 //停止
                 this.mState = 0;
              }
           }else if( Airmod.isCOOL( this.modIn )){
              //冷房運転
              this.mState = 1;
           }else if( Airmod.isHEAT( this.modIn )){
              //暖房運転
              this.mState = 2;
           }else if( Airmod.isVENTILATE( this.modIn )){
              //換気モード
              this.mState = 3;
           }else{
              //停止
              this.mState = 0;
           }
           BestAir.checkOpeData( this.airInRA, "airInRA", this.moduleName, "outputs()", this.name, this.isRecord,
this.message );
        }

        /**
```

```
 * 入力
 */

//状態ノードから室内空気を設定
//室内乾球温度を設定
this.DBra = airInRA.getTempDB();

//室内絶対湿度を設定
this.XGra = airInRA.getHumi();


if( Double.isInfinite( this.DBra ) || Double.isNaN( this.DBra )){
   this.message.append( "(E)DBra異常値→0℃に設定");
   this.mState= -1;
   this.DBra = 0;
}
if( Double.isInfinite( this.XGra ) || Double.isNaN( this.XGra ) || this.XGra < 0 ){
   this.message.append( "(E)XGra異常値→0.004g/gに設定");
   this.mState= -1;
   this.XGra = 0.004;
}

//室内湿球温度を設定
this.WBra = Psychrometrics.FNWbtx(this.DBra, this.XGra);

//    System.out.println("DBin="+DBin+" "+XGin+"  "+WBin+"  ");

//外気乾球温度設定
this.DBoa = airInOA.getTempDB();

//外気絶対湿度を設定
this.XGoa = airInOA.getHumi();

//外気湿球温度を設定
this.WBoa = Psychrometrics.FNWbtx(this.DBoa, this.XGoa);
if( WBoa < -9000.0 ) WBoa =WB_cal(this.DBoa, this.XGoa);


//熱源水入口温度　流量
this.t_watInHS = this.watInHS.getTemp();
if( this.mState == 1 || this.mState == 2 ){//20160601　冷暖房時は熱源水の流量を定格値とする
   this.fRate_watInHS = this.de_fRate_watInHS;

}else{
   this.fRate_watInHS = 0;

}


//PID操作量
this.PIDrate = (BestValue) super.sm.getState( super.getConnectionNode( this.S_NODE_valInPID));

//    System.out.println("mState="+mState+" "+onOff+"  "+mode);
//    System.out.println("DBoa="+DBoa+" "+XGoa+"  "+WBoa);

//計算の切替

//室内機側の計算
switch(this.mState){
case -1:
   //異常停止
   this.message.append("(C)異常停止");
   this.calc_Stop_In0();
   break;
case 0:
   //停止
   this.message.append("(C)停止処理");
   this.calc_Stop_In0();
   break;
case 1:
   //冷房
   if( this.out_Qc_S_1 > 0 ){
```

```
            this.message.append("(C)冷房運転");
            this.cooling_cal_In();
            this.PPE_in = this.in_PPEa_S + this.in_TKEa_R + this.in_EX_PE;//20130717
         }else{
            this.message.append("(C)冷能力=0停止");//20130625
            this.calc_Stop_In0();
            this.mState = 0;
         }
         break;
      case 2:
         //暖房
         if( this.out_Qh_S_1 > 0 ){
            this.message.append("(C)暖房運転");
            this.heating_cal_In();
            this.PPE_in = this.in_PPEa_S + this.in_TKEa_R + this.in_EX_PE;//20130717
         }else{
            this.message.append("(C)暖能力=0停止");//20130625
            this.calc_Stop_In0();
            this.mState = 0;
         }
         break;
      case 3:
         //換気
         this.message.append("(C)換気運転");
         this.ventilation_cal();
         this.PPE_in = this.in_PPEa_S + this.in_TKEa_R + this.in_EX_PE;//20130717
         //
         this.PIDrate.setValue( 0. );
         break;
      default:
         this.message.append("(C)運転モード？停止");//20130625
         this.calc_Stop_In0();
         this.mState = 0;
         System.out.println(this.moduleName + ">>Error<< onOff*modeが範囲外");
      }

      this.allSc_heat = this.Sc_Load;//要求冷房顕熱量
      this.allSh_heat = this.Sh_Load;//要求暖房顕熱量
      this.allTc_heat = this.Tc_Load;//要求冷房全熱量
      this.allTh_heat = this.Th_Load;//要求暖房全熱量

      if( this.isAdjust2012 ){
         if( mState == 1 ){
            this.fRate_watInHS = this.fRate_watInHSc;
         }else if( mState == 2 ){
            this.fRate_watInHS = this.fRate_watInHSh;
         }
      }

/*
      //*************************----------------
      if( this.isAdjust2012 ){
         double rc;
         double c_OutCapa =0;
         double rh;
         double h_OutCapa =0;


         //処理可能熱量　仮計算
         if( this.allTc_heat==0 && this.allTh_heat==0 ){
            //冷却・加熱要求なし
            c_OutCapa = this.out_Qc_S_1;
            h_OutCapa = this.out_Qh_S_1;
         }else if( this.allTh_heat == 0 ){
            //冷却要求のみ coolingOnly_cal()
            this.fRate_watInHS = this.fRate_watInHSc;//固定流量で調整する
            c_OutCapa = this.pacDB.getKctw( this.t_watInHS ) * this.pacDB.getKcf( this.fRate_watInHS /
this.fRate_watInHSc ) * this.pacDB.getKcti( this.WBin )* 1. * this.out_Qc_S_1;//仮計算　室外機処理可能熱量（冷却全熱）[W]
            h_OutCapa = this.out_Qh_S_1;
         }else if( this.allTc_heat == 0 ){
            //加熱要求のみ heatingOnly_cal()
            this.fRate_watInHS = this.fRate_watInHSh;//固定流量で調整する
```

```
            c_OutCapa = this.out_Qc_S_1;
            h_OutCapa = this.pacDB.getKhtw( this.t_watInHS ) * this.pacDB.getKhf( this.fRate_watInHS /
this.fRate_watInHSh ) * this.pacDB.getKhti( this.DBin )* 1. * this.out_Qh_S_1;//仮計算　室外機処理可能熱量（加熱全熱）[W]
         }else if( this.allTc_heat >= this.allTh_heat ){
            //冷却主体運転 cooling_and_heating_cal()
            //this.fRate_watInHS = this.fRate_watInHSc;//固定流量で調整する
            //c_OutCapa = this.pacDB.getKcmtw( this.t_watInHS ) * this.pacDB.getKcmf( this.fRate_watInHS /
this.fRate_watInHSc ) * this.pacDB.getKcti( this.WBra )* this.Lratemin * this.out_Qc_S_1;//仮計算　室外機処理可能熱量（冷却全熱）[W]
            //h_OutCapa = this.pacDB.getKhti( this.DBra ) * this.Lratemin * this.out_Qh_S_1;//室外機処理可能熱量（加熱全熱）[W]
         }else{
            //加熱主体運転 heating_and_cooling_cal()
            //this.fRate_watInHS = this.fRate_watInHSh;//固定流量で調整する
            //c_OutCapa = this.pacDB.getKcti( this.WBra) * this.Lratemin * out_Qc_S_1;//室外機処理可能熱量（冷却全熱）[W]
            //h_OutCapa = this.pacDB.getKhmtw( this.t_watInHS ) * this.pacDB.getKhmf( this.fRate_watInHS /
this.fRate_watInHSh ) * this.pacDB.getKhti( this.DBra )* this.Lratemin * this.out_Qh_S_1;//仮計算　室外機処理可能熱量（加熱全熱）[W]
         }

         //冷却能力調整
         if( c_OutCapa == 0 ){
            //rc = this.allTc_heat / this.out_Qc_S;
            rc = this.allTc_heat / this.out_Qc_S_1;
         }else{
            rc = this.allTc_heat / c_OutCapa;
         }

         if( rc > 1.1 ){
         // rc = 1.1;
         }

         //加熱能力調整
         if( h_OutCapa == 0 ){
            //rh = this.allTh_heat / this.out_Qh_S;
            rh = this.allTh_heat / this.out_Qh_S_1;
         }else{
            rh = this.allTh_heat / h_OutCapa;
         }

         if( rh > 1.1 ){
         // rh = 1.1;
         }

         //*************************----------------
         if( true ){
            //移動平均の最大値で調整していく
            this.rcList.removeLast();
            this.rcList.addFirst( rc );
            //
            double sum_rc = 0;
            for( int i=0; i<this.numAdjustSteps; i++ ){
               sum_rc += this.rcList.get( i );
            }
            this.ave_rcAdjust = sum_rc / this.numAdjustSteps;
            //
            if( sum_rc > this.max_rcAdjust * this.numAdjustSteps ){
               this.max_rcAdjust = sum_rc / this.numAdjustSteps;

               this.out_rcAdjust = this.max_rcAdjust;
               //this.in_run_stop_Num = this.in_run_stop / this.out_rcAdjust;

               this.out_Qc_S   = this.out_Qc_S_1    * this.out_rcAdjust;
               this.out_Qc_C   = this.out_Qc_C_1    * this.out_rcAdjust;
               this.out_PPEc_S = this.out_PPEc_S_1  * this.out_rcAdjust;
               this.out_PPEc_C = this.out_PPEc_C_1  * this.out_rcAdjust;

               this.fRate_watInHSc = this.de_fRate_watInHS * this.out_rcAdjust;

               this.in_Va_S   = this.in_Va_S_1   * this.out_rcAdjust;//= "定格風量";//m3/h
               this.in_PPEa_S = this.in_PPEa_S_1 * this.out_rcAdjust;//= "定格ファン消費電力";//W
```

```
            this.in_HUM_mx = this.in_HUM_mx_1 * this.out_rcAdjust;//= "定格加湿能力";//kg/h

         }
      }


      if( true ){
         //移動平均の最大値で調整していく
         this.rhList.removeLast();
         this.rhList.addFirst( rh );
         //
         double sum_rh = 0;
         for( int i=0; i<this.numAdjustSteps; i++ ){
            sum_rh += this.rhList.get( i );
         }
         this.ave_rhAdjust = sum_rh / this.numAdjustSteps;
         //
         if( sum_rh > this.max_rhAdjust * this.numAdjustSteps ){
            this.max_rhAdjust = sum_rh / this.numAdjustSteps;

            this.out_rhAdjust = this.max_rhAdjust;
            //this.in_run_stop_Num = this.in_run_stop / this.out_rcAdjust;

            this.out_Qh_S   = this.out_Qh_S_1   * this.out_rhAdjust;
            this.out_Qh_C   = this.out_Qh_C_1   * this.out_rhAdjust;
            this.out_Qh_L   = this.out_Qh_L_1   * this.out_rhAdjust;
            this.out_PPEh_S = this.out_PPEh_S_1 * this.out_rhAdjust;
            this.out_PPEh_C = this.out_PPEh_C_1 * this.out_rhAdjust;
            this.out_PPEh_L = this.out_PPEh_L_1 * this.out_rhAdjust;

            this.fRate_watInHSh = this.de_fRate_watInHS * this.out_rhAdjust;

            this.in_Va_S   = this.in_Va_S_1   * this.out_rhAdjust;//= "定格風量";//m3/h
            this.in_PPEa_S = this.in_PPEa_S_1 * this.out_rhAdjust;//= "定格ファン消費電力";//W

            this.in_HUM_mx = this.in_HUM_mx_1 * this.out_rhAdjust;//= "定格加湿能力";//kg/h

         }
      }
      //*************************----------------


      //調整補正
/*    if( rc > 1.0 ){
         this.out_Qc_S *= rc;
         this.out_Qc_C *= rc;
         this.out_PPEc_S *= rc;
         this.out_PPEc_C *= rc;
         //
         this.fRate_airEAc *= rc;
      }
*/
      //調整補正
/*    if( rh > 1.0 ){
         this.out_Qh_S *= rh;
         this.out_Qh_C *= rh;
         this.out_Qh_L *= rh;
         this.out_PPEh_S *= rh;
         this.out_PPEh_C *= rh;
         this.out_PPEh_L *= rh;
         //
         this.fRate_airEAh *= rh;
      }
*/ /*
      this.out_Qc_Adjust = this.out_Qc_S;
      this.out_Qh_Adjust = this.out_Qh_S;

      //System.out.println( "BM out  out_Qc_Adjust=" +out_Qc_Adjust+"  out_Qh_Adjust="+out_Qh_Adjust);
   }

*/
   //*************************----------------
```

```
//室外機側の計算
switch(this.mState){
case -1:
case 0:
   //停止
   this.message.append("(C)停止");
   this.calc_Stop_Out();
   break;
case 1:
   //冷房
   if( this.out_Qc_S_1 > 0 ){
      this.message.append("(C)冷房運転");
      this.cooling_cal_Out();

      if( this.CAPrateC == 0 ){
         this.PIDrate.setValue( 0. );
      }
      this.message.append("(C)冷房処理PID="+AirFormat.df_2(this.PIDrate.getValue()));
   }else{
      this.message.append("(C)冷能力=0停止");//20130625
      this.calc_Stop_Out();
      this.mState = 0;
   }

   break;
case 2:
   //暖房
   if( this.out_Qh_S_1 > 0 ){
      this.message.append("(C)暖房運転");
      this.heating_cal_Out();

      if( this.CAPrateH == 0 ){
         this.PIDrate.setValue( 0. );
      }
      this.message.append("(C)暖房処理PID="+AirFormat.df_2(this.PIDrate.getValue()));
   }else{
      this.message.append("(C)暖能力=0停止");//20130625
      this.calc_Stop_Out();
      this.mState = 0;
   }

   break;
case 3:
   //換気
   this.message.append("(C)換気");
   this.allSc_heat  = 0.0;
   this.allTc_heat  = 0.0;
   this.allSh_heat  = 0.0;
   this.allTh_heat  = 0.0;
   this.Sc_Load = 0.0;
   this.Sh_Load = 0.0;
   this.Tc_Load = 0.0;
   this.Th_Load = 0.0;

   this.PPE_out = this.out_TKEa_T;
   this.CAPrateC = 0.0;
   this.CAPrateH = 0.0;
   //
   this.PIDrate.setValue( 0. );
   break;
default:
   this.message.append("(C)運転モード？停止");//20130625
   this.calc_Stop_Out();
   this.mState = 0;
   //System.out.println(this.moduleName + ">>Error<< onOff*modeが範囲外");
}


super.sm.setState( super.getConnectionNode( this.S_NODE_watOutHS ), this.watOutHS );

this.E_PPE_in = this.PPE_in * Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 );
```

```
this.E_PPE_out= this.PPE_out* Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 );
this.eleIn.setActivePower( this.PPE_out + this.PPE_in );
this.eleIn.setReactivePower(this.E_PPE_out + this.E_PPE_in );
this.eleIn.setPhase( this.phase );
this.eleIn.setVoltage( this.voltage );
this.eleIn.setFrequency( this.frequency );


this.Sc_Load_RM = this.CAPrateC * this.Sc_Load;
this.Tc_Load_RM = this.CAPrateC * this.Tc_Load;
this.Sh_Load_RM = this.CAPrateH * this.Sh_Load;
this.Th_Load_RM = this.CAPrateH * this.Th_Load;

//    System.out.println("CAPrate="+CAPrate+" "+S_Load_RM+" "+S_Load+ " "+mState);

//室内機側の再計算
switch( this.mState ){
case -1:
   //異常停止
   this.message.append("(C)異常処理");
   this.calc_Stop_In();
   break;
case 0:
   //停止
   this.message.append("(C)停止処理");
   this.calc_Stop_In();
   break;
case 1:
   //冷房
   if( this.out_Qc_S_1 > 0 ){
      this.GW = this.in_Va_S;
      this.cooling_supply();
      this.D_Wrate = ( this.XGin - this.XGout ) * this.GW;//g/s
      if( this.D_Wrate < 0.0 ){
         this.D_Wrate = 0.0;//追加090619
      }
      this.CW_Wrate = 0.0;
      this.S_Load_RM = this.Sc_Load_RM;
      this.T_Load_RM = this.Tc_Load_RM;
      if( this.eleIn.getActivePower() != 0. ){
         this.COP = this.T_Load_RM / this.eleIn.getActivePower();
      }else{
         this.COP = 0.0;
      }
      this.message.append("(C)冷房処理PID="+AirFormat.df_2(this.PIDrate.getValue()));
   }else{
      this.message.append("(C)冷能力=0停止");//20130625
      this.calc_Stop_In();
   }
   break;
case 2:
   //暖房
   if( this.out_Qh_S_1 > 0 ){
      this.GW = this.in_Va_S;
      this.heating_supply();
      this.D_Wrate = 0.0;
      this.S_Load_RM = this.Sh_Load_RM;
      this.T_Load_RM = this.Th_Load_RM;
      if( this.eleIn.getActivePower() != 0. ){
         this.COP = this.T_Load_RM / this.eleIn.getActivePower();
      }else{
         this.COP = 0.0;
      }
      this.message.append("(C)暖房処理PID="+AirFormat.df_2(this.PIDrate.getValue()));
   }else{
      this.message.append("(C)暖能力=0停止");//20130625
      this.calc_Stop_In();
   }
   break;
case 3:
   //換気
   this.GW = this.in_Va_S;
```

```
    this.ventilation_supply();
    this.D_Wrate = 0.0;
    this.S_Load_RM = 0.0;
    this.T_Load_RM = 0.0;
    this.COP = 0.0;
    //
    this.PIDrate.setValue( 0. );
    this.message.append("(C)換気処理");
    break;
default:
    this.message.append("(C)運転モード？停止");//20130625
    this.calc_Stop_In();
    //System.out.println( this.moduleName + ">>Error<< onOff*modeが範囲外");
}
//    System.out.println("DBout="+DBout);

this.airOutSA.setTempDB( this.DBout );
this.airOutSA.setHumi( this.XGout );
this.airOutSA.setFlowRate( this.GW );

this.watOutD.setFlowRate( this.D_Wrate );
this.watOutD.setTemp( this.DBout );
this.watInCW.setFlowRate( this.CW_Wrate );
//    System.out.println("DBout="+DBout+" "+XGout+"  "+airOutRM.getFlowRate());

//出力側ノード
super.sm.setState( super.getConnectionNode( this.S_NODE_airOutSA ), this.airOutSA);
super.sm.setState( super.getConnectionNode( this.S_NODE_eleIn ), this.eleIn);
super.sm.setState( super.getConnectionNode( this.S_NODE_watOutD ), this.watOutD);
super.sm.setState( super.getConnectionNode( this.S_NODE_watInCW ), this.watInCW);//090407追加

//グラフデータ追加
if( this.isGVisible ){
    gBMIn.addData( BestTimeManager.getDateWeatherTime(),
            this.Sc_Load_RM,
            this.Tc_Load_RM,
            this.Sh_Load_RM,
            this.Th_Load_RM,
            this.PPE_in,
            this.DBoa,
            this.DBra,
            this.DBin,
            this.DBout,
            this.XGoa,
            this.XGra,
            this.XGin,
            this.XGout,
            this.GW,
            this.D_Wrate );
}


//出力設定

//グラフデータ追加
if( this.isGVisible ){
    gBMout.addData( BestTimeManager.getDateWeatherTime(),
    this.Sc_Loadout,//室外機処理可能熱量（冷却）[W]
    this.Sh_Loadout,//室外機処理可能熱量（加熱）[W]
    this.allTc_heat,//室内機処理要求熱量（冷却）[W]
    this.allTh_heat,//室内機処理要求熱量（加熱）[W]
    this.PPE_out,
    this.DBoa,
    this.DBin,
    this.XGoa,
    this.XGin,
    this.CAPrateC );
}

//記録ノード
if( this.isRecord && super.rm != null ){
    this.record();
```

```
}

message.setLength(0);


//*************************----------------
if( this.isAdjust2012 ){
   //移動平均の最大値で調整していく
   if( true ){
      this.daiListc.removeLast();
      this.daiListc.addFirst( this.in_daicAdjust * this.PIDrate.getValue() );
      //
      double sum_in_dai = 0;
      for( int i=0; i<this.numAdjustSteps; i++ ){
         sum_in_dai += this.daiListc.get( i );
      }
      this.avecdaiAdjust = sum_in_dai / this.numAdjustSteps;
      //
      if( sum_in_dai > this.maxcAdjustdai * this.numAdjustSteps ){
         this.maxcAdjustdai = this.avecdaiAdjust;

         this.in_daicAdjust = this.maxcAdjustdai;
         this.in_run_stop_Num = this.in_run_stop / this.in_daicAdjust;

         this.out_Qc_S   = this.out_Qc_S_1   * this.in_daicAdjust;//= "定格冷房能力";//kW
         this.out_Qc_C   = this.out_Qc_C_1   * this.in_daicAdjust;//= "中間冷房能力";//kW
         this.out_PPEc_S = this.out_PPEc_S_1 * this.in_daicAdjust;//= "定格冷房入力(電力)";//kW
         this.out_PPEc_C = this.out_PPEc_C_1 * this.in_daicAdjust;//= "中間冷房入力(電力)";//kW
         this.in_Qc_S = this.out_Qc_S;

         this.fRate_watInHSc = this.de_fRate_watInHS_1 * this.in_daicAdjust;//水量[g/s]

         this.in_Va_S   = this.in_Va_S_1   * this.in_daicAdjust;//= "定格風量";//m3/h
         this.in_PPEa_S = this.in_PPEa_S_1 * this.in_daicAdjust;//= "定格ファン消費電力";//W

         this.in_HUM_mx = this.in_HUM_mx_1 * this.in_daicAdjust;//= "定格加湿能力";//kg/h

         this.airInRA.setFlowRate(in_Va_S);
         this.airOutSA.setFlowRate(in_Va_S);
      }
   }


   if( true ){
      this.daiListh.removeLast();
      this.daiListh.addFirst( this.in_daihAdjust * this.PIDrate.getValue() );
      //
      double sum_in_dai = 0;
      for( int i=0; i<this.numAdjustSteps; i++ ){
         sum_in_dai += this.daiListh.get( i );
      }
      this.avehdaiAdjust = sum_in_dai / this.numAdjustSteps;
      //
      if( sum_in_dai > this.maxhAdjustdai * this.numAdjustSteps ){
         this.maxhAdjustdai = this.avehdaiAdjust;

         this.in_daihAdjust = this.maxhAdjustdai;
         this.in_run_stop_Num = this.in_run_stop / this.in_daihAdjust;

         this.out_Qh_S   = this.out_Qh_S_1   * this.in_daihAdjust;//= "定格暖房能力";//kW
         this.out_Qh_C   = this.out_Qh_C_1   * this.in_daihAdjust;//= "中間暖房能力";//kW
         this.out_Qh_L   = this.out_Qh_L_1   * this.in_daihAdjust;//= "低温暖房能力";//kW
         this.out_PPEh_S = this.out_PPEh_S_1 * this.in_daihAdjust;//= "定格暖房入力(電力)";//kW
         this.out_PPEh_C = this.out_PPEh_C_1 * this.in_daihAdjust;//= "中間暖房入力(電力)";//kW
         this.out_PPEh_L = this.out_PPEh_L_1 * this.in_daihAdjust;//= "低温暖房入力(電力)";//kW
         this.in_Qh_S = this.out_Qh_S;

         this.fRate_watInHSh = this.de_fRate_watInHS * this.in_daihAdjust;//水量[g/s]

         this.in_Va_S   = this.in_Va_S_1   * this.in_daihAdjust;//= "定格風量";//m3/h
         this.in_PPEa_S = this.in_PPEa_S_1 * this.in_daihAdjust;//= "定格ファン消費電力";//W
```

```
            this.in_HUM_mx = this.in_HUM_mx_1 * this.in_daihAdjust;//= "定格加湿能力";//kg/h

            this.airInRA.setFlowRate(in_Va_S);
            this.airOutSA.setFlowRate(in_Va_S);
          }

       }
    }
    //*************************----------------

}

private void calc_Stop_In0(){
    //出口空気の状態は入口空気の状態とする
    this.airOutSA.copyAllvalState( this.airInRA );
    this.airOutSA.setFlowRate(0.0);
    this.S_Load = 0.0;
    this.T_Load = 0.0;
    this.Sc_Load = 0.0;
    this.Tc_Load = 0.0;
    this.Sh_Load = 0.0;
    this.Th_Load = 0.0;
    //LPrate=1.0;
    this.PPE_in = this.in_TKEa_S;
    this.DBin = this.DBra;
    this.XGin = this.XGra;
    this.WBin = this.WBra;
    //
    this.PIDrate.setValue( 0. );

}

private void calc_Stop_In(){
    this.DBout = this.DBin;//乾球温度
    this.XGout = this.XGin;//絶対湿度
    this.GW = 0.0;
    this.D_Wrate = 0.0;
    this.CW_Wrate = 0.0;
    this.S_Load_RM = 0.0;
    this.T_Load_RM = 0.0;
    this.COP = 0.0;
    //
    this.PIDrate.setValue( 0. );
    //
}


private void calc_Stop_Out(){
    this.allSc_heat  = 0.0;
    this.allTc_heat  = 0.0;
    this.allSh_heat  = 0.0;
    this.allTh_heat  = 0.0;
    this.Sc_Load = 0.0;
    this.Sh_Load = 0.0;
    this.Tc_Load = 0.0;
    this.Th_Load = 0.0;

    this.Tc_Loadout = 0.0;//室外機実処理熱量[W]
    this.Sc_Loadout = 0.0;//室外機処理可能熱量[W]
    this.Th_Loadout = 0.0;//室外機実処理熱量[W]
    this.Sh_Loadout = 0.0;//室外機処理可能熱量[W]

    this.PPE_out = this.out_TKEa_S;
    this.CAPrateC = 0.0;
    this.CAPrateH = 0.0;
    //
    this.PIDrate.setValue( 0. );
}

/**
 * 記録
 */
```

```
   private void record(){
      if( super.rm != null ){
         if( CheckPrintModule.isPrintMessage ){
            //message
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_message, this.name, message.toString() );
         }

         if( CheckPrintModule.isPrintEnergy ){
            //記録ノードに室外機消費電力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID4, this.name,
AirFormat.df_2(this.eleIn.getActivePower()));
         }

         if( CheckPrintModule.isPrintLoad ){
            //記録ノードに室内機処理熱量（全熱）を設定
            if( this.Tc_Loadout > 0 ){
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT,  this.name, AirFormat.df_2(-
this.Tc_Loadout ));//室内機処理全熱量合計#W#熱量
            }else{
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT,  this.name,
AirFormat.df_2( this.Th_Loadout ));//室内機処理全熱量合計#W#熱量
            }
         }

         if( CheckPrintModule.isPrintStateOut ){
            //記録ノードに吹出口乾球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID4i, this.name, AirFormat.df_2(DBout));
            //記録ノードに吹出口絶対湿度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID5i, this.name, AirFormat.df_5(XGout));
            //記録ノードに吹出口風量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID6i, this.name, AirFormat.df_2(GW));
            //記録ノードにドレン量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID7i, this.name,
AirFormat.df_2(D_Wrate));
            //記録ノードに熱源水出口流量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_M_watOutHS, this.name,
AirFormat.df_2(this.watOutHS.getFlowRate()));
            //記録ノードに熱源水出口温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_T_watOutHS, this.name,
AirFormat.df_2(this.watOutHS.getTemp()));
         }

         if( CheckPrintModule.isPrintStateMy ){
            //記録ノードに室内機加熱能力補正比率を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID3c, this.name,
AirFormat.df_2(this.CAPrateC));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID3h, this.name,
AirFormat.df_2(this.CAPrateH));
            //記録ノードに室内機要求熱量（顕熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID1c, this.name,
AirFormat.df_2(this.allSc_heat));//冷却要求顕熱量合計#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID1h, this.name,
AirFormat.df_2(this.allSh_heat));//加熱要求顕熱量合計#W#熱量
            //記録ノードに室内機要求熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID2c, this.name,
AirFormat.df_2(this.allTc_heat));//冷却要求全熱量合計#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID2h, this.name,
AirFormat.df_2(this.allTh_heat));//加熱要求全熱量合計#W#熱量
            //記録ノードに室外機処理可能熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID5c, this.name,
AirFormat.df_2(this.Sc_Loadout));//冷却可能熱量/最大#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID5h, this.name,
AirFormat.df_2(this.Sh_Loadout));//加熱可能熱量/最大#W#熱量
            //記録ノードに室内機処理熱量（顕熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID1,  this.name,
AirFormat.df_2(S_Load_RM));//処理顕熱量合計#W#熱量
            //記録ノードに室内機処理熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID2,  this.name,
AirFormat.df_2(T_Load_RM));//処理全熱量合計#W#熱量

            //COP
```

```
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_COP, this.name, AirFormat.df_3(this.COP));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_pidVal, this.name,
AirFormat.df_4(this.PIDrate.getValue()));
         }


         if( CheckPrintModule.isPrintStateIn ){
            //記録ノードに加湿給水量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID8i, this.name,
AirFormat.df_2(CW_Wrate));
            //記録ノードに入口乾球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBairInRM, this.name,
AirFormat.df_2(DBin));
            //記録ノードに入口絶対湿度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_XairInRM, this.name,
AirFormat.df_5(XGin ));
            //記録ノードに熱源水出口流量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_M_watInHS, this.name,
AirFormat.df_2(this.watInHS.getFlowRate()));
            //記録ノードに熱源水出口温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_T_watInHS, this.name,
AirFormat.df_2(this.watInHS.getTemp()));
         }

         if( CheckPrintModule.isPrintAdjust ){
            //
            if( this.isAdjust2012 ){
               //記録ノードに室外機調整能力を設定
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qc_Adjust, this.name,
this.out_Qc_S );//室外機調整能力[W]out_Qc_Adjust
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qh_Adjust, this.name,
this.out_Qh_S );//室外機調整能力[W]out_Qh_Adjust
               //記録ノードに室外機調整能力を設定
               //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rcAdjust", this.name,
this.out_rcAdjust );//室外機調整率[-]
               //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rhAdjust", this.name,
this.out_rhAdjust );//室外機調整率[-]
               //記録ノードに調整台数を設定
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_in_daicAdjust, this.name,
AirFormat.df_2(this.in_daicAdjust) );
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_in_daihAdjust, this.name,
AirFormat.df_2(this.in_daihAdjust) );
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_avecdaiAdjust, this.name,
AirFormat.df_2(this.avecdaiAdjust) );
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_avehdaiAdjust, this.name,
AirFormat.df_2(this.avehdaiAdjust) );
            }
         }
      }
   }

   /**
    * 水冷EHP計算/各室内機の集計後
    */

   @Override
   public void update() {
      if( this.isUseZoneAir ){
         this.zoneAirforSSinside._update( this.airOutSA );
      }
   }

   public Object viewInternal(TestCommand cmd) {

      ArrayList<Object> result = new ArrayList<Object>();
      result.add(super.sm);
      result.add(super.cm);
      result.add(super.rm);
      //外部定義
      result.add(this.name);
      result.add(this.m_kt);
```

```
    result.add(this.airOutEA.getTempDB());

    result.add(this.out_Qc_S);
    result.add(this.out_Qc_C);
    result.add(this.out_PPEc_S);
    result.add(this.out_PPEc_C);
    result.add(this.out_Qh_S);
    result.add(this.out_Qh_C);
    result.add(this.out_Qh_L);
    result.add(this.out_PPEh_S);
    result.add(this.out_PPEh_C);
    result.add(this.out_PPEh_L);

    result.add(this.DBoa);
    result.add(this.XGoa);
    result.add(this.DBra);
    result.add(this.XGra);

    return result;
}



private void cooling_cal_Out(){
    double rCode0,rCode1,rCode2,rCode3,rCode4;
    double D_fmc,D_pmc,alph,beta;
    double Rp;

    D_fmc    =fmc;
    D_pmc    =pmc;

    //上下限のチェック 20120821nino
    double twatIn = this.t_watInHS;
    double wbIn = this.WBin;

    if( twatIn > this.rangeMaxCOOLING_T_watIn ){
       //熱源水入口温度＞上限の時停止
       this.message.append( "(C)熱源水入口温度＞上限"+rangeMaxCOOLING_T_watIn+"→停止");
       this.CAPrateC = 0.0;
       this.CAPrateH = 0.0;
       this.Tc_Loadout = 0.0;//室外機実処理熱量[W]
       this.Sc_Loadout = 0.0;//室外機処理可能熱量[W]
       this.Th_Loadout = 0.0;//室外機実処理熱量[W]
       this.Sh_Loadout = 0.0;//室外機処理可能熱量[W]
       this.PPE_out = this.out_TKEa_S;
       this.watOutHS.copyAllState( this.watInHS );//20130110
       //
       this.mState = 3;//能力＝０→換気モードとする20130415

       return;
    }
    if( twatIn < this.rangeMinCOOLING_T_watIn ){
       //熱源水入口温度＜下限の時　下限値の特性で運転
       this.message.append( "(C)熱源水入口温度＜下限"+this.rangeMinCOOLING_T_watIn+"→下限値で特性計算");
       twatIn = this.rangeMinCOOLING_T_watIn;
    }
    if( wbIn > this.rangeMaxCOOLING_wbIn ){
       //室内湿球温度＞上限の時　上限値の特性で運転
       this.message.append( "(C)室WB＞上限"+this.rangeMaxCOOLING_wbIn+"→上限値で特性計算");
       wbIn = rangeMaxCOOLING_wbIn;
    }
    if( wbIn < this.rangeMinCOOLING_wbIn ){
       //室内湿球温度＜下限の時　下限値の特性で運転
       this.message.append( "(C)室WB＜下限"+this.rangeMinCOOLING_wbIn+"→下限値で特性計算");
       wbIn = this.rangeMinCOOLING_wbIn;
    }


    //室温補正
    //能力補正
    rCode0 = this.pacDB.getKctw( twatIn );//水温補正
```

```
      rCode1 = this.pacDB.getKcf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
      rCode2 = this.pacDB.getKcti( wbIn );//室温補正

      this.Sc_Loadout = rCode0 * rCode1 * rCode2 * this.out_Qc_S;//室外機処理可能熱量[W]

      //if( twatIn==45 || twatIn==35 ){
      // System.out.println( "室外機能力 TwatIn="+twatIn+" wbIn="+wbIn+" rCode0="+rCode0+" rCode1="+rCode1+"
rCode2="+rCode2+" Sc_Loadout="+Sc_Loadout);
      //}

      this.Rc = this.allTc_heat / this.Sc_Loadout;

      this.CAPrateC = 1.0;
      this.Tc_Loadout = this.allTc_heat;//20120731

      if(this.Rc > 1.0 ) {
         this.CAPrateC = 1.0 / this.Rc;
         this.Rc = 1.0;
         this.Tc_Loadout = this.Sc_Loadout;//20120731
      }

      //低負荷領域の計算仕分け:Rh
      if( this.Rc < this.in_run_stop ){
         switch( this.num_calcTypeLowerRangeLoad){
         case 0://  0_発停運転
            Rp = 0;
            break;

         case 1://  1_下限入力値固定
         case 2://  2_下限COP値固定
         case 3://  3_下限入力値と中間切片
            Rp = this.in_run_stop;
            break;

         default:
            Rp = 0;
         }
      }else{
         Rp = this.Rc;
      }

      alph = this.pacDB.getAlphac( Rp );
      if( D_pmc * D_fmc < 0.001 ){
         beta = 1.0;
      }else{
         beta = this.pacDB.getBetac( Rp, D_fmc, D_pmc);
      }
      //       System.out.println("Rc="+Rc+" "+D_fmc+" "+D_pmc);

      //入力
      rCode0 = alph * beta;
      rCode1 = this.pacDB.getKchpid( Rp );//代表入力補正
      rCode2 = this.pacDB.getKcwtw( twatIn );//水温補正
      rCode3 = this.pacDB.getKcwf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
      rCode4 = this.pacDB.getKcwti( wbIn );

      this.PPE_out = rCode0 * rCode1 * rCode2 * rCode3 * rCode4 * this.out_PPEc_S + this.out_TKEa_R;

      //if( this.PPE_out == 0 ){
      // System.out.println( "rCode0="+rCode0+" rCode1="+rCode1+" rCode2="+rCode2+" rCode3="+rCode3+" rCode4="+rCode4);
      //}
      //if( twatIn==45 || twatIn==35 ){
      // System.out.println( "入力 TwatIn="+twatIn+" Rp="+Rp+"rCode0="+rCode0+" rCode1="+rCode1+" rCode2="+rCode2+"
rCode3="+rCode3+" rCode4="+rCode4);
      //}
      //低負荷領域の計算仕分け:PPE
      if( this.Rc < this.in_run_stop && Rp > 0 ){
         switch( this.num_calcTypeLowerRangeLoad){
         case 0://  0_発停運転
            this.PPE_out = this.out_TKEa_S;
            break;
```

```
            case 1://  1_下限入力値固定
               //何もしない
               break;

            case 2://  2_下限COP値固定
               this.PPE_out *= ( this.Rc / Rp );
               break;

            case 3://  3_下限入力値と中間切片
               this.PPE_out *= ( 0.5 + this.Rc / Rp / 2. );
               break;

            default:
            }
         }else{
            //何もしない
         }


         //airOutEA
         //this.airOutEA.setFlowRate( this.fRate_watInHS );
         //this.airOutEA.setTempDB( ( this.T_Loadout + this.PPE_out ) / this.fRate_watInHS * 1000. +
this.airInOA.getTempDB() );

         //watOutHS
         this.watOutHS.setFlowRate( this.fRate_watInHS );
         this.watOutHS.setTemp( ( this.Tc_Loadout + this.PPE_out ) / this.fRate_watInHS / 4.18605 + this.t_watInHS );

         if( this.Rc < 0.0001 ){
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量[W]
            this.PPE_out = this.out_TKEa_S;
            this.watOutHS.copyAllState( this.watInHS );//20130110
            this.message.append( "(C)負荷率0で停止");
            this.mState = 3;//能力＝０→換気モードとする20130415

         }else if( this.Rc < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
               this.CAPrateC = 0.0;
               this.CAPrateH = 0.0;
               this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
               this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
               this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
               this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
               this.PPE_out = this.out_TKEa_S;
               this.watOutHS.copyAllState( this.watInHS );//20130110
               this.message.append( "(C)負荷率＜停止負荷率で停止");
               this.mState = 3;//能力＝０→換気モードとする20130415
               break;

            case 1://  1_下限入力値固定
               break;

            case 2://  2_下限COP値固定
               break;

            case 3://  3_下限入力値と中間切片
               break;

            default:
         }

      }

      //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);

   }
```

```
  private void heating_cal_Out(){
     double rCode0,rCode1,rCode2,rCode3,rCode4;
     double D_fmh,D_pmh,D_out_Qh_S,D_out_PPEh_S,alph,beta;
     double Rp;

     D_fmh    = this.fmh;
     D_pmh    = this.pmh;
     D_out_Qh_S  = this.out_Qh_S;
     D_out_PPEh_S= this.out_PPEh_S;
     if( this.WBoa < 4.5 && this.WBoa > -7.0 && this.out_Qh_L > 0.0001 ) { //低温入力の場合
           D_fmh    = this.fmh;
        D_pmh      = this.pmh;
        D_out_Qh_S = this.out_Qh_L;
        D_out_PPEh_S= this.out_PPEh_L;
     }

     //上下限のチェック 20120821nino
     double twatIn = this.t_watInHS;
     double dbRA = this.DBin;

     if( twatIn > this.rangeMaxHEATING_T_watIn ){
        //熱源水入口温度＞上限の時　上限値の特性で運転
        this.message.append( "(C)熱源水入口温度＞上限"+this.rangeMaxHEATING_T_watIn+"→上限値で特性計算");
        twatIn = this.rangeMaxHEATING_T_watIn;
     }
     if( twatIn < this.rangeMinHEATING_T_watIn ){
        //熱源水入口温度＜下限の時　停止
        this.message.append( "(C)熱源水入口温度＜下限"+this.rangeMinHEATING_T_watIn+"→停止");
        this.CAPrateC = 0.0;
        this.CAPrateH = 0.0;
        this.Tc_Loadout = 0.0;//室外機実処理熱量[W]
        this.Sc_Loadout = 0.0;//室外機処理可能熱量[W]
        this.Th_Loadout = 0.0;//室外機実処理熱量[W]
        this.Sh_Loadout = 0.0;//室外機処理可能熱量[W]
        this.PPE_out = this.out_TKEa_S;
        this.E_PPE_out = 0.0;
        //this.airOutEA.setFlowRate( 0. );
        //this.airOutEA.setTempDB( this.airInOA.getTempDB() );
        //this.watOutHS.setFlowRate( 0.);//20130110
        //this.watOutHS.setTemp( this.watInHS.getTemp() );//20130110
        this.watOutHS.copyAllState( this.watInHS );//20130110
        //
        this.mState = 3;//能力＝０→換気モードとする20130415
        return;
     }
     if( dbRA > this.rangeMaxHEATING_dbRA ){
        //室内乾球温度＞上限の時　上限値の特性で運転
        this.message.append( "(C)室DB＞上限"+this.rangeMaxHEATING_dbRA+"→上限値で特性計算");
        dbRA = this.rangeMaxHEATING_dbRA;
     }
     if( dbRA < this.rangeMinHEATING_dbRA ){
        //室内乾球温度＜下限の時　下限値の特性で運転
        this.message.append( "(C)室DB＜下限"+this.rangeMinHEATING_dbRA+"→下限値で特性計算");
        dbRA = this.rangeMinHEATING_dbRA;
     }

     //能力補正
     rCode0 = this.pacDB.getKhtw( twatIn );//水温補正
     rCode1 = this.pacDB.getKhf( this.fRate_watInHS / this.fRate_watInHSh );//水量補正
     rCode2 = this.pacDB.getKhti( dbRA );//室温補正
     //rCode3 = this.Lratemin;
     rCode3 = 1.;//20120821nino

     this.Sh_Loadout  = rCode0 * rCode1 * rCode2 * rCode3 * D_out_Qh_S;//室外機処理可能熱量[W]

     //if( twatIn==45 || twatIn==35 ){
     // System.out.println( "室外機能力 TwatIn="+twatIn+" dbRAT="+dbRA+" rCode0="+rCode0+" rCode1="+rCode1+"
rCode2="+rCode2+" Sh_Loadout="+Sh_Loadout);
     //}
```

```
//       System.out.println("allTheat="+allTheat+" T_Loadout="+T_Loadout+" Rh="+Rc+" "+rCode1+" "+rCode2+"
"+rCode3);


      this.Rh = this.allTh_heat / this.Sh_Loadout;

      this.CAPrateH = 1.0;
      this.Th_Loadout = this.allTh_heat;//20120731

      if( this.Rh > 1.0 ) {
         this.CAPrateH = 1.0/ this.Rh;
         this.Rh = 1.0;
         this.Th_Loadout = this.Sh_Loadout;//20120731
      }

      //       System.out.println("Rh="+Rh);
      //低負荷領域の計算仕分け:Rh
      if( this.Rh < this.in_run_stop ){
         switch( this.num_calcTypeLowerRangeLoad){
         case 0://  0_発停運転
            Rp = 0;
            break;

         case 1://  1_下限入力値固定
         case 2://  2_下限COP値固定
         case 3://  3_下限入力値と中間切片
            Rp = this.in_run_stop;
            break;

         default:
            Rp = 0;
         }
      }else{
         Rp = this.Rh;
      }

      alph = this.pacDB.getAlphah( Rp);
      beta = this.pacDB.getBetah( Rp, D_fmh, D_pmh);

      if( D_pmh * D_fmh < 0.001 ){
         beta = 1.0;
      }

      //入力
      rCode0 = alph * beta;
      rCode1 = this.pacDB.getKhhpid( Rp );
      rCode2 = this.pacDB.getKhwtw( twatIn );//水温補正
      rCode3 = this.pacDB.getKhwf( this.fRate_watInHS / this.fRate_watInHSh );//水量補正
      rCode4 = this.pacDB.getKhwti( dbRA );

      this.PPE_out = rCode0 * rCode1 * rCode2 * rCode3 * rCode4 * D_out_PPEh_S + this.out_TKEa_R;

      //if( this.PPE_out == 0 ){
      // System.out.println( "rCode0="+rCode0+" rCode1="+rCode1+" rCode2="+rCode2+" rCode3="+rCode3+" rCode4="+rCode4);
      //}
      //if( twatIn==45 || twatIn==35 ){
      // System.out.println( "入力 TwatIn="+twatIn+" Rp="+Rp+"rCode0="+rCode0+" rCode1="+rCode1+" rCode2="+rCode2+"
rCode3="+rCode3+" rCode4="+rCode4);
      //}

      //低負荷領域の計算仕分け:PPE
      if( this.Rh < this.in_run_stop && Rp > 0 ){
         switch( this.num_calcTypeLowerRangeLoad){
         case 0://  0_発停運転
            this.PPE_out = this.out_TKEa_S;
            break;

         case 1://  1_下限入力値固定
            //何もしない
            break;

         case 2://  2_下限COP値固定
```

```
            this.PPE_out *= ( this.Rh / Rp );
            break;

         case 3://  3_下限入力値と中間切片
            this.PPE_out *= ( 0.5 + this.Rh / Rp / 2. );
            break;

         default:
         }
      }else{
         //何もしない
      }


      //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);
      //System.out.println(" PAC 負荷率="+this.Rh+"    負荷律係数="+rCode1);


      //airOutEA
      //this.airOutEA.setFlowRate( this.fRate_watInHS );
      //this.airOutEA.setTempDB( ( -this.T_Loadout + this.PPE_out ) / this.fRate_watInHS * 1000. +
this.airInOA.getTempDB() );
      //watOutHS
      this.watOutHS.setFlowRate( this.fRate_watInHS );
      this.watOutHS.setTemp( ( -this.Th_Loadout + this.PPE_out ) / this.fRate_watInHS / 4.18605 + this.t_watInHS );

      if( this.Rh < 0.0001 ){
         this.CAPrateC = 0.0;
         this.CAPrateH = 0.0;
         this.Tc_Loadout = 0.0;//室外機実処理熱量[W]
         this.Sc_Loadout = 0.0;//室外機処理可能熱量[W]
         this.Th_Loadout = 0.0;//室外機実処理熱量[W]
         this.Sh_Loadout = 0.0;//室外機処理可能熱量[W]
         this.PPE_out = this.out_TKEa_S;
         this.watOutHS.copyAllState( this.watInHS );//20130110
         this.message.append( "(C)負荷率0で停止");
         this.mState = 3;//能力＝０→換気モードとする20130415

      }else if( this.Rh < this.in_run_stop ){
         switch( this.num_calcTypeLowerRangeLoad){
         case 0://  0_発停運転
            this.CAPrateC = 0.0;
            this.CAPrateH = 0.0;
            this.Tc_Loadout = 0.0;//室外機実処理熱量（冷却全熱）[W]
            this.Sc_Loadout = 0.0;//室外機処理可能熱量（冷却全熱）[W]
            this.Th_Loadout = 0.0;//室外機実処理熱量（加熱全熱）[W]
            this.Sh_Loadout = 0.0;//室外機処理可能熱量（加熱全熱）[W]
            this.PPE_out = this.out_TKEa_S;
            this.watOutHS.copyAllState( this.watInHS );//20130110
            this.message.append( "(C)負荷率＜停止負荷率で停止");
            this.mState = 3;//能力＝０→換気モードとする20130415
            break;

         case 1://  1_下限入力値固定
            break;

         case 2://  2_下限COP値固定
            break;

         case 3://  3_下限入力値と中間切片
            break;

         default:
         }

   }
      //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
   }



   private double WB_cal(double ddb,double xx){
```

```
        double wwb,x1,de=1;
        int kk=-1,j=0;
        wwb=ddb;

        do{
           j++;
           x1=Psychrometrics.FNXtw(ddb,wwb);
           if(Math.abs(x1-xx)<0.000003) {
              //          System.out.println(" j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
              return wwb;
           }
           if(x1<xx) {
              if(kk==-1) de=de/2.0;
              wwb=wwb+de;
              kk=1;
           }
              if(x1>=xx) {
              if(kk==1) de=de/2.0;
              wwb=wwb-de;
              kk=-1;
           }

        }while(j<1000);
        //    System.out.println("*j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
        return wwb;

     }



     private void cooling_cal_In(){
        //室側の冷房計算
        double heatCapacity=0.0;
        double rCode0 = 0.0,rCode1 = 0.0,rCode2 = 0.0;
        double IAin,IAout1,DBout1,XGout1;
        double S_heat,T_heat;
        double HEXrate;

        HEXrate = this.in_EX_S;

        //バイパス制御 追加 エンタルピで判断
        if( this.isHexbypass == true ){  //20101212nino
           //全熱交換器バイパス有
           this.IAra = Psychrometrics.FNH( this.DBra, this.XGra );//20130109
           this.IAoa = Psychrometrics.FNH( this.DBoa, this.XGoa );

           if( this.IAra > this.IAoa && this.PIDrate.getValue() > 0.001 && this.HEXselect==1 ){//20130109
              //エンタルピ基準
              HEXrate = 0.0;
           }
           if( this.DBra > this.DBoa && this.PIDrate.getValue() > 0.001 && this.HEXselect==2 ){//20130109
              //温度/顕熱基準
              HEXrate = 0.0;
           }
        }
        if( Airswc.isON( this.swcInOA )){//20101212nino
           //吸込み状態を求める
           this.DBin = (( this.in_Va_S - this.in_OA_S ) * this.DBra + (( this.DBoa - this.DBra ) * ( 1.0 - HEXrate ) +
this.DBra ) * this.in_OA_S ) / this.in_Va_S;//20130109
           this.XGin = (( this.in_Va_S - this.in_OA_S ) * this.XGra + (( this.XGoa - this.XGra ) * ( 1.0 - HEXrate ) +
this.XGra ) * this.in_OA_S ) / this.in_Va_S;//20130109
        }else{//20101212nino
           DBin = DBra;//20130109
           XGin = XGra;//20130109
        }
        this.WBin = Psychrometrics.FNWbtx( this.DBin, this.XGin );

        // System.out.println("C/DBin="+DBin+" "+XGin+"  ");
        // System.out.println("C/DBoa="+DBoa+" "+XGoa+"  ");
        // System.out.println("C/DBra="+DBra+" "+XGra+"  ");
        // System.out.println("  ");
        // System.out.println("DBra="+DBra+" "+XGra+"  ");
```

```
        //上下限のチェック 20120821nino
        double twatIn = this.t_watInHS;
        double wbIn = this.WBin;

        if( twatIn > this.rangeMaxCOOLING_T_watIn ){
           //熱源水入口温度＞上限の時停止
           this.message.append( "(C)熱源水入口温度＞上限"+this.rangeMaxCOOLING_T_watIn+"→停止");
           this.S_Load = 0.0;
           this.T_Load = 0.0;
           this.Sc_Load = 0.0;
           this.Tc_Load = 0.0;
           this.Sh_Load = 0.0;
           this.Th_Load = 0.0;
           //
           this.mState = 3;//能力＝０→換気モードとする20130415
           return;
        }
        if( twatIn < this.rangeMinCOOLING_T_watIn ){
           //熱源水入口温度＜下限の時　下限値の特性で運転
           this.message.append( "(C)熱源水入口温度＜下限"+this.rangeMinCOOLING_T_watIn+"→下限値で特性計算");
           twatIn = this.rangeMinCOOLING_T_watIn;
        }
        if( wbIn > this.rangeMaxCOOLING_wbIn ){
           //室内湿球温度＞上限の時　上限値の特性で運転
           this.message.append( "(C)室WB＞上限"+this.rangeMaxCOOLING_wbIn+"→上限値で特性計算");
           wbIn = this.rangeMaxCOOLING_wbIn;
        }
        if( wbIn < this.rangeMinCOOLING_wbIn ){
           //室内湿球温度＜下限の時　下限値の特性で運転
           this.message.append( "(C)室WB＜下限"+this.rangeMinCOOLING_wbIn+"→下限値で特性計算");
           wbIn = rangeMinCOOLING_wbIn;
        }


        //能力補正
        rCode0 = this.pacDB.getKctw( twatIn );//水温補正
        rCode1 = this.pacDB.getKcf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
        rCode2 = this.pacDB.getKcti(wbIn );//室温補正

        //heatCapacity = rCode0 * rCode1 * rCode2 * this.in_Qc_S;
        heatCapacity = rCode0 * rCode1 * rCode2 * this.in_Qc_S * this.PIDrate.getValue();

        //if( twatIn==45 || twatIn==35 ){
        // System.out.println( "室内機能力 TwatIn="+twatIn+" wbIn="+wbIn+" rCode0="+rCode0+" rCode1="+rCode1+"
rCode2="+rCode2+" heatCapacity="+heatCapacity);
        //}

        if( heatCapacity < 0 ){
           heatCapacity = 0;
        }
        // System.out.println("heatCapacity ="+rCode1 * rCode2 +" "+heatCapacity+"  ");

        IAin   = Psychrometrics.FNH(this.DBin, this.XGin);
        IAout1 = IAin - heatCapacity / this.in_Va_S * 1000.0;
        DBout1 = Psychrometrics.FNDbrh( 90.0, IAout1 );//乾球温度
        XGout1 = Psychrometrics.FNxtr( DBout1, 90.0);//絶対湿度
        // System.out.println("IAra="+IAra+" "+IAout1+"  "+MHP_in_Vc_S);

        if( this.XGin < XGout1 ){
           XGout1 = this.XGin;
           DBout1 = Psychrometrics.FNDbxh( XGout1, IAout1 );
           S_heat = heatCapacity;
        }else{
           S_heat = ( this.DBin - DBout1 ) * this.in_Va_S * 1.006;
           if( S_heat > heatCapacity ){
              S_heat = heatCapacity;
           }
        }

        if( Math.abs( this.XGin - XGout1 ) < 0.00001 ){
           T_heat = S_heat;
```

```
    }else{
       T_heat = heatCapacity;
    }

    this.T_C_heat = T_heat;

    //this.S_Load = this.PIDrate.getValue() * S_heat;//室要求顕熱負荷
    //this.T_Load = this.PIDrate.getValue() * T_heat;//室要求全熱負荷
    this.S_Load = S_heat;//室要求顕熱負荷
    this.T_Load = T_heat;//室要求全熱負荷
    this.Sc_Load = this.S_Load;
    this.Tc_Load = this.T_Load;
    this.Sh_Load = 0.0;
    this.Th_Load = 0.0;

    if( this.PIDrate.getValue() < this.in_run_stop_Num ){
       switch( this.num_calcTypeLowerRangeLoad){
       case 0://  0_発停運転
          this.S_Load = 0.0;
          this.T_Load = 0.0;
          this.Sc_Load = 0.0;
          this.Tc_Load = 0.0;
          this.Sh_Load = 0.0;
          this.Th_Load = 0.0;
          this.message.append( "(C)負荷率＜停止負荷率で停止");
          this.mState = 3;//能力＝０→換気モードとする20130415
          //
          this.PIDrate.setValue( 0. );
          break;

       case 1://  1_下限入力値固定
          break;

       case 2://  2_下限COP値固定
          break;

       case 3://  3_下限入力値と中間切片
          break;

       default:
       }
    }

    // System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate);
}



private void heating_cal_In(){
    //室側の暖房計算
    double heatCapacity=0.0;
    double rCode0 = 0.0,  rCode1 = 0.0,rCode2 = 0.0;
    //double IAin,IAout1,DBout1,XGout1;
    double S_heat;
    double HEXrate;

    HEXrate=in_EX_S;

    //バイパス制御 追加 20101212nino
    if( this.isHexbypass == true ){
       this.IAra = Psychrometrics.FNH( this.DBra,  this.XGra );//20130109
       this.IAoa = Psychrometrics.FNH( this.DBoa,  this.XGoa );
       if( this.IAra < this.IAoa && this.PIDrate.getValue() > 0.001 && this.HEXselect == 1 ){//20130109
          HEXrate = 0.0;
       }
       if( this.DBra < this.DBoa && this.PIDrate.getValue() > 0.001 && this.HEXselect == 2 ){//20130109
          HEXrate = 0.0;
       }
       if( this.IAra > this.IAoa && this.PIDrate.getValue() < 0.001 && this.HEXselect == 1 ){//20130109
          HEXrate = 0.0;
       }
       if( this.DBra > this.DBoa && this.PIDrate.getValue() < 0.001 && this.HEXselect == 2 ){//20130109
```

```
            HEXrate = 0.0;
         }
      }
      if( Airswc.isON( this.swcInOA )){//20101212nino
         this.DBin = (( this.in_Va_S - this.in_OA_S ) * this.DBra + ( this.DBra - ( this.DBra - this.DBoa ) * ( 1.0 -
HEXrate )) * this.in_OA_S ) / this.in_Va_S;//20130109
         this.XGin = (( this.in_Va_S - this.in_OA_S ) * this.XGra + ( this.XGra - ( this.XGra - this.XGoa ) * ( 1.0 -
HEXrate )) * this.in_OA_S ) / this.in_Va_S;//20130109
      }else{//20101212nino
         this.DBin = this.DBra;//20130109
         this.XGin = this.XGra;//20130109
      }
      this.WBin = Psychrometrics.FNWbtx(this.DBin,this.XGin);//20130109

      // System.out.println("H/DBin="+DBin+" "+XGin+"  ");
      // System.out.println("H/DBoa="+DBoa+" "+XGoa+"  ");
      // System.out.println("H/DBra="+DBra+" "+XGra+"  ");
      // System.out.println("  ");

      //上下限のチェック 20120821nino
      double twatIn = this.t_watInHS;
      double dbIn = this.DBin;

      if( twatIn > this.rangeMaxHEATING_T_watIn ){
         //熱源水入口温度＞上限の時　上限値の特性で運転
         this.message.append( "(C)熱源水入口温度＞上限"+this.rangeMaxHEATING_T_watIn+"→上限値で特性計算");
         twatIn = this.rangeMaxHEATING_T_watIn;
      }
      if( twatIn < this.rangeMinHEATING_T_watIn ){
         //熱源水入口温度＜下限の時　停止
         this.message.append( "(C)熱源水入口温度＜下限"+this.rangeMinHEATING_T_watIn+"→停止");
         this.S_Load = 0.0;
         this.T_Load = 0.0;
         this.Sc_Load = 0.0;
         this.Tc_Load = 0.0;
         this.Sh_Load = 0.0;
         this.Th_Load = 0.0;
         //
         this.mState = 3;//能力＝０→換気モードとする20130415
         return;
      }
      if( dbIn > this.rangeMaxHEATING_dbRA ){
         //室内乾球温度＞上限の時　上限値の特性で運転
         this.message.append( "(C)室DB＞上限"+this.rangeMaxHEATING_dbRA+"→上限値で特性計算");
         dbIn = this.rangeMaxHEATING_dbRA;
      }
      if( dbIn < this.rangeMinHEATING_dbRA ){
         //室内乾球温度＜下限の時　下限値の特性で運転
         this.message.append( "(C)室DB＜下限"+this.rangeMinHEATING_dbRA+"→下限値で特性計算");
         dbIn = this.rangeMinHEATING_dbRA;
      }


      //能力補正
      rCode0 = this.pacDB.getKhtw( twatIn );//水温補正
      rCode1 = this.pacDB.getKhf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
      rCode2 = this.pacDB.getKhti( dbIn );//室温補正

      //heatCapacity = rCode0 * rCode1 * rCode2 * this.in_Qh_S;
      heatCapacity = rCode0 * rCode1 * rCode2 * this.in_Qh_S * this.PIDrate.getValue();
      if( heatCapacity < 0 ){
         heatCapacity = 0;
      }

      //if( twatIn==45 || twatIn==35 ){
      // System.out.println( "室内機能力 TwatIn="+twatIn+" dbIn="+dbIn+" rCode0="+rCode0+" rCode1="+rCode1+"
rCode2="+rCode2+" heatCapacity="+heatCapacity);
      //}
      // System.out.println("heatCapacity ="+rCode1+" "+rCode2+" "+heatCapacity+"  ");

      //IAin   = Psychrometrics.FNH( this.DBin, this.XGin );
      //IAout1 = IAin + heatCapacity / this.in_Va_S * 1000.0;
```

```
      //XGout1 = this.XGin;
      //DBout1 = Psychrometrics.FNDbxh( XGout1, IAout1 );
      S_heat = heatCapacity;
      //this.S_Load = this.PIDrate.getValue() * S_heat;
      //this.T_Load = this.PIDrate.getValue() * heatCapacity;
      this.S_Load = S_heat;
      this.T_Load = heatCapacity;
      this.Sc_Load = 0.0;
      this.Tc_Load = 0.0;
      this.Sh_Load = this.S_Load;
      this.Th_Load = this.T_Load;

      if( this.PIDrate.getValue() < this.in_run_stop_Num ){
         switch( this.num_calcTypeLowerRangeLoad){
         case 0://  0_発停運転
            this.S_Load = 0.0;
            this.T_Load = 0.0;
            this.Sc_Load = 0.0;
            this.Tc_Load = 0.0;
            this.Sh_Load = 0.0;
            this.Th_Load = 0.0;
            this.message.append( "(C)負荷率＜停止負荷率で停止");
            this.mState = 3;//能力＝０→換気モードとする20130415
            //
            this.PIDrate.setValue( 0. );
            break;

         case 1://  1_下限入力値固定
            break;

         case 2://  2_下限COP値固定
            break;

         case 3://  3_下限入力値と中間切片
            break;

         default:
         }
      }

      // System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate+" DBout="+DBout1);

   }

   private void ventilation_cal(){
      //double heatCapacity=0.0,rCode1 = 0.0,rCode2 = 0.0,rCode3 = 0.0,rCode4 = 0.0,rCode5 = 0.0,rCode6 = 0.0,rCode7 =
0.0,rCode8 = 0.0;
      //double IAra,IAout1,DBout1,XGout1,S_heat,T_heat;
      double HEXrate;

      HEXrate = this.in_EX_S;

      //換気の時は全熱交換器は停止とする
      HEXrate = 0.0;

      //外気と室空気の混合空気の計算
      if( Airswc.isON( this.swcInOA )){//20101212nino
         this.DBin = (( this.in_Va_S - this.in_OA_S ) * this.DBra + (( this.DBoa - this.DBra ) * ( 1.0 - HEXrate ) +
this.DBra ) * this.in_OA_S ) / this.in_Va_S;
         this.XGin = (( this.in_Va_S - this.in_OA_S ) * this.XGra + (( this.XGoa - this.XGra ) * ( 1.0 - HEXrate ) +
this.XGra ) * this.in_OA_S ) / this.in_Va_S;
      }else{//20101212nino
         DBin = DBra;
         XGin = XGra;
      }
      this.WBin = Psychrometrics.FNWbtx( this.DBin, this.XGin );
      //    System.out.println("C/DBin="+DBin+" "+XGin+"  ");
      //    System.out.println("C/DBoa="+DBoa+" "+XGoa+"  ");
      //    System.out.println("C/DBra="+DBra+" "+XGra+"  ");
      //    System.out.println("  ");

      //    System.out.println("DBra="+DBra+" "+XGra+"  ");
```

```
        //this.LPrate = 1.0;

        this.T_C_heat = 0.0;

        this.S_Load = 0.0;
        this.T_Load = 0.0;
        this.Sc_Load = 0.0;
        this.Tc_Load = 0.0;
        this.Sh_Load = 0.0;
        this.Th_Load = 0.0;

        // System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate.getValue());
    }


    private void cooling_supply(){
        /*
           double IAra,IAout;
           IAra=Psychrometrics.FNH(DBra,XGra);
           IAout=IAra-T_Load_RM/in_Va_S*1000.0;
           DBout=DBra-S_Load_RM/in_Va_S;//乾球温度
           XGout= Psychrometrics.FNXth(DBout, IAout);//絶対湿度
         */
        //******************090725-s
        double IAout,sRc;
        double SS_rm;

        IAin = Psychrometrics.FNH( this.DBin, this.XGin );
        if( this.T_C_heat > 0.0 ){
           sRc = this.Tc_Load_RM / this.T_C_heat;
        }else{
           sRc = 0.0;
        }
        IAout = IAin - this.Tc_Load_RM / this.in_Va_S * 1000.0;
        SS_rm = this.Sc_Load_RM + ( this.Tc_Load_RM - this.Sc_Load_RM ) * ( 1.0 - sRc ) * this.saijohatsu / 100.0;
        if( SS_rm > this.Tc_Load_RM ){
           SS_rm = this.Tc_Load_RM;
        }
        this.DBout = this.DBin - SS_rm / this.in_Va_S / 1.006;//乾球温度
        this.XGout = Psychrometrics.FNXth( this.DBout, IAout);//絶対湿度

        /*
           if(saijohatsu>0.0 && Psychrometrics.FNRhtx(DBin, XGin)<40.0){
              System.out.println("40以下"+(++u40)+"S_Load_RM="+S_Load_RM+" T_Load=_RM"+T_Load_RM+" sRc="+sRc+"
SS_rm="+SS_rm);
              System.out.println("   DBout0="+(DBra-S_Load_RM/in_Va_S)+"  DBout1="+(DBra-SS_rm/in_Va_S));
              System.out.println("   XXout0="+Psychrometrics.FNXth((DBra-S_Load_RM/in_Va_S), IAout)+"
XXout1="+Psychrometrics.FNXth((DBra-SS_rm/in_Va_S), IAout));
              System.out.println("   DBin  ="+DBin+"  XGin  ="+XGin+"  RHin  ="+Psychrometrics.FNRhtx(DBin, XGin));
              System.out.println("   DBoa  ="+DBoa+"  XGoa  ="+XGoa+"  RHoa  ="+Psychrometrics.FNRhtx(DBoa, XGoa));
              System.out.println("   DBra  ="+DBra+"  XGra  ="+XGra+"  RHra  ="+Psychrometrics.FNRhtx(DBra, XGra));

           }
         */
        //******************090725-e
        //    **********************************************************************
        //    System.out.println("DBout="+DBout+" "+XGout+"  ");
    }



    private void heating_supply(){
        double IAout;

        this.RHin = Psychrometrics.FNRhtx( this.DBin, this.XGin );
        this.Sh_Load_RM = this.CAPrateH * this.Sh_Load;
        this.Th_Load_RM = this.CAPrateH * this.Th_Load;

        if( this.in_HUM_on <= this.RHin || this.in_HUM_mx < 0.00001 ){
           IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
           IAout = IAin +this.Th_Load_RM / this.in_Va_S * 1000.0;
```

```
            this.DBout = this.DBin + this.Sh_Load_RM / this.in_Va_S / 1.006;//乾球温度
            this.XGout = Psychrometrics.FNXth( this.DBout, IAout );//絶対湿度
            this.CW_Wrate = 0.0;
            //    System.out.println("1 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
            //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
        } else {
            IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
            IAout = IAin + this.Th_Load_RM / this.in_Va_S * 1000.0;
            this.DBout = Psychrometrics.FNDbrh( this.in_HUM_rt, IAout);
            this.XGout = Psychrometrics.FNxtr( this.DBout, this.in_HUM_rt);
            this.CW_Wrate = ( this.XGout - this.XGin ) * this.GW;

            //定格加湿量をオーバーするときの処理
            if( this.CW_Wrate > this.in_HUM_mx ){
               this.XGout = this.in_HUM_mx / this.GW+ this.XGin;
               this.DBout =Psychrometrics.FNDbxh( this.XGout, IAout);
               this.CW_Wrate = this.in_HUM_mx;
            }
            if( this.XGout < this.XGin){
               IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
               IAout = IAin + this.Th_Load_RM / this.in_Va_S * 1000.0;
               this.DBout = this.DBin + this.Sh_Load_RM / this.in_Va_S / 1.006;//乾球温度
               this.XGout = Psychrometrics.FNXth( this.DBout, IAout);//絶対湿度
               this.CW_Wrate = 0.0;
            }
            //    System.out.println("2 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
            //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
        }
        //    this.watInCW.setFlowRate(CW_Wrate);
    }


    private void ventilation_supply(){

        this.Sh_Load_RM = 0.0;
        this.Th_Load_RM = 0.0;

        this.DBout = this.DBin;//乾球温度
        this.XGout = this.XGin;//絶対湿度
        this.CW_Wrate = 0.0;

    }

/*
    private double WB_cal(double ddb,double xx){
        double wwb,x1,de=1;
        int i,kk=-1,j=0;
        wwb=ddb;

        do{
            j++;
            x1=Psychrometrics.FNXtw(ddb,wwb);
            if(Math.abs(x1-xx)<0.000003) {
                //          System.out.println(" j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
                return wwb;
            }
            if(x1<xx) {
                if(kk==-1) de=de/2.0;
                wwb=wwb+de;
                kk=1;
            }
                if(x1>=xx) {
                if(kk==1) de=de/2.0;
                wwb=wwb-de;
                kk=-1;
            }

        }while(j<1000);
        //    System.out.println("*j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
        return wwb;
    }
*/
```

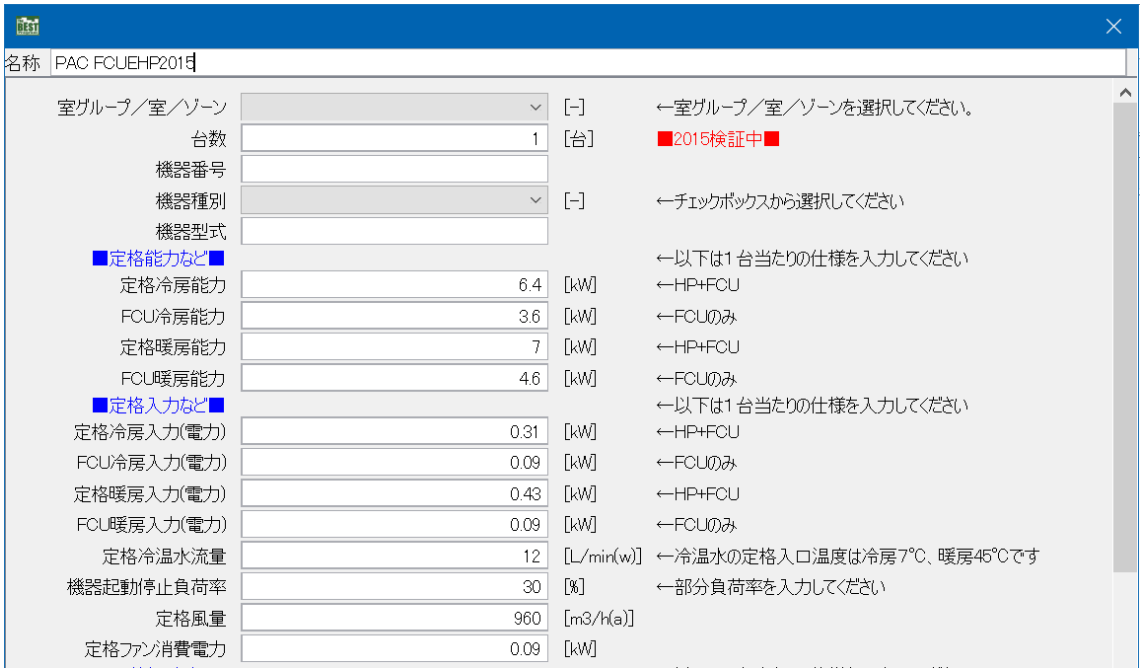
```
}
```

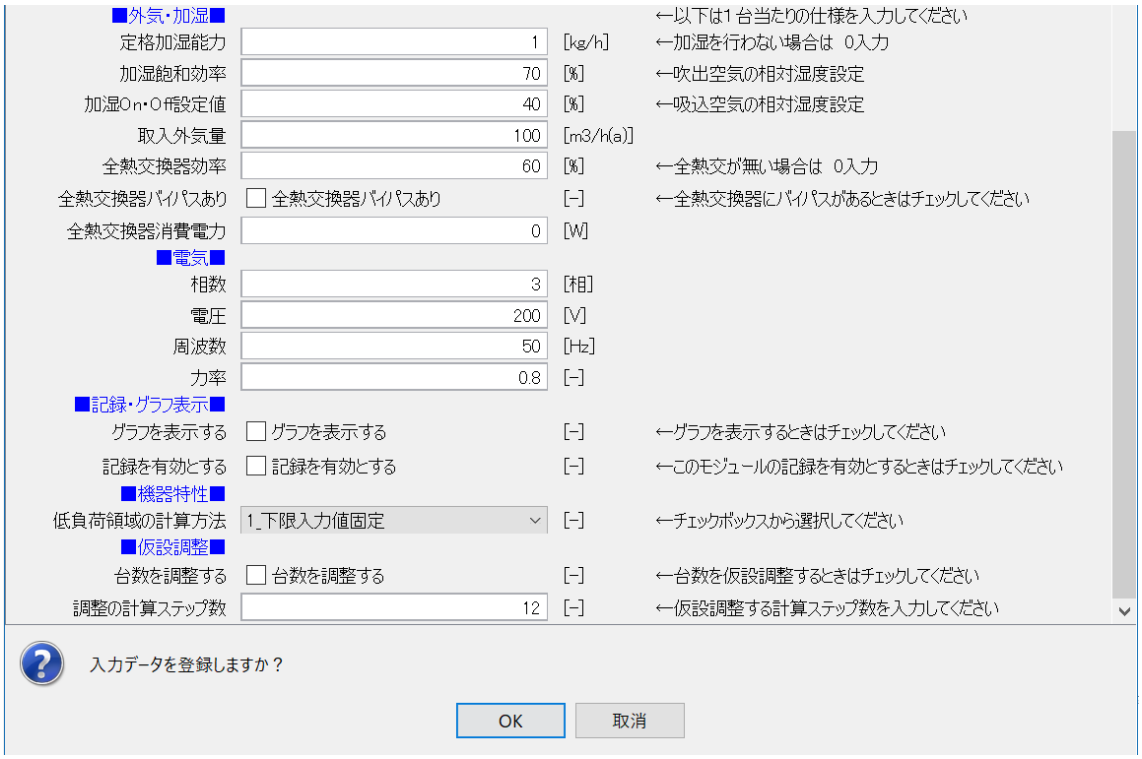
「PAC FCUEHP2015」（場所：設備 2015／個別分散 2015／）

モジュール名	PAC FCUEHP2015
クラス	FCUwithEHPModule2015

（1）入力画面

・スペック





（2）モジュールの概要

FCU＋水冷 PAC タイプモジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

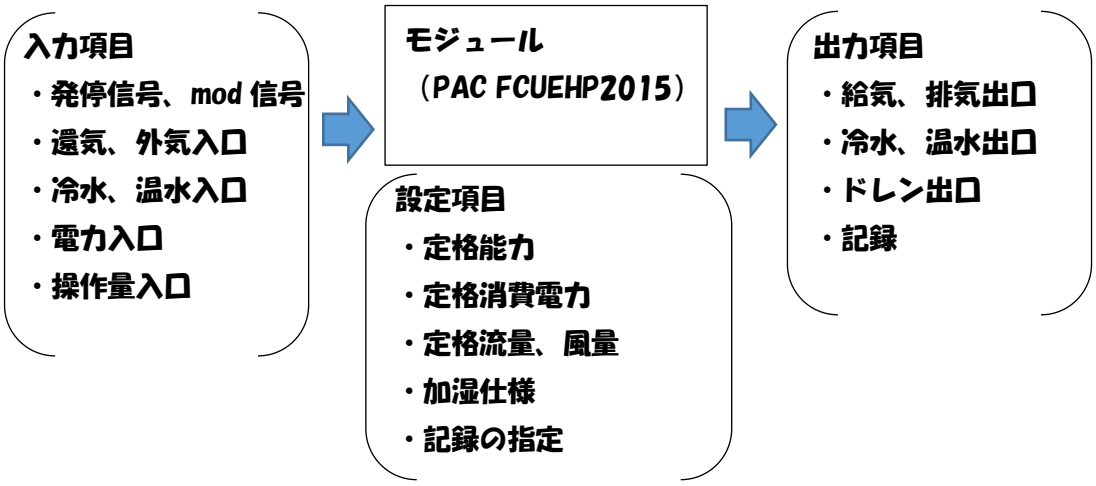


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	室グループ／室／ゾーン	String	zoneAirforSSinside	#RoomGroup	[-]	ー	ー		←室グループ／室／ゾーンを選択してください。
2	台数	double	in_dai	1	[台]	ー	0		
3	機器番号	String	m_no			ー	ー		
4	機器種別	String	m_ty		[-]	ー	ー		←チェックボックスから選択してください
5	機器型式	String	m_kt			ー	ー		
6	■定格能力など■					ー	ー		←以下は１台当たりの仕様を入力してください
7	定格冷房能力	double	in_Qc_S	6.4	[kW]	ー	0		←HP+FCU
8	FCU 冷房能力	double	in_Qc_FCU	3.6	[kW]	ー	0		←FCU のみ
9	定格暖房能力	double	in_Qh_S	7	[kW]	ー	0		←HP+FCU
10	FCU 暖房能力	double	in_Qh_FCU	4.6	[kW]	ー	0		←FCU のみ
11	■定格入力など■					ー	ー		←以下は１台当たりの仕様を入力してください
12	定格冷房入力(電力)	double	in_PPEc_S	0.31	[kW]	ー	0		←HP+FCU
13	FCU 冷房入力(電力)	double	in_PPEc_FCU	0.09	[kW]	ー	0		←FCU のみ
14	定格暖房入力(電力)	double	in_PPEh_S	0.43	[kW]	ー	0		←HP+FCU
15	FCU 暖房入力(電力)	double	in_PPEh_FCU	0.09	[kW]	ー	0		←FCU のみ
16	定格冷温水流量	double	de_fRate_watInHS	12	[L/min(w)]	ー	0		←冷温水の定格入口温度は冷房７℃、暖房 45℃です
17	機器起動停止負荷率	double	in_run_stop	30	[%]	100	0		←部分負荷率を入力してください
18	定格風量	double	in_Va_S	960	[m3/h(a)]	ー	0		

19	定格ファン消費電力	double	in_PPEa_S	0.09	[kW]	―	0		
20	■外気・加湿■					―	―		←以下は１台当たりの仕様を入力してください
21	定格加湿能力	double	in_HUM_mx	1	[kg/h]	―	0		←加湿を行わない場合は　0 入力
22	加湿飽和効率	double	in_HUM_rt	70	[%]	―	0		←吹出空気の相対湿度設定
23	加湿 On・Off 設定値	double	in_HUM_on	40	[%]	―	0		←吸込空気の相対湿度設定
24	取入外気量	double	in_OA_S	100	[m3/h(a)]	―	0		
25	全熱交換器効率	double	in_EX_S	60	[%]	100	0		←全熱交が無い場合は　0 入力
26	全熱交換器バイパスあり	boolean	isHexbypass	FALSE	[-]	―	―		←全熱交換器にバイパスがあるときはチェックしてください
27	全熱交換器消費電力	double	in_EX_PE	0	[W]	―	0		
28	■電気■					―	―		
29	相数	int	phase	3	[相]	―	1		
30	電圧	double	voltage	200	[V]	―	100		
31	周波数	double	frequency	50	[Hz]	60	50		
32	力率	double	powerFactor	0.8	[-]	1	0		
33	■記録・グラフ表示■					―	―		
34	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
35	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください
36	■機器特性■					―	―		
37	低負荷領域の計算方法	String	calcTypeLowerRangeLoad	0_発停運転、1_下限入力値固定、2_下限COP値固定、3_下限入力	[-]	―	―		←チェックボックスから選択してください

				値と中間切片					
38	■仮設調整■					—	—		
39	台数を調整する	boolean	isAdjust2012	FALSE	[-]	—	—		←台数を仮設調整するときはチェックしてください
40	調整の計算ステップ数	int	numAdjustSteps	12	[-]	—	1		←仮設調整する計算ステップ数を入力してください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	外気導入信号入口	L1_swcInOA	swcInOA	-	制御	OnOff 信号	入口	
4	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
5	PID モード信号入口	L1_modInPID	modInPID	-	制御	制御モード	入口	
6	還気入口	LO_airInRA	airInRA	-	状態	空気	入口	
7	給気出口	LO_airOutSA	airOutSA	-	状態	空気	出口	
8	外気入口	LO_airInOA	airInOA	-	状態	空気	入口	
9	排気出口	LO_airOutEA	airOutEA	-	状態	空気	出口	
10	冷水入口	LO_watInC	watInC	-	状態	水	入口	
11	冷水出口	LO_watOutC	watOutC	-	状態	水	出口	
12	温水入口	LO_watInH	watInH	-	状態	水	入口	
13	温水出口	LO_watOutH	watOutH	-	状態	水	出口	
14	給水入口	LO_watInCW	watInCW	-	状態	水	入口	
15	ドレン出口	LO_watOutD	watOutD	-	状態	水	出口	
16	操作量入口	LO_valInCtrl	valInCtrl	-	値	値	入口	
17	電力入口	LO_eleIn	eleIn		状態	電気	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	FCUHP_Message#-#-	メッセージ	—	メッセージ
2	FCUHP_処理顕熱量合計#W#熱量	FCUHP_処理顕熱量合計#W#熱量	W	My
3	FCUHP_処理全熱量合計#W#熱量	FCUHP_処理全熱量合計#W#熱量	W	My
4	FCUHP_冷却要求全熱量合計#W#熱量	FCUHP_冷却要求全熱量合計#W#熱量	W	My
5	FCUHP_加熱要求全熱量合計#W#熱量	FCUHP_加熱要求全熱量合計#W#熱量	W	My
6	FCUHP_消費電力#W#消費電力	FCUHP_消費電力#W#消費電力	W	エネルギー消費
7	FCUHP_冷却可能熱量/最大#W#熱量	FCUHP_冷却可能熱量/最大#W#熱量	W	My
8	FCUHP_加熱可能熱量/最大#W#熱量	FCUHP_加熱可能熱量/最大#W#熱量	W	My
9	FCUHP_処理全熱量合計負荷#W#-	FCUHP_処理全熱量合計負荷#W#-	W	負荷
10	FCUHP_PID 操作量#-#-	FCUHP_PID 操作量#-#-	—	入口
11	FCUHP_watInC 温度#℃#温度	FCUHP_watInC 温度#℃#温度	℃	入口
12	FCUHP_watInH 温度#℃#温度	FCUHP_watInH 温度#℃#温度	℃	入口
13	FCUHP_watOutC 温度#℃#温度	FCUHP_watOutC 温度#℃#温度	℃	出口
14	FCUHP_watOutH 温度#℃#温度	FCUHP_watOutH 温度#℃#温度	℃	出口
15	FCUHP_watOutC 流量#g/s#質量流量	FCUHP_watOutC 流量#g/s#質量流量	g/s	出口
16	FCUHP_watOutH 流量#g/s#質量流量	FCUHP_watOutH 流量#g/s#質量流量	g/s	出口
17	FCUHP_COP#-#COP	FCUHP_COP#-#COP	—	My
18	FCUHP_室吹出乾球温度#℃#温度	FCUHP_室吹出乾球温度#℃#温度	℃	出口
19	FCUHP_室吹出絶対湿度#g/g#湿度	FCUHP_室吹出絶対湿度#g/g#湿度	g/g	出口
20	FCUHP_室吹出風量#g/s#質量流量	FCUHP_室吹出風量#g/s#質量流量	g/s	出口
21	FCUHP_ドレン量#g/s#質量流量	FCUHP_ドレン量#g/s#質量流量	g/s	出口
22	FCUHP_加湿給水量#g/s#質量流量	FCUHP_加湿給水量#g/s#質量流量	g/s	入口
23	FCUHP_吸込乾球温度#℃#温度	FCUHP_吸込乾球温度#℃#温度	℃	入口
24	FCUHP_吸込絶対湿度#g/g#湿度	FCUHP_吸込絶対湿度#g/g#湿度	g/g	入口
25	FCUHP_室乾球温度#℃#温度	FCUHP_室乾球温度#℃#温度	℃	入口
26	FCUHP_室絶対湿度#g/g#湿度	FCUHP_室絶対湿度#g/g#湿度	g/g	入口
27	FCUHP_調整冷却能力#W#熱量	FCUHP_調整冷却能力#W#熱量	W	調整
28	FCUHP_調整加熱能力#W#熱量	FCUHP_調整加熱能力#W#熱量	W	調整
29	FCUHP_調整冷却台数#-#台数	FCUHP_調整冷却台数#-#台数	—	調整
30	FCUHP_調整加熱台数#-#台数	FCUHP_調整加熱台数#-#台数	—	調整
31	FCUHP_調整ステップ平均冷却台数#-#台数	FCUHP_調整ステップ平均冷却台数#-#台数	—	調整

32	FCUHP_調整ステップ平均加熱台数#-#台数	FCUHP_調整ステップ平均加熱台数#-#台数	—	調整
33				

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.essentials.utility.Psychrometrics;
import jp.or.ibec.best.log.BestLogger;
import jp.or.ibec.best.message.BestEngineMessageHandler;
import jp.or.ibec.best.message.BestEngineMessageParam;
import jp.or.ibec.best.message.constants.SystemMessageConstants;

/**
 * @author HIROSHI NINOMIYA/201508
 *
 * FCU+水冷EHP(PAFMAC)
 *
 * 201508
 *
 */
public class FCUwithEHPModule2015 extends AbstractBestModule implements
IBestMetaModule  {

   /**
    * 論理アドレス
    */
   private final String moduleName="(FCUwithEHPModule2015)";

   private final String S_NODE_airInOA  = "L0_airInOA";//外気
   private final String S_NODE_airOutEA = "L0_airOutEA";//外気
   private final String S_NODE_watInC   = "L0_watInC";   //FCU入口冷水
   private final String S_NODE_watInH   = "L0_watInH";   //FCU入口温水
   private final String S_NODE_watOutC  = "L0_watOutC";//FCU出口冷水
   private final String S_NODE_watOutH  = "L0_watOutH";//FCU出口温水
// private final String S_NODE_watIn    = "L0_watInHS";//熱源水入口
// private final String S_NODE_watOut   = "L0_watOutHS";//熱源水出口

   private final String S_NODE_airInRA  = "L0_airInRA";//室内吸込口
   private final String S_NODE_airOutSA = "L0_airOutSA";//室内吹出し

   private final String S_NODE_watOutD  = "L0_watOutD";//コイル出口ドレン
   private final String S_NODE_watInCW  = "L0_watInCW";//watInCW---090407追加**************

   private final String S_NODE_valInPID = "L0_valInCtrl";//供給熱量 from PID           : in

   private final String S_NODE_eleIn    = "L0_eleIn";//電力

   private final String C_NODE_swcIn    = "L1_swcIn";//運転状態：off/on
   private final String C_NODE_swcInOA  = "L1_swcInOA";//外気運転状態：off/on
   private final String C_NODE_modIn    = "L1_modIn";//空調モード：停止/冷却/加熱
   private final String C_NODE_modInPID = "L1_modInPID";//空調モード：停止/冷却/加熱
   private final String C_NODE_modInPIPE= "L1_modInPIPE";//冷水／温水切替

   private final String R_NODE = "L2_recOut";
```

```
    /**
     * 外部定義
     */
    private final String SPEC_name          = "名称";

    private final String SPEC_grzName  = "室グループ／室／ゾーン";

    private final String SPEC_m_no          = "機器番号";
    private final String SPEC_m_ty          = "機器種別";
    private final String SPEC_m_kt          = "機器型式";
    private final String SPEC_in_Qc_S       = "定格冷房能力[W]";
    private final String SPEC_in_Qc_FCU     = "FCU冷房能力[W]";
    private final String SPEC_in_Qh_S       = "定格暖房能力[W]";
    private final String SPEC_in_Qh_FCU     = "FCU暖房能力[W]";
//  private final String SPEC_out_Qh_L      = "低温暖房能力[W]";//kW
    private final String SPEC_in_PPEc_S     = "定格冷房入力(電力)[W]";
    private final String SPEC_in_PPEc_FCU   = "FCU冷房入力(電力)[W]";
    private final String SPEC_in_PPEh_S     = "定格暖房入力(電力)[W]";
    private final String SPEC_in_PPEh_FCU   = "FCU暖房入力(電力)[W]";
//  private final String SPEC_out_PPEh_L    = "低温暖房入力(電力)[W]";//kW
/*
    private final String SPEC_out_CLEa_R    = "クランクケースヒータ(運転時)";//W
    private final String SPEC_out_CLEa_S    = "クランクケースヒータ(停止時)";//W
    private final String SPEC_out_TKEa_R    = "定格待機電力(運転時)";//W
    private final String SPEC_out_TKEa_T    = "定格待機電力(待機時)";//W
    private final String SPEC_out_TKEa_S    = "定格待機電力(停止時)";//W
*/
    private final String SPEC_de_fRate_watInHS   = "定格冷温水流量[g/s]";

    private final String SPEC_in_run_stop  = "機器起動停止負荷率[-]";//[%]
    private final String SPEC_Phase       = "相数[-]";
    private final String SPEC_Voltage     = "電圧[V]";
    private final String SPEC_Frequency   = "周波数[Hz]";
    private final String SPEC_PowerFactor = "力率[-]";
    //
    private final String SPEC_isGVisible = "グラフを表示する";
    private final String SPEC_maxItemCount = "最大同時表示ステップ数";
    private final String SPEC_isRecord = "記録を有効とする";

    private final String SPEC_CalcTypeLowerRangeLoad = "低負荷領域の計算方法";

    private final String SPEC_isAdjust2012   = "台数を調整する";
    private final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";

    private StringBuffer message= new StringBuffer();

    /**
     * 外部定義
     */

    private final String SPEC_in_dai       = "台数[-]";

    private final String SPEC_in_Va_S         = "定格風量[g/s]";//m3/h
    private final String SPEC_in_PPEa_S    = "定格ファン消費電力[W]";//W
//  private final String SPEC_in_TKEa_R     = "定格待機電力(運転時)";//W
//  private final String SPEC_in_THEa_S     = "定格待機電力(停止時)";//W

    private final String SPEC_in_HUM_mx       = "定格加湿能力[g/s]";//kg/h
    private final String SPEC_in_HUM_rt       = "加湿飽和効率[-]";//[%]
    private final String SPEC_in_HUM_on       = "加湿On・Off設定値[-]";//[%]
    private final String SPEC_in_OA_S         = "取入外気量[g/s]";//m3/h
    private final String SPEC_in_EX_S         = "全熱交換器効率[-]";//[%]
    private final String SPEC_isHexbypass      = "全熱交換器バイパスあり[-]";
    private final String SPEC_in_EX_PE        = "全熱交換器消費電力[W]";


    /**
     * 変数（外部定義に対応）
     */
```

```
    private String name              ;// "機器名称";
    private String m_no              ;//= "機器番号";
    private String m_ty              ;//= "機器種別";
    private String m_kt              ;//= "機器型式";
    private double in_Qc_S      ;//= "定格冷房能力";//kW
    private double in_Qc_FCU    ;//= "FCU冷房能力";//kW
    private double in_PPEc_S    ;//= "定格冷房入力(電力)";//kW
    private double in_PPEc_FCU  ;//= "FCU冷房入力(電力)";//kW
    private double in_Qh_S      ;//= "定格暖房能力";//kW
    private double in_Qh_FCU    ;//= "FCU暖房能力";//kW
// private double out_Qh_L     ;//= "低温暖房能力";//kW
    private double in_PPEh_S    ;//= "定格暖房入力(電力)";//kW
    private double in_PPEh_FCU  ;//= "FCU暖房入力(電力)";//kW
// private double out_PPEh_L    ;//= "低温暖房入力(電力)";//kW
/*
    private double out_CLEa_R=0.0    ;//= "クランクケースヒータ(運転時)";//W
    private double out_CLEa_S=0.0    ;//= "クランクケースヒータ(停止時)";//W
    private double out_TKEa_R =0.0   ;//= "定格待機電力(運転時)";//W
    private double out_TKEa_T =0.0   ;//= "定格待機電力(待機時)";//W
    private double out_TKEa_S =0.0  ;//= "定格待機電力(停止時)";//W
*/
    private double t_watInHS;// = this.watInHS.getTemp();

    private double fRate_watInHS ;//熱源水水量[g/s]
    private double de_fRate_watInHS ;//熱源水定格水量[g/s]

    private double in_run_stop    ;//= "機器起動停止負荷率";//[%]
    private int phase;         //相数[-]
    private double voltage;    //電圧[V]
    private double frequency;  //周波数[Hz]
    private double powerFactor;   //力率[-]

    private boolean isGVisible = false;//このグラフを表示する=true
    private int  maxItemCount = 100;
    private boolean isRecord   = false;//記録を有効とする=true


    /**
     * 変数（外部定義に対応）
     */


    private double in_dai         ;//= "台数";

    private double in_Va_S        ;//= "定格風量";//m3/h
    private double in_PPEa_S      ;//= "定格ファン消費電力";//W
    private double in_TKEa_R=0.0  ;//= "定格待機電力(運転時)";//W
    private double in_TKEa_S=0.0  ;//= "定格待機電力(停止時)";//W

    private double in_HUM_mx          ;//= "定格加湿能力";//kg/h
    private double in_HUM_rt          ;//= "加湿飽和効率";//[%]
    private double in_HUM_on          ;//= "加湿On・Off設定値";//[%]
    private double in_OA_S        ;//= "取入外気量";//m3/h
    private double in_EX_S        ;//= "全熱交換器効率";//[%]
    private boolean isHexbypass = false;//熱交バイパスあり=true
    private double in_EX_PE    ;//= "全熱交換器消費電力[W]";

    private BestAir airInOA  = null;
    private BestAir airOutEA   = null;

    private BestWater watInC  = null;
    private BestWater watOutC = null;
    private BestWater watInH  = null;
    private BestWater watOutH = null;
    private BestWater watInHS = null;
    private BestWater watOutHS = null;

    private BestElectricity eleIn = null;  //電力

    private PACDBManager pacDB=null;

    private int modIn  ;
```

```
    private int modInPID  ;
    private int modInPIPE ;
    private int swcIn ;
    private int swcInOA ;
    private int mState;  //停止、冷房、暖房フラグ



    private BestAir airInRA   = null;
    private BestAir airOutSA  = null;         //給気
    private BestWater watOutD = null;
    private BestWater watInCW = null;

    private BestValue valInPID = null;

  // private PACDBManager pacDB=null;


    /**
     * 出力（建物）
     */
    private final String RECORD_message      = "FCUHP_Message#-#-";
    private final String RECORD_opeLoadS     = "FCUHP_処理顕熱量合計#W#熱量";
    private final String RECORD_opeLoadT     = "FCUHP_処理全熱量合計#W#熱量";
    private final String RECORD_coolingLoadT = "FCUHP_冷却要求全熱量合計#W#熱量";
    private final String RECORD_heatingLoadT = "FCUHP_加熱要求全熱量合計#W#熱量";

    private final String RECORD_ID4        = "FCUHP_消費電力#W#消費電力";
    private final String RECORD_capCooling = "FCUHP_冷却可能熱量/最大#W#熱量";
    private final String RECORD_capHeating = "FCUHP_加熱可能熱量/最大#W#熱量";

    private final String RECORD_qT         = "FCUHP_処理全熱量合計負荷#W#-";
    private final String RECORD_pidVal     = "FCUHP_PID操作量#-#-";

    private final String RECORD_T_watInC   = "FCUHP_watInC温度#℃#温度";
    private final String RECORD_T_watInH   = "FCUHP_watInH温度#℃#温度";
    private final String RECORD_T_watOutC  = "FCUHP_watOutC温度#℃#温度";
    private final String RECORD_T_watOutH  = "FCUHP_watOutH温度#℃#温度";
    private final String RECORD_M_watOutC  = "FCUHP_watOutC流量#g/s#質量流量";
    private final String RECORD_M_watOutH  = "FCUHP_watOutH流量#g/s#質量流量";

    private final String RECORD_COP = "FCUHP_COP#-#COP";


    /**
     * 出力（建物）
     */

    private final String RECORD_DBout     = "FCUHP_室吹出乾球温度#℃#温度";
    private final String RECORD_XGout     = "FCUHP_室吹出絶対湿度#g/g#湿度";
    private final String RECORD_GW        = "FCUHP_室吹出風量#g/s#質量流量";
    private final String RECORD_D_Wrate   = "FCUHP_ドレン量#g/s#質量流量";
    private final String RECORD_ID8i      = "FCUHP_加湿給水量#g/s#質量流量";

    private final String RECORD_DBairIn   = "FCUHP_吸込乾球温度#℃#温度";
    private final String RECORD_XairIn    = "FCUHP_吸込絶対湿度#g/g#湿度";
    private final String RECORD_DBairInRM = "FCUHP_室乾球温度#℃#温度";
    private final String RECORD_XairInRM  = "FCUHP_室絶対湿度#g/g#湿度";



    private final String RECORD_out_Qc_Adjust = "FCUHP_調整冷却能力#W#熱量";
    private final String RECORD_out_Qh_Adjust = "FCUHP_調整加熱能力#W#熱量";
    private final String RECORD_in_daicAdjust = "FCUHP_調整冷却台数#-#台数";
    private final String RECORD_in_daihAdjust = "FCUHP_調整加熱台数#-#台数";
    private final String RECORD_avecdaiAdjust = "FCUHP_調整ステップ平均冷却台数#-#台数";
    private final String RECORD_avehdaiAdjust = "FCUHP_調整ステップ平均加熱台数#-#台数";

    private double DBin;          //室内乾球温度[℃]
    private double WBin;          //室内湿球温度[℃]
    private double XGin;
    private double IAin;
```

```
    private double RHin;

    private double DBoa;          //外気乾球温度[℃]
    private double WBoa;          //外気湿球温度[℃]
    private double XGoa;
    private double IAoa;

    private double DBra;          //室内機吸込乾球温度[℃]
    private double WBra;          //室内機吸込湿球温度[℃]
    private double XGra;
    private double IAra;

    private double DBout;            //乾球温度[℃]
    private double XGout;            //絶対湿度[kg/kg.DA]

    private double GW;               //風量(g/s)
    private double D_Wrate;       //ドレン水量(g/s)
    private double CW_Wrate;         //加湿水量(g/s)

    private double PPE_out;
    private double E_PPE_out;
    private double PPE_in;
    private double E_PPE_in;

    private double opeLoadS;//処理熱量（顕熱）[W]
    private double opeLoadT;//処理熱量（全熱）[W]
    private double opeCoolingLoadS;//処理熱量（顕熱）[W]
    private double opeCoolingLoadT;//処理熱量（全熱）[W]
    private double opeHeatingLoadS;//処理熱量（顕熱）[W]
    private double opeHeatingLoadT;//処理熱量（全熱）[W]
//******************090725-s
    private double saijohatsu;
// private double T_C_heat;
// private int u40=0;

//******************090725-e
    private int dbflg=0;
    private String path="EHP";
    private String[] filenames= new String[1];
    private String equipmentName;
    //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    private int HEXselect=1;//1=エンタルピ基準  2=温度基準
    //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    //グラフ表示など
    private GraphJFrameBuilMultiIn_S20101212 gBMIn = null;


    //低負荷領域の計算方法
    private String calcTypeLowerRangeLoad = null;//"低負荷領域の計算方法";
    private int num_calcTypeLowerRangeLoad;//"低負荷領域の計算方法";

    /**
     * その他変数
     */

    private double Rc;               //部分負荷率
    private double Rh;               //部分負荷率

    private double capCooling;//処理可能熱量（冷却全熱）[W]
    private double capHeating;//処理可能熱量（加熱全熱）[W]

    private double coolingLoadT;//要求処理熱量（冷却全熱）[W]
    private double heatingLoadT;//要求処理熱量（加熱全熱）[W]

    private String kiki_file;

    private double COP;  //成績係数(-)

    //グラフ表示など
    private GraphJFrameBuilMultiOut_S20101212 gBMout = null;
```

```
    //仮設調整モード2012

    private boolean isAdjust2012  = false;//true="台数を調整する";//
    //private boolean isAdjust2012 = false;//20120406nino 建築学会大会空衛学会OS論文検討用＝true
    private int numAdjustSteps;//"調整の計算ステップ数";
    private LinkedList<Double> daiListc = null;//必要台数
    private LinkedList<Double> daiListh = null;//必要台数
    private LinkedList<Double> rcList = null;//必要台数
    private LinkedList<Double> rhList = null;//必要台数
    private double maxcAdjustdai = 1;
    private double maxhAdjustdai = 1;
    private double avecdaiAdjust = 0;
    private double avehdaiAdjust = 0;
    private double in_daicAdjust = 1;//= "調整台数";
    private double in_daihAdjust = 1;//= "調整台数";
    private double in_run_stop_Num;//= "機器起動停止負荷率"台数補正;

// private double max_rcAdjust = 1.;
// private double max_rhAdjust = 1.;
// private double ave_rcAdjust = 0;
// private double ave_rhAdjust = 0;
// private double out_rcAdjust = 1.;//= "調整率冷房";
// private double out_rhAdjust = 1.;//= "調整率暖房";

    private double in_Qc_S_1      ;//= "定格冷房能力";//kW
    private double in_Qc_FCU_1      ;//= "FCU冷房能力";//kW
    private double in_PPEc_S_1    ;//= "定格冷房入力(電力)";//kW
    private double in_PPEc_FCU_1    ;//= "FCU冷房入力(電力)";//kW
    private double in_Qh_S_1      ;//= "定格暖房能力";//kW
    private double in_Qh_FCU_1      ;//= "FCU暖房能力";//kW
// private double out_Qh_L_1      ;//= "低温暖房能力";//kW
    private double in_PPEh_S_1    ;//= "定格暖房入力(電力)";//kW
    private double in_PPEh_FCU_1    ;//= "FCU暖房入力(電力)";//kW
// private double out_PPEh_L_1    ;//= "低温暖房入力(電力)";//kW

// private double out_Qc_Adjust;//= "調整冷却能力";
// private double out_Qh_Adjust;//= "調整加熱能力";

    private double fRate_watInHSc ;//水量[g/s]
    private double fRate_watInHSh ;//水量[g/s]

    private double in_Va_S_1      ;//= "定格風量";//m3/h
    private double in_PPEa_S_1    ;//= "定格ファン消費電力";//W

    private double in_HUM_mx_1    ;//= "定格加湿能力";//kg/h

    private ZoneAirforSSinsideModule201502 zoneAirforSSinside = null;
    private String grzName = null;
    private boolean isUseZoneAir = false;

    private double t_watInHS_UpperLimitC;
    private double t_watInHS_LowerLimitC;
    private double wb_airInRA_UpperLimitC;
    private double wb_airInHS_LowerLimitC;

    private double t_watInHS_UpperLimitH;
    private double t_watInHS_LowerLimitH;
    private double db_airInRA_UpperLimitH;
    private double db_airInRA_LowerLimitH;


    //グラフ表示など
    private GraphCommonJFrame2015 gGCJF = null;
    private GraphCommonData2015[] gData = null;

    private boolean isCPipe = false;//冷水配管有効の時true

    @Override
    public void setProfile(BestSpecs spec) {
//    外部定義項目取得
      if(spec == null) {
```

```
        return;
    }
    Map<String,String> map=spec.getSpec();
    if(map == null) {
        return;
    }

    // 機器名称
    this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

    this.grzName = spec.getSpecValue( this.SPEC_grzName, "" );
    if( !this.grzName.equals( "" )){
        this.isUseZoneAir = true;
    }

    if( this.isUseZoneAir ){
        this.zoneAirforSSinside = new ZoneAirforSSinsideModule201502();
        this.zoneAirforSSinside._setProfile( this.name, this.grzName );
    }

    // 機器番号
//  this.m_no = spec.getSpecValue( this.SPEC_m_no, "" );
    if(null!=map.get(this.SPEC_m_no)){
        this.m_no = (String)map.get(this.SPEC_m_no);
        //******************090725-s
        try{
            if(m_no.substring(0 , 1).equals("+")==true){
                saijohatsu=Double.parseDouble(m_no.substring(1 , 3));
            }else{
                saijohatsu=50.0;
            }
        }
        catch(NumberFormatException e){
            saijohatsu=50.0;
        }
        //******************090725-s
    } else {
        ///System.out.println("(W)SPEC_機器番号がありません");
        saijohatsu=50.0;  //******************090725
    }

    //  機器種別  0_定速型、1_インバータ型
    this.m_ty = spec.getSpecValue( this.SPEC_m_ty, "FCU_EHP_2015" );

    //kiki_file="EHP_WaterCooled20100625";
    this.kiki_file="FCU_EHP_2015";
    //
    if(m_ty.equals("0_201303_定速型")){
        this.kiki_file="EHP_WaterCooled201303";
    }
    if(m_ty.equals("1_201303_インバータ型")){
        this.kiki_file="EHP_WaterCooledInv201303";
                    //EHP_WaterCooledInv201303
    }

    //
    if(m_ty.equals("0_定速型")){
        this.kiki_file="EHP_WaterCooled20100625";
    }
    if(m_ty.equals("1_インバータ型")){
        this.kiki_file="EHP_WaterCooledInv20100625";
    }


    // 機器型式
    this.m_kt = spec.getSpecValue( this.SPEC_m_kt, "" );

    // 台数
    this.in_dai = spec.getSpecValue( this.SPEC_in_dai, 1. );

    // 定格冷房能力
    this.in_Qc_S    = spec.getSpecValue( this.SPEC_in_Qc_S, 0. ) * in_dai;
```

```
        this.in_Qc_S_1 = spec.getSpecValue( this.SPEC_in_Qc_S, 0. );

        // 中間冷房能力
        this.in_Qc_FCU   = spec.getSpecValue( this.SPEC_in_Qc_FCU, 0. ) * in_dai;
        this.in_Qc_FCU_1 = spec.getSpecValue( this.SPEC_in_Qc_FCU, 0. );

        // 定格冷房入力(電力)
        this.in_PPEc_S   = spec.getSpecValue( this.SPEC_in_PPEc_S, 0. ) * in_dai;
        this.in_PPEc_S_1 = spec.getSpecValue( this.SPEC_in_PPEc_S, 0. );

        // 中間冷房入力(電力)
        this.in_PPEc_FCU   = spec.getSpecValue( this.SPEC_in_PPEc_FCU, 0. ) * in_dai;
        this.in_PPEc_FCU_1 = spec.getSpecValue( this.SPEC_in_PPEc_FCU, 0. );

        // 定格暖房能力
        this.in_Qh_S   = spec.getSpecValue( this.SPEC_in_Qh_S, 0. ) * in_dai;
        this.in_Qh_S_1 = spec.getSpecValue( this.SPEC_in_Qh_S, 0. );

        // 中間暖房能力
        this.in_Qh_FCU   = spec.getSpecValue( this.SPEC_in_Qh_FCU, 0. ) * in_dai;
        this.in_Qh_FCU_1 = spec.getSpecValue( this.SPEC_in_Qh_FCU, 0. );

/*      // 低温暖房能力
        this.out_Qh_L   = spec.getSpecValue( this.SPEC_out_Qh_L, 0. ) * in_dai;
        this.out_Qh_L_1 = spec.getSpecValue( this.SPEC_out_Qh_L, 0. ) * in_dai;
*/
        // 定格暖房入力(電力)
        this.in_PPEh_S   = spec.getSpecValue( this.SPEC_in_PPEh_S, 0. ) * in_dai;
        this.in_PPEh_S_1 = spec.getSpecValue( this.SPEC_in_PPEh_S, 0. );

        // 中間暖房入力(電力)
        this.in_PPEh_FCU   = spec.getSpecValue( this.SPEC_in_PPEh_FCU, 0. ) * in_dai;
        this.in_PPEh_FCU_1 = spec.getSpecValue( this.SPEC_in_PPEh_FCU, 0. );

/*      // 低温暖房入力(電力)
        this.out_PPEout_PPEh_Lh_C   = spec.getSpecValue( this.SPEC_out_PPEh_L, 0. ) * in_dai;
        this.out_PPEout_PPEh_Lh_C_1 = spec.getSpecValue( this.SPEC_out_PPEh_L, 0. );
*/
/*
        // クランクケースヒータ(運転時)
        this.out_CLEa_R   = spec.getSpecValue( this.SPEC_out_CLEa_R, 0. ) * in_dai;
        this.out_CLEa_R_1 = spec.getSpecValue( this.SPEC_out_CLEa_R, 0. );

        // クランクケースヒータ(停止時)
        this.out_CLEa_S   = spec.getSpecValue( this.SPEC_out_CLEa_S, 0. ) * in_dai;
        this.out_CLEa_S_1 = spec.getSpecValue( this.SPEC_out_CLEa_S, 0. );

        // 待機電力(運転時)
        this.out_TKEa_R   = spec.getSpecValue( this.SPEC_out_TKEa_R, 0. ) * in_dai;
        this.out_TKEa_R_1 = spec.getSpecValue( this.SPEC_out_TKEa_R, 0. );

        // 待機電力(待機時)
        this.out_TKEa_T   = spec.getSpecValue( this.SPEC_out_TKEa_T, 0. ) * in_dai;
        this.out_TKEa_T_1 = spec.getSpecValue( this.SPEC_out_TKEa_T, 0. );

        // 待機電力(停止時)
        this.out_TKEa_S   = spec.getSpecValue( this.SPEC_out_TKEa_S, 0. ) * in_dai;
        this.out_TKEa_S_1 = spec.getSpecValue( this.SPEC_out_TKEa_S, 0. );
*/

        //SPEC_de_fRate_watInHS   = "熱源水定格水量[g/s]";
        this.de_fRate_watInHS = spec.getSpecValue( this.SPEC_de_fRate_watInHS, 0. );

        if( this.de_fRate_watInHS <= 0 ){
           BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                 SystemMessageConstants.CAL_ERROR2 ,
                 new String[]{ "( "+de_fRate_watInHS+"/"+moduleName+" )[ PAC EHP水熱源 ]", "熱源水定格水量を設定してくださ
い"} ));
           this.de_fRate_watInHS = this.in_Qc_S * 0.11;//20110505nino
        }

/*      // 機器起動停止負荷率
```

```
        if(null!=map.get(this.SPEC_in_run_stop)){
           this.in_run_stop = Double.parseDouble((String)map.get(this.SPEC_in_run_stop))*100.0;
        } else {
           System.out.println(this.moduleName +"(E)SPEC_機器起動停止負荷率がありません");
           in_run_stop=30.0;
        }
*/
        // 相数
        this.phase = spec.getSpecValue( this.SPEC_Phase, 3 );

        // 電圧
        this.voltage = spec.getSpecValue( this.SPEC_Voltage, 200. );

        // 周波数
        this.frequency = spec.getSpecValue( this.SPEC_Frequency, 50. );

        // 力率
        this.powerFactor = spec.getSpecValue( this.SPEC_PowerFactor, 0.8 );



        // 定格風量
        this.in_Va_S   = spec.getSpecValue( this.SPEC_in_Va_S, 0. ) * in_dai;
        this.in_Va_S_1 = spec.getSpecValue( this.SPEC_in_Va_S, 0. );

        if( this.in_Va_S <= 0 ){
           BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                 SystemMessageConstants.CAL_ERROR2 ,
                 new String[]{ "( "+in_Va_S+"/"+moduleName+" )[ PAC EHP水熱源 ]", "風量を設定してください"} ));

        }

        // 定格ファン消費電力
        this.in_PPEa_S   = spec.getSpecValue( this.SPEC_in_PPEa_S, 0. ) * in_dai;
        this.in_PPEa_S_1 = spec.getSpecValue( this.SPEC_in_PPEa_S, 0. );

/*
        // 待機電力(運転時)
        this.in_TKEa_R   = spec.getSpecValue( this.SPEC_in_TKEa_R, 0. ) * in_dai;
        this.in_TKEa_R_1 = spec.getSpecValue( this.SPEC_in_TKEa_R, 0. );

        // 待機電力(停止時)
        this.in_TKEa_S   = spec.getSpecValue( this.SPEC_in_THEa_S, 0. ) * in_dai;
        this.in_TKEa_S_1 = spec.getSpecValue( this.SPEC_in_THEa_S, 0. );
*/

        // 機器起動停止負荷率
        this.in_run_stop = spec.getSpecValue( this.SPEC_in_run_stop, 0.3 );

        this.in_run_stop_Num = this.in_run_stop;

        // 定格加湿能力
        this.in_HUM_mx   = spec.getSpecValue( this.SPEC_in_HUM_mx, 0. ) * in_dai;
        this.in_HUM_mx_1 = spec.getSpecValue( this.SPEC_in_HUM_mx, 0. );

        // 加湿飽和効率
        this.in_HUM_rt   = spec.getSpecValue( this.SPEC_in_HUM_rt, 0.7 ) * 100.;

        // 加湿器ONOFF湿度
        this.in_HUM_on   = spec.getSpecValue( this.SPEC_in_HUM_on, 0.4 ) * 100.;

        // 取入外気量
        this.in_OA_S   = spec.getSpecValue( this.SPEC_in_OA_S, 0. ) * in_dai;
        //this.in_OA_S_1 = spec.getSpecValue( this.SPEC_in_OA_S, 0. );

        // 全熱交換器効率
        this.in_EX_S = spec.getSpecValue( this.SPEC_in_EX_S, 0.6 );

        //SPEC_isHexbypass = "熱交バイパスあり";
        this.isHexbypass = spec.getSpecValue( this.SPEC_isHexbypass, false );

        // 全熱交換器消費電力[W]
```

```
    this.in_EX_PE   = spec.getSpecValue( this.SPEC_in_EX_PE, 0. ) * in_dai;
    //this.in_EX_PE_1 = spec.getSpecValue( this.SPEC_in_EX_PE, 0. );


    //isGVisibleを取得
    this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

    // 最大同時表示ステップ数
    this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

    //isRecordを取得
    this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );


    //低負荷領域の計算方法
    this.calcTypeLowerRangeLoad = spec.getSpecValue( this.SPEC_CalcTypeLowerRangeLoad, "0_発停運転" );
    //calcTypeLowerRangeLoad
    //  0_発停運転
    //  1_下限入力値固定
    //  2_下限COP値固定
    //  3_下限入力値と中間切片
    if( this.calcTypeLowerRangeLoad.equals( "0_発停運転" )){
       this.num_calcTypeLowerRangeLoad = 0;
    }else if(this.calcTypeLowerRangeLoad.equals( "1_下限入力値固定" )){
       this.num_calcTypeLowerRangeLoad = 1;
    }else if(this.calcTypeLowerRangeLoad.equals( "2_下限COP値固定" )){
       this.num_calcTypeLowerRangeLoad = 2;
    }else if(this.calcTypeLowerRangeLoad.equals( "3_下限入力値と中間切片" )){
       this.num_calcTypeLowerRangeLoad = 3;
    }else{
       this.num_calcTypeLowerRangeLoad = 0;
    }

    //isAdjust2012 = "台数を調整する";//
    this.isAdjust2012 = spec.getSpecValue( this.SPEC_isAdjust2012, false );

    // 調整の計算ステップ数
    this.numAdjustSteps = spec.getSpecValue( this.SPEC_NumAdjustSteps, 12 );


    this.daiListc = new LinkedList<Double>();
    this.daiListh = new LinkedList<Double>();
    for( int i=0; i<this.numAdjustSteps; i++ ){
       this.daiListc.add( 0. );
       this.daiListh.add( 0. );
    }
    this.rcList = new LinkedList<Double>();
    this.rhList = new LinkedList<Double>();
    for( int i=0; i<this.numAdjustSteps; i++ ){
       this.rcList.add( 0. );
       this.rhList.add( 0. );
    }

    this.fRate_watInHSc = this.de_fRate_watInHS;
    this.fRate_watInHSh = this.de_fRate_watInHS;

}



@Override
public void initialize(IBestStateMessage stateNodes,
       IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
       IBestDomainSchedule schedule) {
    //System.out.println(this.moduleName + " BuillMultiOutイニシャライズ");
    //状態ノードを受け取る
    super.sm = stateNodes;
    //制御ノードを受け取る
    super.cm = commandNodes;
    //記録ノードを受け取る
    super.rm = recordNodes;
```

```
        //eleIn
        this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

        //airInOA
        this.airInOA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInOA );
        this.airInOA.setMaxFlowRate( this.in_OA_S );//20150423

        //airOutEA
        this.airOutEA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutEA );
        this.airOutEA.setMaxFlowRate( this.in_OA_S );//20150423

        //watInC H
        this.watInC = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInC );
        this.watInH = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInH );

        //watOutC H
        this.watOutC = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutC );
        this.watOutH = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutH );

        this.watInHS = new  BestWater();
        this.watOutHS = new  BestWater();

        //watInHS
        //this.watInHS = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watIn );

        //watOutHS
        //this.watOutHS = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOut );

//      rmlinemap= (Map<String,BM_EHPdata>)super.sm.getState(super.getConnectionNode(this.S_NODE_valInLine));
//      if(null == rmlinemap) {
//         rmlinemap = new HashMap<String,BM_EHPdata>();
//         System.out.println("out=NULL ");
//      } else {
//         num = 0;
//         Set<String> LineEntry = this.rmlinemap.keySet();
//         for(Iterator i = LineEntry.iterator();i.hasNext();){
//            String key = (String) i.next();
////             System.out.println(" name="+name+" key="+key);
//            RMdata = rmlinemap.get(key);
//            rmnm[num]=key;
////             System.out.println("key="+key+" 0="+rmdata[0]+" 1="+rmdata[1]+" 2="+rmdata[2]+" 3="+rmdata[3]+"
4="+rmdata[4]+" 5="+rmdata[5]+" 6="+rmdata[6]);
//            num++;
//         }
//         //出力設定
//         for(int i=0;i<num;i++){
//            RMdata = rmlinemap.get(rmnm[i]);
//            RMdata.set_kiki(kiki_file);
//            rmlinemap.put(rmnm[i],RMdata);
//         }
//         super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine), rmlinemap);
//      }


        filenames[0]=kiki_file;
        equipmentName=kiki_file;
        pacDB=new PACDBManager(path,filenames);
        pacDB.setEquipmentName(equipmentName);

        System.out.println( "filenames =" +filenames[0] + "  equipmentName="+equipmentName);
        //201509
        this.t_watInHS_UpperLimitC = this.pacDB.getformulaRangeMaxRaw( this.pacDB.getFACKcQtwi() );
        this.t_watInHS_LowerLimitC = this.pacDB.getformulaRangeMinRaw( this.pacDB.getFACKcQtwi() );
        this.wb_airInRA_UpperLimitC = this.pacDB.getformulaRangeMaxCol( this.pacDB.getFACKcQtwi() );
        this.wb_airInHS_LowerLimitC = this.pacDB.getformulaRangeMinCol( this.pacDB.getFACKcQtwi() );

        this.t_watInHS_UpperLimitH = this.pacDB.getformulaRangeMaxRaw( this.pacDB.getFACKhQtwi() );
        this.t_watInHS_LowerLimitH = this.pacDB.getformulaRangeMinRaw( this.pacDB.getFACKhQtwi() );
        this.db_airInRA_UpperLimitH = this.pacDB.getformulaRangeMaxCol( this.pacDB.getFACKhQtwi() );
        this.db_airInRA_LowerLimitH = this.pacDB.getformulaRangeMinCol( this.pacDB.getFACKhQtwi() );
```

```
        //airOutRM
        this.airOutSA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutSA );
        this.airOutSA.setMaxFlowRate( this.in_Va_S );//20150423
        if( this.isUseZoneAir ){
           this.zoneAirforSSinside.checkNumberAirChange( this.in_Va_S, "in_Va_S", this.moduleName, "initialize()",
this.name, this.isRecord, this.message);
           this.zoneAirforSSinside._initialize( );
        }

        //**************090407追加**************
        //watInCW
        this.watInCW = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInCW );

        //**************************************


        //watOutD
        this.watOutD = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutD );

        //出口


        //接続ノード　入口
        //S_NODE_airInRA
        this.airInRA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInRA );
        this.airInRA.setTempDB(24.0);
        this.airInRA.setHumi(0.01);
        this.airInRA.setMaxFlowRate( this.in_Va_S );//20150423

        //S_NODE_valInPID
        this.valInPID = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInPID );


        //グラフ表示の準備
        if( this.isGVisible ){
           this.gData = new GraphCommonData2015[13];

           int i = 0;

           this.gData[i++] = new GraphCommonData2015( "M_watIn[g/s]", this.maxItemCount, 0, 50, 0 );
           this.gData[i++] = new GraphCommonData2015( "M_watInCW[g/s]", this.maxItemCount, 0, 50, 0 );
           this.gData[i++] = new GraphCommonData2015( "M_watOutD[g/s]", this.maxItemCount, 0, 50, 0 );

           this.gData[i++] = new GraphCommonData2015( "T_watIn[℃]", this.maxItemCount, 0, 50, 1 );
           this.gData[i++] = new GraphCommonData2015( "T_watOut[℃]", this.maxItemCount, 0, 50, 1 );
           this.gData[i++] = new GraphCommonData2015( "T_watInCW[℃]", this.maxItemCount, 0, 50, 1 );
           this.gData[i++] = new GraphCommonData2015( "T_airIn[℃]", this.maxItemCount, 0, 50, 1 );
           this.gData[i++] = new GraphCommonData2015( "T_airOut[℃]", this.maxItemCount, 0, 50, 1 );

           this.gData[i++] = new GraphCommonData2015( "Q_opeLoadT[W]", this.maxItemCount, 0, 1000, 2 );
           this.gData[i++] = new GraphCommonData2015( "Q_opeCapaT[W]", this.maxItemCount, 0, 1000, 2 );
           this.gData[i++] = new GraphCommonData2015( "P_eleIn[W]", this.maxItemCount, 0, 1000, 2 );

           this.gData[i++] = new GraphCommonData2015( "COP[-]", this.maxItemCount, 0, 10, 3 );
           this.gData[i++] = new GraphCommonData2015( "PID_valIn[-]", this.maxItemCount, 0, 10, 3 );

           String[] yName = { "質量流量[g/s]", "温度[℃]", "熱量・電力[W]","COP・PID[-]" };

           gGCJF = new GraphCommonJFrame2015( this.name+"_FCUHP", this.gData, yName );
        }

     }


     /**
      * 水冷EHP計算及び結果出力/各室内機よりも先に動きます。
      */
     @Override
     public void outputs() {

        if (null == super.cm || null == super.sm ){
           return;
```

```
}

/**
 * 入力
 */
//---20060211-----------------------------------------------
this.PPE_out = 0;
this.opeLoadS = 0;
this.opeLoadT = 0;
this.coolingLoadT = 0.0;//要求処理熱量（冷却全熱）[W]
this.heatingLoadT = 0.0;//要求処理熱量（加熱全熱）[W]
this.opeHeatingLoadS = 0.0;//室内機処理熱量（加熱顕熱）[W]
this.opeCoolingLoadS = 0.0;//室内機処理熱量（冷却顕熱）[W]
this.opeHeatingLoadT = 0.0;//室内機処理熱量（加熱全熱）[W]
this.opeCoolingLoadT = 0.0;//室内機処理熱量（冷却全熱）[W]


if(dbflg==0){
   //filenames[0]=RMdata.get_kiki();
   //equipmentName=RMdata.get_kiki();
   //pacDB=new PACDBManager(path,filenames);
   //pacDB.setEquipmentName(equipmentName);
}
dbflg = 1;

//
this.airInRA = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airInRA));
if( this.isUseZoneAir ){
   this.zoneAirforSSinside._outputSetRA( this.airInRA );
}

this.airInOA = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airInOA));

this.watInC = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watInC));
if( Double.isNaN( watInC.getTemp() ) ){//20150305
   BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
         SystemMessageConstants.WARNING ,
         new String[]{ "(E)入口水温が異常値", this.name }));
   BestLogger.info( "(E)入口水温が異常値"
         + "__"+this.name+"__("+moduleName+"_outputs_watInC)" + AirSystemControl.getTimeStr() );
}

this.watInH = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watInH));
if( Double.isNaN( watInH.getTemp() ) ){//20150305
   BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
         SystemMessageConstants.WARNING ,
         new String[]{ "(E)入口水温が異常値", this.name }));
   BestLogger.info( "(E)入口水温が異常値"
         + "__"+this.name+"__("+moduleName+"_outputs_watInH)" + AirSystemControl.getTimeStr() );
}

//this.watInHS = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watIn ));
//if( Double.isNaN( watInHS.getTemp() ) ){//20150305
// BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
//       SystemMessageConstants.WARNING ,
//       new String[]{ "(E)入口水温が異常値", this.name }));
// BestLogger.info( "(E)入口水温が異常値"
//       + "__"+this.name+"__("+moduleName+"_outputs_watInHS)" + AirSystemControl.getTimeStr() );
//}


//停止中も熱源水は入口の状態で出ていくものとする
//this.watOutHS.copyAllState( this.watInHS );

this.swcIn     = super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcIn));
this.swcInOA   = super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcInOA));
this.modIn     = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modIn));
this.modInPID  = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modInPID));
this.modInPIPE = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modInPIPE));

//PID操作量
this.valInPID = (BestValue) super.sm.getState( super.getConnectionNode( this.S_NODE_valInPID));
```

```
//
if( Airswc.isOFF( this.swcIn )){
   //停止
   //swcInで停止のときのみ入口流量を０とする
   this.fRate_watInHS = 0;//定格流量で定流量
   this.mState = 0;

}else if( Airswc.isON( this.swcIn )){
   //運転
   if( Airmod.isCOOL( this.modIn ) && Airmod.isHEAT( this.modIn )){
      //冷暖房同時の時はmodInPIPEで冷水と温水の切り替えを行う。
      //冷水・温水配管切替 入口
      if( Airmod.isCOOL( this.modInPIPE ) ){
         if( this.isRecord ){
            this.message.append( "(C)modIn冷暖modPIPE冷→冷水" );
         }
         //冷暖房、冷水使用
         this.fRate_watInHS = this.fRate_watInHSc;//定格流量で定流量
         this.watInC.setFlowRate( this.fRate_watInHS );
         this.watInH.setFlowRate( 0 );
         this.watInHS.copyAllState( this.watInC );
         this.isCPipe = true;

      }else if( Airmod.isHEAT( this.modInPIPE )){
         if( this.isRecord ){
            this.message.append( "(C)modIn冷暖modPIPE暖→温水" );
         }
         //冷暖房、温水使用
         this.fRate_watInHS = this.fRate_watInHSh;//定格流量で定流量
         this.watInC.setFlowRate( 0 );
         this.watInH.setFlowRate( this.fRate_watInHS );
         this.watInHS.copyAllState( this.watInH );
         this.isCPipe = false;

      }else{
         //modInPIPEが冷房でも暖房でもないときは　冷房として扱う
         if( this.isRecord ){
            this.message.append( "(C)modIn冷暖modPIPEなし→冷水" );
         }
         //冷暖房、冷水使用
         this.fRate_watInHS = this.fRate_watInHSc;//定格流量で定流量
         this.watInC.setFlowRate( this.fRate_watInHS );
         this.watInH.setFlowRate( 0 );
         this.watInHS.copyAllState( this.watInC );
         this.isCPipe = true;

      }

      //計算が冷房か暖房かは　modInPID　で判断する
      if( Airmod.isCOOL( this.modInPID )){
         if( this.isRecord ){
            this.message.append( "(C)modInPID冷" );
         }
         //冷房運転
         this.mState = 1;
      }else{
         if( this.isRecord ){
            this.message.append( "(C)modInPID暖" );
         }
         //暖房運転
         this.mState = 2;
      }

   }else if( Airmod.isCOOL( this.modIn )){
      //冷房専用の時
      if( this.isRecord ){
         this.message.append( "(C)modIn冷→冷水" );
      }
      //冷房運転、冷水使用
      this.fRate_watInHS = this.fRate_watInHSc;//定格流量で定流量
      this.watInC.setFlowRate( this.fRate_watInHS );
```

```
            this.watInH.setFlowRate( 0 );
            this.watInHS.copyAllState( this.watInC );
            this.isCPipe = true;
            this.mState = 1;

         }else if( Airmod.isHEAT( this.modIn )){
            //暖房専用の時
            if( this.isRecord ){
               this.message.append( "(C)modIn暖→温水" );
            }
            //暖房運転、温水使用
            this.fRate_watInHS = this.fRate_watInHSh;//定格流量で定流量
            this.watInC.setFlowRate( 0 );
            this.watInH.setFlowRate( this.fRate_watInHS );
            this.watInHS.copyAllState( this.watInH );
            this.isCPipe = false;
            this.mState = 2;

         }else if( Airmod.isVENTILATE( this.modIn )){
            if( this.isRecord ){
               this.message.append( "(C)modIn換" );
            }
            //換気モード
            this.mState = 3;
            this.fRate_watInHS = 0;
            this.watInC.setFlowRate( 0 );
            this.watInH.setFlowRate( 0 );
            this.watInHS.copyAllState( this.watInC );

         }else{
            //停止
            this.mState = 0;
         }
         BestAir.checkOpeData( this.airInRA, "airInRA", this.moduleName, "outputs()", this.name, this.isRecord,
this.message );
      }


      /**
       * 入力
       */

      //状態ノードから室内空気を設定
      //室内乾球温度を設定
      this.DBra = airInRA.getTempDB();
      this.WBra = airInRA.getTempWB();

      //室内絶対湿度を設定
      this.XGra = airInRA.getHumi();

      if( Double.isInfinite( this.DBra ) || Double.isNaN( this.DBra )){
         if( this.isRecord ){
            this.message.append( "(E)DBra異常値→0℃に設定");
         }
         this.mState= -1;
         this.DBra = 0;
      }
      if( Double.isInfinite( this.XGra ) || Double.isNaN( this.XGra ) || this.XGra < 0 ){
         if( this.isRecord ){
            this.message.append( "(E)XGra異常値→0.004g/gに設定");
         }
         this.mState= -1;
         this.XGra = 0.004;
      }

      //室内湿球温度を設定
      this.WBra = Psychrometrics.FNWbtx(this.DBra, this.XGra);

      //    System.out.println("DBin="+DBin+" "+XGin+"  "+WBin+"  ");

      //外気乾球温度設定
      this.DBoa = airInOA.getTempDB();
```

```
//外気絶対湿度を設定
this.XGoa = airInOA.getHumi();

//外気湿球温度を設定
this.WBoa = Psychrometrics.FNWbtx(this.DBoa, this.XGoa);
if( WBoa < -9000.0 ) WBoa =WB_cal(this.DBoa, this.XGoa);


//熱源水入口温度　流量
this.t_watInHS = this.watInHS.getTemp();
//this.fRate_watInHS = this.watInHS.getFlowRate();



//    System.out.println("mState="+mState+" "+onOff+"  "+mode);
//    System.out.println("DBoa="+DBoa+" "+XGoa+"  "+WBoa);

//計算の切替

//室内機側の計算
switch(this.mState){
case -1:
   //異常停止
   if( this.isRecord ){
      this.message.append("(C)異常停止");
   }
   this.fRate_watInHS = 0;
   this.calc_Stop_In0();
   this.watInC.setFlowRate( 0 );
   this.watInH.setFlowRate( 0 );
   this.watInHS.setFlowRate( 0 );
   this.watOutC.copyAllState( this.watInC );
   this.watOutH.copyAllState( this.watInH );
   break;
case 0:
   //停止
   if( this.isRecord ){
      this.message.append("(C)停止処理");
   }
   this.calc_Stop_In0();
   this.watInC.setFlowRate( 0 );
   this.watInH.setFlowRate( 0 );
   this.watInHS.setFlowRate( 0 );
   this.watOutC.copyAllState( this.watInC );
   this.watOutH.copyAllState( this.watInH );
   break;
case 1:
   //冷房
   if( this.in_Qc_S_1 > 0 ){
      if( this.isRecord ){
         this.message.append("(C)冷房運転");
      }
      this.GW = this.in_Va_S;
      this.cooling_cal();
      this.PPE_in = this.in_PPEa_S + this.in_TKEa_R + this.in_EX_PE;//20130717
   }else{
      if( this.isRecord ){
         this.message.append("(C)冷能力=0停止");//20130625
      }
      this.calc_Stop_In0();
      this.mState = 0;
   }
   //冷水・温水配管切替 出口
   if( this.isCPipe ){
      this.watOutC.copyAllState( this.watOutHS );
      this.watOutH.copyAllState( this.watInH );

   }else if( !this.isCPipe ){
      this.watOutC.copyAllState( this.watInC );
      this.watOutH.copyAllState( this.watOutHS );
```

```
            }

            break;
        case 2:
            //暖房
            if( this.in_Qh_S_1 > 0 ){
                if( this.isRecord ){
                    this.message.append("(C)暖房運転");
                }
                this.GW = this.in_Va_S;
                this.heating_cal();
                this.PPE_in = this.in_PPEa_S + this.in_TKEa_R + this.in_EX_PE;//20130717
            }else{
                if( this.isRecord ){
                    this.message.append("(C)暖能力=0停止");//20130625
                }
                this.calc_Stop_In0();
                this.mState = 0;
            }
            //冷水・温水配管切替 出口
            if( this.isCPipe ){
                this.watOutC.copyAllState( this.watOutHS );
                this.watOutH.copyAllState( this.watInH );

            }else if( !this.isCPipe ){
                this.watOutC.copyAllState( this.watInC );
                this.watOutH.copyAllState( this.watOutHS );

            }

            break;
        case 3:
            //換気
            if( this.isRecord ){
                this.message.append("(C)換気運転");
            }
            this.GW = this.in_Va_S;
            this.ventilation_cal();
            this.PPE_in = this.in_PPEa_S + this.in_TKEa_R + this.in_EX_PE;//20130717
            //
            this.valInPID.setValue( 0. );
            this.watInC.setFlowRate( 0 );
            this.watInH.setFlowRate( 0 );
            this.watInHS.setFlowRate( 0 );
            this.watOutC.copyAllState( this.watInC );
            this.watOutH.copyAllState( this.watInH );
            break;
        default:
            if( this.isRecord ){
                this.message.append("(C)運転モード？停止");//20130625
            }
            this.fRate_watInHS = 0;
            this.calc_Stop_In0();
            this.watInC.setFlowRate( 0 );
            this.watInH.setFlowRate( 0 );
            this.watInHS.setFlowRate( 0 );
            this.watOutC.copyAllState( this.watInC );
            this.watOutH.copyAllState( this.watInH );
            this.mState = 0;
            System.out.println(this.moduleName + ">>Error<< onOff*modeが範囲外");
        }

//      this.allSc_heat = this.coolingLoadS;//要求冷房顕熱量
//      this.allSh_heat = this.Sh_Load;//要求暖房顕熱量
//      this.coolingLoadT = this.Tc_Load;//要求冷房全熱量
//      this.heatingLoadT = this.Th_Load;//要求暖房全熱量

        if( this.PPE_in + this.PPE_out <= 0 ){
            this.COP = 0;
        }else{
            this.COP = this.opeLoadT / ( this.PPE_in + this.PPE_out );
        }
```

```
      if( this.isAdjust2012 ){
         if( mState == 1 ){
            this.fRate_watInHS = this.fRate_watInHSc;
         }else if( mState == 2 ){
            this.fRate_watInHS = this.fRate_watInHSh;
         }
      }

      //*************************----------------
      //super.sm.setState( super.getConnectionNode( this.S_NODE_watOut ), this.watOutHS );
      super.sm.setState( super.getConnectionNode( this.S_NODE_watOutC ), this.watOutC );
      super.sm.setState( super.getConnectionNode( this.S_NODE_watOutH ), this.watOutH );

      this.E_PPE_in = this.PPE_in * Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 );
      this.E_PPE_out= this.PPE_out* Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 );
      this.eleIn.setActivePower( this.PPE_out + this.PPE_in );
      this.eleIn.setReactivePower(this.E_PPE_out + this.E_PPE_in );
      this.eleIn.setPhase( this.phase );
      this.eleIn.setVoltage( this.voltage );
      this.eleIn.setFrequency( this.frequency );



      //    System.out.println("CAPrate="+CAPrate+" "+S_Load_RM+" "+S_Load+ " "+mState);
      this.airOutSA.setTempDB( this.DBout );
      this.airOutSA.setHumi( this.XGout );
      this.airOutSA.setFlowRate( this.GW );

      this.watOutD.setFlowRate( this.D_Wrate );
      this.watOutD.setTemp( this.DBout );
      this.watInCW.setFlowRate( this.CW_Wrate );
      //    System.out.println("DBout="+DBout+" "+XGout+"  "+airOutRM.getFlowRate());

      //出力側ノード
      super.sm.setState( super.getConnectionNode( this.S_NODE_airOutSA ), this.airOutSA);
      super.sm.setState( super.getConnectionNode( this.S_NODE_eleIn ), this.eleIn);
      super.sm.setState( super.getConnectionNode( this.S_NODE_watOutD ), this.watOutD);
      super.sm.setState( super.getConnectionNode( this.S_NODE_watInCW ), this.watInCW);//090407追加

      //出力設定

      //グラフデータ追加
      if( this.isGVisible ){
         //とりあえずwatOutCW[0]のみグラフ化
         //gTank.addData( BestTimeManager.getDateWeatherTime(),this.watInCW_GT, this.watOutCW[0], this.m_watMy,
this.m_start, this.m_stop, this.m_capacity );

         double[] gData = new double[13];

      /* this.gData[i++] = new GraphCommonData2015( "M_watIn[g/s]", this.maxItemCount, 0, 50, 0 );
         this.gData[i++] = new GraphCommonData2015( "M_watInCW[g/s]", this.maxItemCount, 0, 50, 0 );
         this.gData[i++] = new GraphCommonData2015( "M_watOutD[g/s]", this.maxItemCount, 0, 50, 0 );

         this.gData[i++] = new GraphCommonData2015( "T_watIn[g/s]", this.maxItemCount, 0, 50, 1 );
         this.gData[i++] = new GraphCommonData2015( "T_watOut[g/s]", this.maxItemCount, 0, 50, 1 );
         this.gData[i++] = new GraphCommonData2015( "T_watInCW[g/s]", this.maxItemCount, 0, 50, 1 );
         this.gData[i++] = new GraphCommonData2015( "T_airIn[g/s]", this.maxItemCount, 0, 50, 1 );
         this.gData[i++] = new GraphCommonData2015( "T_airOut[g/s]", this.maxItemCount, 0, 50, 1 );

         this.gData[i++] = new GraphCommonData2015( "Q_opeLoadT[W]", this.maxItemCount, 0, 1000, 2 );
         this.gData[i++] = new GraphCommonData2015( "Q_opeCapaT[W]", this.maxItemCount, 0, 1000, 2 );
         this.gData[i++] = new GraphCommonData2015( "P_eleIn[W]", this.maxItemCount, 0, 1000, 2 );

         this.gData[i++] = new GraphCommonData2015( "COP[-]", this.maxItemCount, 0, 10, 3 );
         this.gData[i++] = new GraphCommonData2015( "Ctrl_valIn[-]", this.maxItemCount, 0, 10, 3 );
      */

         int i=0;
         gData[i++] = this.watInHS.getFlowRate();
         gData[i++] = this.watInCW.getFlowRate();
         gData[i++] = this.watOutD.getFlowRate();
```

```
    gData[i++] = this.watInHS.getTemp();
    gData[i++] = this.watOutHS.getTemp();
    gData[i++] = this.watInCW.getTemp();
    gData[i++] = this.airInRA.getTempDB();
    gData[i++] = this.airOutSA.getTempDB();

    gData[i++] = this.opeLoadT;
    gData[i++] = this.capCooling + this.capHeating;
    gData[i++] = this.eleIn.getActivePower();

    gData[i++] = this.COP;
    gData[i++] = this.valInPID.getValue();

    this.gGCJF.addData( BestTimeManager.getDateWeatherTime(), gData );
}

//記録ノード
if( this.isRecord && super.rm != null ){
    this.record();
}

message.setLength(0);


//*************************----------------
if( this.isAdjust2012 ){
    //移動平均の最大値で調整していく
    if( true ){
        this.daiListc.removeLast();
        this.daiListc.addFirst( this.in_daicAdjust * this.valInPID.getValue() );
        //
        double sum_in_dai = 0;
        for( int i=0; i<this.numAdjustSteps; i++ ){
            sum_in_dai += this.daiListc.get( i );
        }
        this.avecdaiAdjust = sum_in_dai / this.numAdjustSteps;
        //
        if( sum_in_dai > this.maxcAdjustdai * this.numAdjustSteps ){
            this.maxcAdjustdai = this.avecdaiAdjust;

            this.in_daicAdjust = this.maxcAdjustdai;
            this.in_run_stop_Num = this.in_run_stop / this.in_daicAdjust;

            this.in_Qc_S   = this.in_Qc_S_1   * this.in_daicAdjust;//= "定格冷房能力";//kW
            this.in_Qc_FCU = this.in_Qc_FCU_1 * this.in_daicAdjust;//= "中間冷房能力";//kW
            this.in_PPEc_S = this.in_PPEc_S_1 * this.in_daicAdjust;//= "定格冷房入力(電力)";//kW
            this.in_PPEc_FCU = this.in_PPEc_FCU_1 * this.in_daicAdjust;//= "中間冷房入力(電力)";//kW

            this.fRate_watInHSc = this.de_fRate_watInHS * this.in_daicAdjust;//水量[g/s]

            this.in_Va_S   = this.in_Va_S_1   * this.in_daicAdjust;//= "定格風量";//m3/h
            this.in_PPEa_S = this.in_PPEa_S_1 * this.in_daicAdjust;//= "定格ファン消費電力";//W

            this.in_HUM_mx = this.in_HUM_mx_1 * this.in_daicAdjust;//= "定格加湿能力";//kg/h

            this.airInRA.setFlowRate(in_Va_S);
            this.airOutSA.setFlowRate(in_Va_S);
        }
    }


    if( true ){
        this.daiListh.removeLast();
        this.daiListh.addFirst( this.in_daihAdjust * this.valInPID.getValue() );
        //
        double sum_in_dai = 0;
        for( int i=0; i<this.numAdjustSteps; i++ ){
            sum_in_dai += this.daiListh.get( i );
        }
        this.avehdaiAdjust = sum_in_dai / this.numAdjustSteps;
        //
```

```
            if( sum_in_dai > this.maxhAdjustdai * this.numAdjustSteps ){
               this.maxhAdjustdai = this.avehdaiAdjust;

               this.in_daihAdjust = this.maxhAdjustdai;
               this.in_run_stop_Num = this.in_run_stop / this.in_daihAdjust;

               this.in_Qh_S   = this.in_Qh_S_1   * this.in_daihAdjust;//= "定格暖房能力";
               this.in_Qh_FCU   = this.in_Qh_FCU_1   * this.in_daihAdjust;//= "FCU暖房能力";
//             this.in_Qh_L   = this.in_Qh_L_1   * this.in_daihAdjust;//= "低温暖房能力";
               this.in_PPEh_S = this.in_PPEh_S_1 * this.in_daihAdjust;//= "定格暖房入力(電力)";
               this.in_PPEh_FCU = this.in_PPEh_FCU_1 * this.in_daihAdjust;//= "FCU暖房入力(電力)";
//             this.in_PPEh_L = this.in_PPEh_L_1 * this.in_daihAdjust;//= "低温暖房入力(電力)";

               this.fRate_watInHSh = this.de_fRate_watInHS * this.in_daihAdjust;//水量[g/s]

               this.in_Va_S   = this.in_Va_S_1   * this.in_daihAdjust;//= "定格風量";//m3/h
               this.in_PPEa_S = this.in_PPEa_S_1 * this.in_daihAdjust;//= "定格ファン消費電力";//W

               this.in_HUM_mx = this.in_HUM_mx_1 * this.in_daihAdjust;//= "定格加湿能力";//kg/h

               this.airInRA.setFlowRate(in_Va_S);
               this.airOutSA.setFlowRate(in_Va_S);
            }

         }
      }
      //*************************----------------

   }

   private void calc_Stop_In0(){
      //出口空気の状態は入口空気の状態とする
      this.airOutSA.copyAllvalState( this.airInRA );
      this.airOutSA.setFlowRate(0.0);

      this.PPE_in = this.in_TKEa_S;
      this.DBin = this.DBra;
      this.XGin = this.XGra;
      this.WBin = this.WBra;
      //
      this.DBout = this.DBin;//乾球温度
      this.XGout = this.XGin;//絶対湿度
      this.GW = 0.0;
      this.D_Wrate = 0.0;
      this.CW_Wrate = 0.0;
      this.COP = 0.0;
      //
      this.valInPID.setValue( 0. );
      //
      this.watOutHS.setFlowRate( this.fRate_watInHS );

      this.opeCoolingLoadS = 0.0;
      this.opeCoolingLoadT = 0.0;
      this.opeHeatingLoadS = 0.0;
      this.opeHeatingLoadT = 0.0;
      this.opeLoadS = 0.0;
      this.opeLoadT = 0.0;
      this.capCooling = 0.0;
      this.capHeating = 0.0;


   }



   /**
    * 記録
    */
   private void record(){
      if( super.rm != null ){
         if( CheckPrintModule.isPrintMessage ){
```

```
            //message
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_message, this.name, message.toString() );
         }

         if( CheckPrintModule.isPrintEnergy ){
            //記録ノードに室外機消費電力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID4, this.name,
AirFormat.df_2(this.eleIn.getActivePower()));
         }

         if( CheckPrintModule.isPrintLoad ){
            //記録ノードに室内機処理熱量（全熱）を設定
            if( this.opeCoolingLoadT > 0 ){
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT,  this.name, AirFormat.df_2(-
this.opeLoadT ));//処理全熱量合計#W#熱量
            }else{
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT,  this.name,
AirFormat.df_2( this.opeLoadT ));//処理全熱量合計#W#熱量
            }
         }

         if( CheckPrintModule.isPrintStateOut ){
            //記録ノードに吹出口乾球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBout, this.name,
AirFormat.df_2( this.airOutSA.getTempDB() ));
            //記録ノードに吹出口絶対湿度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_XGout, this.name,
AirFormat.df_5( this.airOutSA.getHumi() ));
            //記録ノードに吹出口風量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_GW, this.name,
AirFormat.df_2( this.airOutSA.getFlowRate() ));
            //記録ノードにドレン量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_D_Wrate, this.name,
AirFormat.df_2(D_Wrate));
            //記録ノードに熱源水出口流量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_M_watOutC, this.name,
AirFormat.df_2(this.watOutC.getFlowRate()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_M_watOutH, this.name,
AirFormat.df_2(this.watOutH.getFlowRate()));
            //記録ノードに熱源水出口温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_T_watOutC, this.name,
AirFormat.df_2(this.watOutC.getTemp()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_T_watOutH, this.name,
AirFormat.df_2(this.watOutH.getTemp()));
         }

         if( CheckPrintModule.isPrintStateMy ){

            //記録ノードに要求熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_coolingLoadT, this.name,
AirFormat.df_2(this.coolingLoadT));//冷却要求全熱量合計#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_heatingLoadT, this.name,
AirFormat.df_2(this.heatingLoadT));//加熱要求全熱量合計#W#熱量

            //記録ノードに処理可能熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_capCooling, this.name,
AirFormat.df_2(this.capCooling));//冷却可能熱量/最大#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_capHeating, this.name,
AirFormat.df_2(this.capHeating));//加熱可能熱量/最大#W#熱量

            //記録ノードに室内機処理熱量（顕熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_opeLoadS,  this.name,
AirFormat.df_2(opeLoadS));//処理顕熱量合計#W#熱量
            //記録ノードに室内機処理熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_opeLoadT,  this.name,
AirFormat.df_2(opeLoadT));//処理全熱量合計#W#熱量

            //COP
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_COP, this.name, AirFormat.df_3(this.COP));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_pidVal, this.name,
```

```
AirFormat.df_4(this.valInPID.getValue()));
        }


        if( CheckPrintModule.isPrintStateIn ){
           //記録ノードに加湿給水量を設定
           super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID8i, this.name,
AirFormat.df_2(CW_Wrate));
           //記録ノードに入口乾球温度を設定
           super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBairIn, this.name,
AirFormat.df_2(DBin));
           //記録ノードに入口絶対湿度を設定
           super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_XairIn, this.name,
AirFormat.df_5(XGin ));
           //記録ノードに入口乾球温度を設定
           super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBairInRM, this.name,
AirFormat.df_2( this.airInRA.getTempDB() ));
           //記録ノードに入口絶対湿度を設定
           super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_XairInRM, this.name,
AirFormat.df_5( this.airInRA.getHumi() ));

           //記録ノードに熱源水出口温度を設定
           super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_T_watInC, this.name,
AirFormat.df_2(this.watInC.getTemp()));
           super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_T_watInH, this.name,
AirFormat.df_2(this.watInH.getTemp()));
        }

        if( CheckPrintModule.isPrintAdjust ){
           //
           if( this.isAdjust2012 ){
              //記録ノードに室外機調整能力を設定
              super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qc_Adjust, this.name,
this.in_Qc_S );//室外機調整能力[W]out_Qc_Adjust
              super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qh_Adjust, this.name,
this.in_Qh_S );//室外機調整能力[W]out_Qh_Adjust
              //記録ノードに室外機調整能力を設定
              //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rcAdjust", this.name,
this.out_rcAdjust );//室外機調整率[-]
              //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rhAdjust", this.name,
this.out_rhAdjust );//室外機調整率[-]
              //記録ノードに調整台数を設定
              super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_in_daicAdjust, this.name,
AirFormat.df_2(this.in_daicAdjust) );
              super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_in_daihAdjust, this.name,
AirFormat.df_2(this.in_daihAdjust) );
              super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_avecdaiAdjust, this.name,
AirFormat.df_2(this.avecdaiAdjust) );
              super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_avehdaiAdjust, this.name,
AirFormat.df_2(this.avehdaiAdjust) );
           }
        }
     }
  }

  /**
   * 水冷EHP計算/各室内機の集計後
   */

  @Override
  public void update() {
     if( this.isUseZoneAir ){
        this.zoneAirforSSinside._update( this.airOutSA );
     }
  }

  public Object viewInternal(TestCommand cmd) {

     ArrayList<Object> result = new ArrayList<Object>();
     result.add(super.sm);
     result.add(super.cm);
     result.add(super.rm);
```

```
        //外部定義
        result.add(this.name);
        result.add(this.m_kt);

        result.add(this.airOutEA.getTempDB());

        result.add(this.in_Qc_S);
        result.add(this.in_Qc_FCU);
        result.add(this.in_PPEc_S);
        result.add(this.in_PPEc_FCU);
        result.add(this.in_Qh_S);
        result.add(this.in_Qh_FCU);
//      result.add(this.in_Qh_L);
        result.add(this.in_PPEh_S);
        result.add(this.in_PPEh_FCU);
//      result.add(this.in_PPEh_L);

        result.add(this.DBoa);
        result.add(this.XGoa);
        result.add(this.DBra);
        result.add(this.XGra);

        return result;
    }




    private double WB_cal(double ddb,double xx){
        double wwb,x1,de=1;
        int kk=-1,j=0;
        wwb=ddb;

        do{
            j++;
            x1=Psychrometrics.FNXtw(ddb,wwb);
            if(Math.abs(x1-xx)<0.000003) {
                //          System.out.println(" j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
                return wwb;
            }
            if(x1<xx) {
                if(kk==-1) de=de/2.0;
                wwb=wwb+de;
                kk=1;
            }
                if(x1>=xx) {
                if(kk==1) de=de/2.0;
                wwb=wwb-de;
                kk=-1;
            }

        }while(j<1000);
        //    System.out.println("*j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
        return wwb;

    }



    private void cooling_cal(){
        //室側の冷房計算
        double heatCapacity = 0.0;
        double heatCapacityHPFCU=0.0;
        double heatCapacityFCU =0.0;
        double eleHPFCU = 0.0;
        double eleFCU = 0.0;
        double ele = 0.0;
        double rCode0 = 0.0,rCode1 = 0.0;
        double IAin,IAout1,DBout1,XGout1;
        double HEXrate;
```

```
      HEXrate = this.in_EX_S;

      //外気取り入れありの時の吸い込み状態の計算
      //バイパス制御 追加 エンタルピで判断
      if( this.isHexbypass == true ){  //20101212nino
         //全熱交換器バイパス有
         this.IAra = Psychrometrics.FNH( this.DBra, this.XGra );//20130109
         this.IAoa = Psychrometrics.FNH( this.DBoa, this.XGoa );

         if( this.IAra > this.IAoa && this.valInPID.getValue() > 0.001 && this.HEXselect==1 ){//20130109
            //エンタルピ基準
            HEXrate = 0.0;
         }
         if( this.DBra > this.DBoa && this.valInPID.getValue() > 0.001 && this.HEXselect==2 ){//20130109
            //温度/顕熱基準
            HEXrate = 0.0;
         }
      }
      if( Airswc.isON( this.swcInOA )){//20101212nino
         //吸込み状態を求める
         this.DBin = (( this.in_Va_S - this.in_OA_S ) * this.DBra + (( this.DBoa - this.DBra ) * ( 1.0 - HEXrate ) +
this.DBra ) * this.in_OA_S ) / this.in_Va_S;//20130109
         this.XGin = (( this.in_Va_S - this.in_OA_S ) * this.XGra + (( this.XGoa - this.XGra ) * ( 1.0 - HEXrate ) +
this.XGra ) * this.in_OA_S ) / this.in_Va_S;//20130109
      }else{//20101212nino
         DBin = DBra;//20130109
         XGin = XGra;//20130109
      }
      this.WBin = Psychrometrics.FNWbtx( this.DBin, this.XGin );

      // System.out.println("C/DBin="+DBin+" "+XGin+"  ");
      // System.out.println("C/DBoa="+DBoa+" "+XGoa+"  ");
      // System.out.println("C/DBra="+DBra+" "+XGra+"  ");
      // System.out.println("  ");
      // System.out.println("DBra="+DBra+" "+XGra+"  ");

      //特性の上下限のチェック 20120821nino
      double twatIn = this.t_watInHS;
      double wbIn = this.WBin;

      if( twatIn > this.t_watInHS_UpperLimitC ){
         //熱源水入口温度＞上限の時停止
         if( this.isRecord ){
            this.message.append( "(C)熱源水入口温度＞"+this.t_watInHS_LowerLimitC+"上限→停止");
         }

         this.mState = 3;//能力＝０→換気モードとする20130415
         this.calc_Stop_In0();
         return;
      }
      if( twatIn < this.t_watInHS_LowerLimitC ){
         //熱源水入口温度＜下限の時　下限値の特性で運転
         if( this.isRecord ){
            this.message.append( "(C)熱源水入口温度＜"+this.t_watInHS_LowerLimitC+"下限→下限値で特性計算");
         }
         twatIn = this.t_watInHS_LowerLimitC;
      }
      if( wbIn > this.wb_airInRA_UpperLimitC ){
         //室内湿球温度＞上限の時　上限値の特性で運転
         if( this.isRecord ){
            this.message.append( "(C)室WB＞"+this.wb_airInRA_UpperLimitC+"上限→上限値で特性計算");
         }
         wbIn = this.wb_airInRA_UpperLimitC;
      }
      if( wbIn < this.wb_airInHS_LowerLimitC ){
         //室内湿球温度＜下限の時　下限値の特性で運転
         if( this.isRecord ){
            this.message.append( "(C)室WB＜"+this.wb_airInHS_LowerLimitC+"下限→下限値で特性計算");
         }
         wbIn = this.wb_airInHS_LowerLimitC;
      }
```

```
        //能力補正
        rCode0 = this.pacDB.getKcQtwi( twatIn, wbIn );//水温WB補正
        rCode1 = this.pacDB.getKcf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
        heatCapacityHPFCU = rCode0 * rCode1 * this.in_Qc_S;

        rCode0 = this.pacDB.getKcQtwif( twatIn, wbIn );//水温WB補正
        //rCode1 = this.pacDB.getKcf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
        heatCapacityFCU =  rCode0 * rCode1 * this.in_Qc_FCU;

        //要求負荷　処理負荷
        this.coolingLoadT = heatCapacityHPFCU * this.valInPID.getValue();
        this.opeCoolingLoadT = this.coolingLoadT;

        //ele
        rCode0 = this.pacDB.getKcWtwi( twatIn, wbIn );//水温WB補正
        rCode1 = this.pacDB.getKcwf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
        eleHPFCU = rCode0 * rCode1 * this.in_PPEc_S;

        rCode0 = this.pacDB.getKcWtwif( twatIn, wbIn );//水温WB補正
        //rCode1 = this.pacDB.getKcwf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
        eleFCU = rCode0 * rCode1 * this.in_PPEc_FCU;

        if( heatCapacityHPFCU < this.coolingLoadT ){
           //要求負荷が能力を超えているとき
           heatCapacity = heatCapacityHPFCU;
           this.opeCoolingLoadT = heatCapacityHPFCU;
           ele = eleHPFCU;

        }else if( heatCapacityFCU >= this.coolingLoadT ){
           //FCUだけで能力が足りている
           heatCapacity = heatCapacityFCU;
           //ele
           ele = eleFCU;

        }else{
           //HPの運転が必要
           heatCapacity = heatCapacityHPFCU;
           //ele
           ele = eleHPFCU;

        }

        this.capCooling = heatCapacity;


        //吹き出し状態の計算
        IAin   = Psychrometrics.FNH(this.DBin, this.XGin);
        IAout1 = IAin - this.opeCoolingLoadT / this.in_Va_S * 1000.0;
        DBout1 = Psychrometrics.FNDbrh( 90.0, IAout1 );//乾球温度
        XGout1 = Psychrometrics.FNxtr( DBout1, 90.0);//絶対湿度
        // System.out.println("IAra="+IAra+" "+IAout1+"  "+MHP_in_Vc_S);

        if( this.XGin < XGout1 ){
           XGout1 = this.XGin;
           DBout1 = this.DBin - this.opeCoolingLoadT / this.in_Va_S;
           this.opeCoolingLoadS = this.opeCoolingLoadT;

        }else{
           this.opeCoolingLoadS = ( this.DBin - DBout1 ) * this.in_Va_S;
        }

        this.DBout = DBout1;//乾球温度
        this.XGout = XGout1;//絶対湿度

        /*
           if(saijohatsu>0.0 && Psychrometrics.FNRhtx(DBin, XGin)<40.0){
              System.out.println("40以下"+(++u40)+"S_Load_RM="+S_Load_RM+" T_Load=_RM"+T_Load_RM+" sRc="+sRc+"
SS_rm="+SS_rm);
              System.out.println("   DBout0="+(DBra-S_Load_RM/in_Va_S)+"  DBout1="+(DBra-SS_rm/in_Va_S));
              System.out.println("   XXout0="+Psychrometrics.FNXth((DBra-S_Load_RM/in_Va_S), IAout)+"
XXout1="+Psychrometrics.FNXth((DBra-SS_rm/in_Va_S), IAout));
```

```
      System.out.println("   DBin  ="+DBin+"  XGin  ="+XGin+"  RHin  ="+Psychrometrics.FNRhtx(DBin, XGin));
      System.out.println("   DBoa  ="+DBoa+"  XGoa  ="+XGoa+"  RHoa  ="+Psychrometrics.FNRhtx(DBoa, XGoa));
      System.out.println("   DBra  ="+DBra+"  XGra  ="+XGra+"  RHra  ="+Psychrometrics.FNRhtx(DBra, XGra));

   }
 */

// System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate);
double Rp;

if( this.capCooling <= 0. ){
   this.Rc = 0;

}else{
   this.Rc = this.coolingLoadT / this.capCooling;

}

if(this.Rc > 1.0 ) {
   this.Rc = 1.0;
}

//低負荷領域の計算仕分け:Rh
if( this.Rc < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = this.Rc;
}


//低負荷領域の計算仕分け:PPE
if( this.Rc < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      ele = 0.0;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      ele *= ( this.Rc / Rp );
      break;

   case 3://  3_下限入力値と中間切片
      ele *= ( 0.5 + this.Rc / Rp / 2. );
      break;

   default:
   }
}else{
   //何もしない
}

this.PPE_out = ele;

//airOutEA
//this.airOutEA.setFlowRate( this.fRate_watInHS );
```

```
      //this.airOutEA.setTempDB( ( this.T_Loadout + this.PPE_out ) / this.fRate_watInHS * 1000. +
this.airInOA.getTempDB() );

      //watOutHS
      this.watOutHS.setFlowRate( this.fRate_watInHS );
      this.watOutHS.setTemp( ( this.opeCoolingLoadT + this.PPE_out ) / this.fRate_watInHS / 4.18605 + this.t_watInHS );

      if( this.Rc < 0.0001 ){
         this.opeCoolingLoadS = 0.0;//実処理熱量（冷却顕熱）[W]
         this.opeCoolingLoadT = 0.0;//実処理熱量[W]
         this.capCooling      = 0.0;//処理可能熱量[W]
         this.opeHeatingLoadS = 0.0;//実処理熱量（加熱顕熱）[W]
         this.opeHeatingLoadT = 0.0;//実処理熱量[W]
         this.capHeating      = 0.0;//処理可能熱量[W]
         this.PPE_out = 0.0;
         this.watOutHS.copyAllState( this.watInHS );//20130110
         if( this.isRecord ){
            this.message.append( "(C)負荷率0で停止");
         }
         this.mState = 3;//能力＝０→換気モードとする20130415

      }else if( this.Rc < this.in_run_stop ){
         switch( this.num_calcTypeLowerRangeLoad){
         case 0://  0_発停運転
            this.opeCoolingLoadS = 0.0;//実処理熱量（冷却顕熱）[W]
            this.opeCoolingLoadT = 0.0;//実処理熱量（冷却全熱）[W]
            this.capCooling      = 0.0;//処理可能熱量（冷却全熱）[W]
            this.opeHeatingLoadS = 0.0;//実処理熱量（加熱顕熱）[W]
            this.opeHeatingLoadT = 0.0;//実処理熱量（加熱全熱）[W]
            this.capHeating      = 0.0;//処理可能熱量（加熱全熱）[W]
            this.PPE_out = 0.0;
            this.watOutHS.copyAllState( this.watInHS );//20130110
            if( this.isRecord ){
               this.message.append( "(C)負荷率＜停止負荷率で停止");
            }
            this.mState = 3;//能力＝０→換気モードとする20130415
            break;

         case 1://  1_下限入力値固定
            break;

         case 2://  2_下限COP値固定
            break;

         case 3://  3_下限入力値と中間切片
            break;

         default:
         }

      }

      this.opeLoadS = this.opeCoolingLoadS;
      this.opeLoadT = this.opeCoolingLoadT;

      //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
   }



   private void heating_cal(){
      //室側の暖房計算
      double heatCapacity=0.0;
      double heatCapacityHPFCU=0.0;
      double heatCapacityFCU=0.0;
      double rCode0 = 0.0, rCode1 = 0.0;
      double HEXrate;
      double ele=0;
      double eleHPFCU=0;
      double eleFCU=0;

      HEXrate=in_EX_S;
```

```
      //バイパス制御 追加 20101212nino
      if( this.isHexbypass == true ){
         this.IAra = Psychrometrics.FNH( this.DBra, this.XGra );//20130109
         this.IAoa = Psychrometrics.FNH( this.DBoa, this.XGoa );
         if( this.IAra < this.IAoa && this.valInPID.getValue() > 0.001 && this.HEXselect == 1 ){//20130109
            HEXrate = 0.0;
         }
         if( this.DBra < this.DBoa && this.valInPID.getValue() > 0.001 && this.HEXselect == 2 ){//20130109
            HEXrate = 0.0;
         }
         if( this.IAra > this.IAoa && this.valInPID.getValue() < 0.001 && this.HEXselect == 1 ){//20130109
            HEXrate = 0.0;
         }
         if( this.DBra > this.DBoa && this.valInPID.getValue() < 0.001 && this.HEXselect == 2 ){//20130109
            HEXrate = 0.0;
         }
      }
      if( Airswc.isON( this.swcInOA )){//20101212nino
         this.DBin = (( this.in_Va_S - this.in_OA_S ) * this.DBra + ( this.DBra - ( this.DBra - this.DBoa ) * ( 1.0 -
HEXrate )) * this.in_OA_S ) / this.in_Va_S;//20130109
         this.XGin = (( this.in_Va_S - this.in_OA_S ) * this.XGra + ( this.XGra - ( this.XGra - this.XGoa ) * ( 1.0 -
HEXrate )) * this.in_OA_S ) / this.in_Va_S;//20130109
      }else{//20101212nino
         this.DBin = this.DBra;//20130109
         this.XGin = this.XGra;//20130109
      }
      this.WBin = Psychrometrics.FNWbtx(this.DBin,this.XGin);//20130109

      // System.out.println("H/DBin="+DBin+" "+XGin+"  ");
      // System.out.println("H/DBoa="+DBoa+" "+XGoa+"  ");
      // System.out.println("H/DBra="+DBra+" "+XGra+"  ");
      // System.out.println("  ");

      //上下限のチェック 20120821nino
      double twatIn = this.t_watInHS;
      double dbIn = this.DBin;

      if( twatIn > this.t_watInHS_UpperLimitH ){
         //熱源水入口温度＞上限の時　上限値の特性で運転
         if( this.isRecord ){
            this.message.append( "(C)熱源水入口温度＞"+this.t_watInHS_UpperLimitH+"上限→上限値で特性計算");
         }
         twatIn = this.t_watInHS_UpperLimitH;
      }
      if( twatIn < this.t_watInHS_LowerLimitH ){
         //熱源水入口温度＜下限の時　停止
         if( this.isRecord ){
            this.message.append( "(C)熱源水入口温度＜"+this.t_watInHS_LowerLimitH+"下限→停止");
         }

         this.mState = 3;//能力＝０→換気モードとする20130415
         this.calc_Stop_In0();
         return;
      }
      if( dbIn > this.db_airInRA_UpperLimitH ){
         //室内乾球温度＞上限の時　上限値の特性で運転
         if( this.isRecord ){
            this.message.append( "(C)室DB＞"+this.db_airInRA_UpperLimitH+"上限→上限値で特性計算");
         }
         dbIn = this.db_airInRA_UpperLimitH;
      }
      if( dbIn < this.db_airInRA_LowerLimitH ){
         //室内乾球温度＜下限の時　下限値の特性で運転
         if( this.isRecord ){
            this.message.append( "(C)室DB＜"+this.db_airInRA_LowerLimitH+"下限→下限値で特性計算");
         }
         dbIn = this.db_airInRA_LowerLimitH;
      }


      //能力補正
```

```
rCode0 = this.pacDB.getKhQtwi( twatIn, dbIn );//水温DB補正
rCode1 = this.pacDB.getKhf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正

heatCapacityHPFCU = rCode0 * rCode1 * this.in_Qh_S;

rCode0 = this.pacDB.getKhQtwif( twatIn, dbIn );//水温DB補正
//rCode1 = this.pacDB.getKhf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正

heatCapacityFCU = rCode0 * rCode1 * this.in_Qh_FCU;

//要求負荷　処理負荷
this.heatingLoadT = heatCapacityHPFCU * this.valInPID.getValue();
this.opeHeatingLoadT = this.heatingLoadT;

//ele
rCode0 = this.pacDB.getKhWtwi( twatIn, dbIn );//水温DB補正
rCode1 = this.pacDB.getKhwf( this.fRate_watInHS / this.fRate_watInHSh );//水量補正
eleHPFCU = rCode0 * rCode1 * this.in_PPEh_S;

rCode0 = this.pacDB.getKhWtwif( twatIn, dbIn );//水温DB補正
//rCode1 = this.pacDB.getKhwf( this.fRate_watInHS / this.fRate_watInHSh );//水量補正
eleFCU = rCode0 * rCode1 * this.in_PPEh_FCU;

if( heatCapacityHPFCU < this.heatingLoadT ){
   //要求負荷が能力を超えているとき
   heatCapacity = heatCapacityHPFCU;
   this.opeHeatingLoadT = heatCapacityHPFCU;
   ele = eleHPFCU;

}else if( heatCapacityFCU >= this.heatingLoadT ){
   //FCUだけで能力が足りている
   heatCapacity = heatCapacityFCU;
   //ele
   ele = eleFCU;

}else{
   //HPの運転が必要
   heatCapacity = heatCapacityHPFCU;
   //ele
   ele = eleHPFCU;

}

this.capHeating = heatCapacity;
this.opeHeatingLoadS = this.opeHeatingLoadT;


//吹き出し状態の計算

// System.out.println("heatCapacity ="+rCode1+" "+rCode2+" "+heatCapacity+"  ");

//IAin   = Psychrometrics.FNH( this.DBin, this.XGin );
//IAout1 = IAin + heatCapacity / this.in_Va_S * 1000.0;
//XGout1 = this.XGin;
//DBout1 = Psychrometrics.FNDbxh( XGout1, IAout1 );
double IAout;

this.RHin = Psychrometrics.FNRhtx( this.DBin, this.XGin );

if( this.in_HUM_on <= this.RHin || this.in_HUM_mx < 0.00001 ){
   IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
   IAout = IAin +this.opeHeatingLoadT / this.in_Va_S * 1000.0;
   this.DBout = this.DBin + this.opeHeatingLoadS / this.in_Va_S;//乾球温度
   this.XGout = Psychrometrics.FNXth( this.DBout, IAout );//絶対湿度
   this.CW_Wrate = 0.0;
   //    System.out.println("1 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
   //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
} else {
   IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
   IAout = IAin + this.opeHeatingLoadT / this.in_Va_S*1000.0;
   this.DBout = Psychrometrics.FNDbrh( this.in_HUM_rt, IAout);
   this.XGout = Psychrometrics.FNxtr( this.DBout, this.in_HUM_rt);
```

```
    this.CW_Wrate = ( this.XGout - this.XGin ) * this.GW;

    //定格加湿量をオーバーするときの処理
    if( this.CW_Wrate > this.in_HUM_mx ){
       this.XGout = this.in_HUM_mx / this.GW+ this.XGin;
       this.DBout =Psychrometrics.FNDbxh( this.XGout, IAout);
       this.CW_Wrate = this.in_HUM_mx;
    }
    if( this.XGout < this.XGin){
       IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
       IAout = IAin + this.opeHeatingLoadT / this.in_Va_S * 1000.0;
       this.DBout = this.DBin + this.opeHeatingLoadS / this.in_Va_S;//乾球温度
       this.XGout = Psychrometrics.FNXth( this.DBout, IAout);//絶対湿度
       this.CW_Wrate = 0.0;
    }
    //    System.out.println("2 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
    //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
}

if( this.capHeating <= 0. ){
   this.Rh = 0.;

}else{
   this.Rh = this.heatingLoadT / this.capHeating;

}

if( this.Rh > 1.0 ) {
   this.Rh = 1.0;
}

double Rp;

//       System.out.println("Rh="+Rh);
//低負荷領域の計算仕分け:Rh
if( this.Rh < this.in_run_stop ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
   case 3://  3_下限入力値と中間切片
      Rp = this.in_run_stop;
      break;

   default:
      Rp = 0;
   }
}else{
   Rp = this.Rh;
}


//入力
this.PPE_out = ele;

//低負荷領域の計算仕分け:PPE
if( this.Rh < this.in_run_stop && Rp > 0 ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      this.PPE_out = 0.0;
      break;

   case 1://  1_下限入力値固定
      //何もしない
      break;

   case 2://  2_下限COP値固定
      this.PPE_out *= ( this.Rh / Rp );
      break;
```

```
            case 3://  3_下限入力値と中間切片
               this.PPE_out *= ( 0.5 + this.Rh / Rp / 2. );
               break;

            default:
            }
         }else{
            //何もしない
         }


         //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);
         //System.out.println(" PAC 負荷率="+this.Rh+"    負荷律係数="+rCode1);


         //airOutEA
         //this.airOutEA.setFlowRate( this.fRate_watInHS );
         //this.airOutEA.setTempDB( ( -this.T_Loadout + this.PPE_out ) / this.fRate_watInHS * 1000. +
this.airInOA.getTempDB() );
         //watOutHS
         this.watOutHS.setFlowRate( this.fRate_watInHS );
         this.watOutHS.setTemp( ( -this.opeHeatingLoadT + this.PPE_out ) / this.fRate_watInHS / 4.18605 + this.t_watInHS );

         if( this.Rh < 0.0001 ){
            this.opeCoolingLoadS = 0.0;//実処理熱量（冷却顕熱）[W]
            this.opeCoolingLoadT = 0.0;//実処理熱量[W]
            this.capCooling      = 0.0;//処理可能熱量[W]
            this.opeHeatingLoadS = 0.0;//実処理熱量（加熱顕熱）[W]
            this.opeHeatingLoadT = 0.0;//実処理熱量[W]
            this.capHeating      = 0.0;//処理可能熱量[W]
            this.PPE_out = 0.0;
            this.watOutHS.copyAllState( this.watInHS );//20130110
            if( this.isRecord ){
               this.message.append( "(C)負荷率0で停止");
            }
            this.mState = 3;//能力＝０→換気モードとする20130415

         }else if( this.Rh < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
               this.opeCoolingLoadS = 0.0;//実処理熱量（冷却顕熱）[W]
               this.opeCoolingLoadT = 0.0;//実処理熱量（冷却全熱）[W]
               this.capCooling      = 0.0;//処理可能熱量（冷却全熱）[W]
               this.opeHeatingLoadS = 0.0;//実処理熱量（加熱顕熱）[W]
               this.opeHeatingLoadT = 0.0;//実処理熱量（加熱全熱）[W]
               this.capHeating      = 0.0;//処理可能熱量（加熱全熱）[W]
               this.PPE_out = 0.0;
               this.watOutHS.copyAllState( this.watInHS );//20130110
               if( this.isRecord ){
                  this.message.append( "(C)負荷率＜停止負荷率で停止");
               }
               this.mState = 3;//能力＝０→換気モードとする20130415
               break;

            case 1://  1_下限入力値固定
               break;

            case 2://  2_下限COP値固定
               break;

            case 3://  3_下限入力値と中間切片
               break;

            default:
            }

      }

         //    this.watInCW.setFlowRate(CW_Wrate);
         this.opeLoadS = this.opeHeatingLoadS;
         this.opeLoadT = this.opeHeatingLoadT;
```

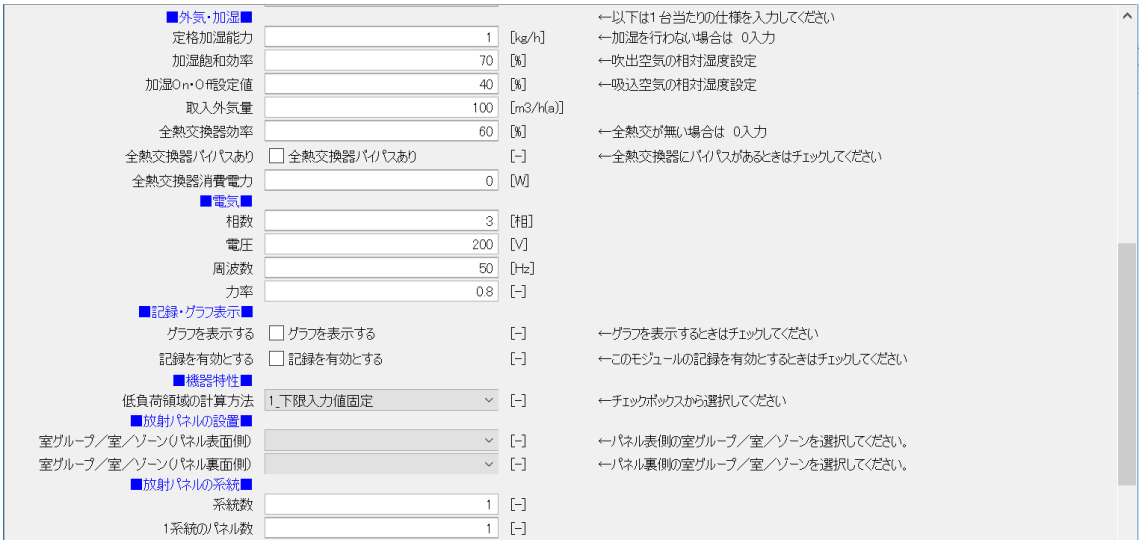
```
    }

    private void ventilation_cal(){
        //double heatCapacity=0.0,rCode1 = 0.0,rCode2 = 0.0,rCode3 = 0.0,rCode4 = 0.0,rCode5 = 0.0,rCode6 = 0.0,rCode7 =
0.0,rCode8 = 0.0;
        //double IAra,IAout1,DBout1,XGout1,S_heat,T_heat;
        double HEXrate;

        HEXrate = this.in_EX_S;

        //換気の時は全熱交換器は停止とする
        HEXrate = 0.0;

        //外気と室空気の混合空気の計算
        if( Airswc.isON( this.swcInOA )){//20101212nino
            this.DBin = (( this.in_Va_S - this.in_OA_S ) * this.DBra + (( this.DBoa - this.DBra ) * ( 1.0 - HEXrate ) +
this.DBra ) * this.in_OA_S ) / this.in_Va_S;
            this.XGin = (( this.in_Va_S - this.in_OA_S ) * this.XGra + (( this.XGoa - this.XGra ) * ( 1.0 - HEXrate ) +
this.XGra ) * this.in_OA_S ) / this.in_Va_S;
        }else{//20101212nino
            DBin = DBra;
            XGin = XGra;
        }
        this.WBin = Psychrometrics.FNWbtx( this.DBin, this.XGin );
        //    System.out.println("C/DBin="+DBin+" "+XGin+"  ");
        //    System.out.println("C/DBoa="+DBoa+" "+XGoa+"  ");
        //    System.out.println("C/DBra="+DBra+" "+XGra+"  ");
        //    System.out.println("  ");

        //    System.out.println("DBra="+DBra+" "+XGra+"  ");

        this.opeCoolingLoadS = 0.0;
        this.opeCoolingLoadT = 0.0;
        this.opeHeatingLoadS = 0.0;
        this.opeHeatingLoadT = 0.0;
        this.opeLoadS = 0.0;
        this.opeLoadT = 0.0;

        this.DBout = this.DBin;//乾球温度
        this.XGout = this.XGin;//絶対湿度
        this.CW_Wrate = 0.0;

    }


}
```

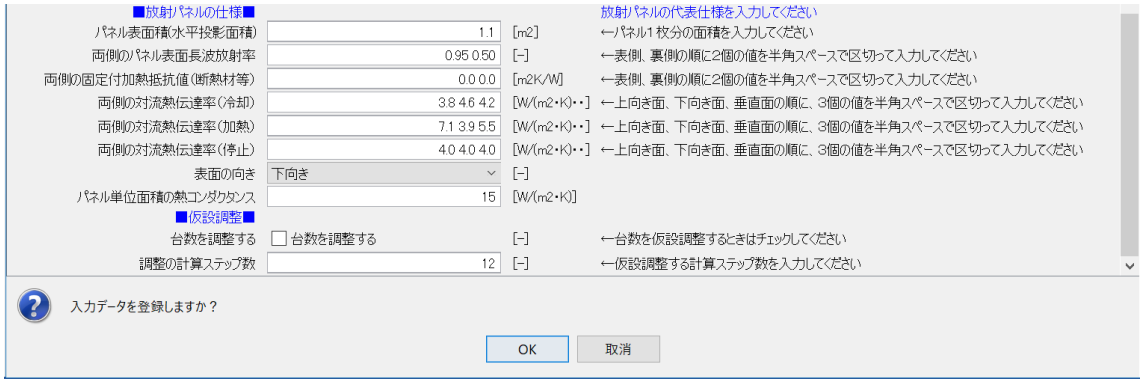
「PAC FCUEHPRAD2015」（場所：設備 2015／個別分散 2015／）

モジュール名	PAC FCUEHPRAD2015
クラス	FCUwithEHPRADModule2015 RadPanelModule2014

（1）入力画面

・スペック





（2）モジュールの概要

FCU+水冷 EHP(PAFMAC)+放射パネルのモジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

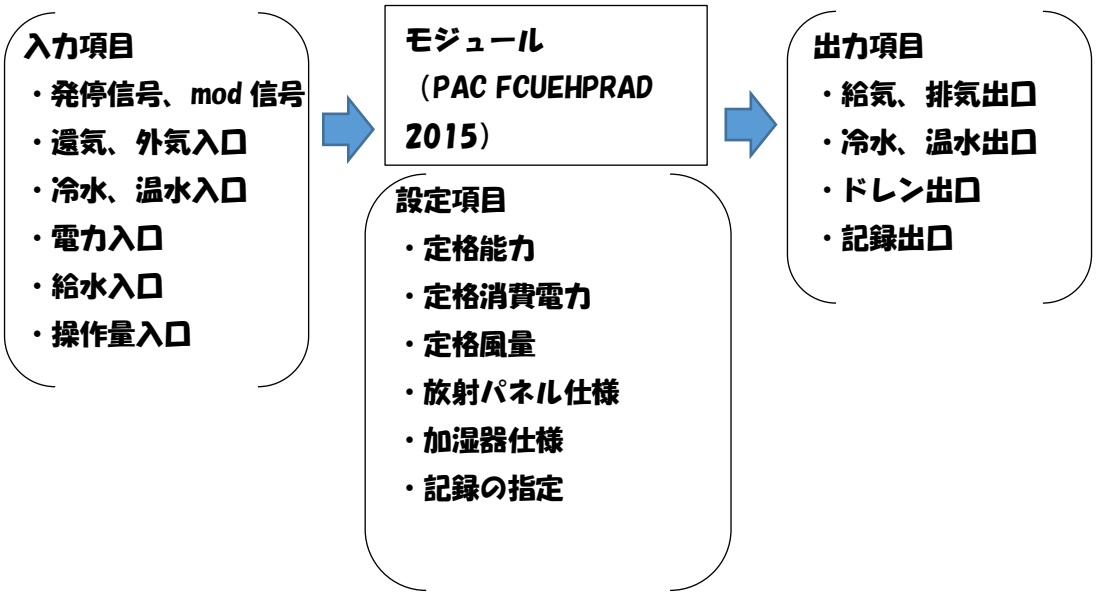


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	室グループ／室／ゾーン	String	zoneAirforSSinside	#RoomGroup	[-]	―	―		←室グループ／室／ゾーンを選択してください。
2	台数	double	in_dai	1	[台]	―	0		
3	機器番号	String	m_no			―	―		
4	機器種別	String	m_ty		[-]	―	―		←チェックボックスから選択してください
5	機器型式	String	m_kt			―	―		
6	■定格能力など■					―	―		←以下は１台当たりの仕様を入力してください
7	定格冷房能力	double	de_Qc_FCU_RAD	3.9	[kW]	―	0		←FCU+放射(入口 7℃)
8	温水時冷房能力	double	de_Qc_HP_RAD	1.6	[kW]	―	0		←HP+R 放射 （入口 45℃）
9	放射冷房能力	double	de_Qc_RAD	0.7	[kW]	―	0		←放射のみ
10	定格暖房能力	double	de_Qh_FCU_RAD	3.9	[kW]	―	0		←FCU+放射(入口 45℃)
11	冷水時暖房能力	double	de_Qh_HP_RAD	1.8	[kW]	―	0		←HP+放射(入口 7℃)
12	放射暖房能力	double	de_Qh_RAD	0.9	[kW]	―	0		←放射のみ
13	■定格入力など■					―	―		←以下は１台当たりの仕様を入力してください
14	定格冷房入力(電力)	double	de_PPEc_FCU_RAD	0.09	[kW]	―	0		←HP+FCU
15	温水時冷房入力(電力)	double	de_PPEc_HP_RAD	0.66	[kW]	―	0		←FCU のみ
16	放射冷房入力(電力)	double	de_PPEc_RAD		[kW]	―	0		
17	定格暖房入力(電力)	double	de_PPEh	0.09	[kW]	―	0		←HP+FCU

			_FCU_RAD						
18	冷水時暖房入力(電力)	double	de_PPEh_HP_RAD	0.45	[kW]	―	0		←FCU のみ
19	放射暖房入力(電力)	double	de_PPEh_RAD		[kW]	―	0		
20	定格冷温水流量	double	de_fRate_watInHS	10	[L/min(w)]	―	0		←熱源水の定格入口温度は冷房 7℃、暖房 45℃です
21	機器起動停止負荷率	double	in_run_stop	30	[%]	100	0		←部分負荷率を入力してください
22	定格風量	double	de_Va_S	480	[m3/h(a)]	―	0		
23	定格ファン消費電力	double	de_PPEa_S	0.06	[kW]	―	0		
24	放射定格水量	double	de_fRate_watRAD	7.2	[L/min(w)]	―	0		←放射用流量
25	定格ポンプ消費電力	double		0.03	[kW]	―	0		←放射用送水ポンプ
26	■外気・加湿■					―	―		←以下は１台当たりの仕様を入力してください
27	定格加湿能力	double	de_HUM_mx	1	[kg/h]	―	0		←加湿を行わない場合は　0 入力
28	加湿飽和効率	double	de_HUM_rt	70	[%]	―	0		←吹出空気の相対湿度設定
29	加湿 On・Off 設定値	double	de_HUM_on	40	[%]	―	0		←吸込空気の相対湿度設定
30	取入外気量	double	de_OA_S	100	[m3/h(a)]	―	0		
31	全熱交換器効率	double	de_EX_S	60	[%]	100	0		←全熱交が無い場合は　0 入力
32	全熱交換器バイパスあり	boolean	de_EX_PE	FALSE	[-]	―	―		←全熱交換器にバイパスがあるときはチェックしてください
33	全熱交換器消費電力	double		0	[W]	―	0		
34	■電気■					―	―		
35	相数	int	phase	3	[相]	―	1		
36	電圧	double	voltage	200	[V]	―	100		

37	周波数	double	frequency	50	[Hz]	60	50		
38	力率	double	powerFactor	0.8	[-]	1	0		
39	■記録・グラフ表示■					—	—		
40	グラフを表示する	boolean	isGVisible	FALSE	[-]	—	—		←グラフを表示するときはチェックしてください
41	記録を有効とする	boolean	isRecord	FALSE	[-]	—	—		←このモジュールの記録を有効とするときはチェックしてください
42	■機器特性■					—	—		
43	低負荷領域の計算方法	String	calcTypeLowerRangeLoad	0_発停運転、1_下限入力値固定、2_下限COP値固定、3_下限入力値と中間切片	[-]	—	—		←チェックボックスから選択してください
44	■放射パネルの設置■					—	—		
45	室グループ／室／ゾーン（パネル表面側）	String	grzName	#RoomGroup	[-]	—	—		←パネル表側の室グループ／室／ゾーンを選択してください。
46	室グループ／室／ゾーン（パネル裏面側）	String	grzNameC	#RoomGroup	[-]	—	—		←パネル裏側の室グループ／室／ゾーンを選択してください。
47	■放射パネルの系統■					—	—		<html><font color=blue>
48	系統数	int	npr	1	[-]	—	0		
49	1系統のパネル数	int	ns	1	[-]	—	0		
50	■放射パネルの仕様■					—	—		<html><font color=blue>放射パネルの代表仕様を入力してください
51	パネル表面積(水平投影面積)	double	area_p	1.1	[m2]	—	0		←パネル１枚分の面積を入力してください
52	両側のパネル表面長波放射率	String	Npr[]	0.95 0.50	[-]	—	—		←表側、裏側の順に２個の値を半角スペースで区切って入力してください
53	両側の固定付加熱抵抗値（断熱材等）	String	Rf[]	0.0 0.0	[m2K/W]	—	—		←表側、裏側の順に２個の値を半角スペースで区切って入力してください

54	両側の対流熱伝達率（冷却）	String	alpha_c[0]	3.8 4.6 4.2	[W/(m2･K)・・]	―	―		←上向き面、下向き面、垂直面の順に、３個の値を半角スペースで区切って入力してください
55	両側の対流熱伝達率（加熱）	String	alpha_c[1]	7.1 3.9 5.5	[W/(m2･K)・・]	―	―		←上向き面、下向き面、垂直面の順に、３個の値を半角スペースで区切って入力してください
56	両側の対流熱伝達率（停止）	String	alpha_c[2]	4.0 4.0 4.0	[W/(m2･K)・・]	―	―		←上向き面、下向き面、垂直面の順に、３個の値を半角スペースで区切って入力してください
57	表面の向き	String	Face[]	下向き、上向き、垂直面	[-]	―	―		
58	パネル単位面積の熱コンダクタンス	double	Kt_w_otp	15	[W/(m2･K)]	―	0		
59	■仮設調整■					―	―		
60	台数を調整する	boolean	isAdjust2012	FALSE	[-]	―	―		←台数を仮設調整するときはチェックしてください
61	調整の計算ステップ数	int	numAdjustSteps	12	[-]	―	1		←仮設調整する計算ステップ数を入力してください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	外気導入信号入口	L1_swcInOA	swcInOA	-	制御	OnOff 信号	入口	
4	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
5	PID モード信号入口	L1_modInPID	modInPID	-	制御	制御モード	入口	
6	冷暖切り替えモード信号入口	L0_modInPIPE	modInPIPE	-	制御	制御モード	入口	
7	還気入口	L0_airInRA	airInRA	-	状態	空気	入口	
8	給気出口	L0_airOutSA	airOutSA	-	状態	空気	出口	
9	外気入口	L0_airInOA	airInOA	-	状態	空気	入口	
10	排気出口	L0_airOutEA	airOutEA	-	状態	空気	出口	
11	冷水入口	L0_watInC	watInC	-	状態	水	入口	
12	冷水出口	L0_watOutC	watOutC	-	状態	水	出口	
13	温水入口	L0_watInH	watInH	-	状態	水	入口	
14	温水出口	L0_watOutH	watOutH	-	状態	水	出口	
15	給水入口	L0_watInCW	watInCW	-	状態	水	入口	
16	ドレン出口	L0_watOutD	watOutD	-	状態	水	出口	
17	操作量入口	L0_valInCtrl	valInCtrl	-	値	値	入口	
18	電力入口	L0_eleIn	eleIn		状態	電気	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	FCUHPRP_Message#-#-	メッセージ	—	メッセージ
2	FCUHPRP_処理顕熱量合計#W#熱量	FCUHPRP_処理顕熱量合計	W	My
3	FCUHPRP_処理全熱量合計#W#熱量	FCUHPRP_処理全熱量合計	W	My
4	FCUHPRP_冷却要求全熱量合計#W#熱量	FCUHPRP_冷却要求全熱量合計	W	My
5	FCUHPRP_加熱要求全熱量合計#W#熱量	FCUHPRP_加熱要求全熱量合計	W	My
6	FCUHPRP_消費電力#W#消費電力	FCUHPRP_消費電力	W	エネルギー消費
7	FCUHPRP_冷却可能熱量/最大#W#熱量	FCUHPRP_冷却可能熱量/最大	W	My
8	FCUHPRP_加熱可能熱量/最大#W#熱量	FCUHPRP_加熱可能熱量/最大	W	My
9	FCUHPRP_処理全熱量合計負荷#W#-	FCUHPRP_処理全熱量合計負荷	W	負荷
10	FCUHPRP_PID 操作量#-#-	FCUHPRP_PID 操作量	—	入口
11	FCUHPRP_watInC 温度#℃#温度	FCUHPRP_watInC 温度	℃	入口
12	FCUHPRP_watInH 温度#℃#温度	FCUHPRP_watInH 温度	℃	入口
13	FCUHPRP_watOutC 温度#℃#温度	FCUHPRP_watOutC 温度	℃	出口
14	FCUHPRP_watOutH 温度#℃#温度	FCUHPRP_watOutH 温度	℃	出口
15	FCUHPRP_watOutC 流量#g/s#質量流量	FCUHPRP_watOutC 流量	g/s	出口
16	FCUHPRP_watOutH 流量#g/s#質量流量	FCUHPRP_watOutH 流量	g/s	出口
17	FCUHPRP_COP#-#COP	FCUHPRP_COP	—	My
18	FCUHPRP_室吹出乾球温度#℃#温度	FCUHPRP_室吹出乾球温度	℃	出口
19	FCUHPRP_室吹出絶対湿度#g/g#湿度	FCUHPRP_室吹出絶対湿度	g/g	出口
20	FCUHPRP_室吹出風量#g/s#質量流量	FCUHPRP_室吹出風量	g/s	出口
21	FCUHPRP_ドレン量#g/s#質量流量	FCUHPRP_ドレン量	g/s	出口
22	FCUHPRP_加湿給水量#g/s#質量流量	FCUHPRP_加湿給水量	g/s	入口
23	FCUHPRP_室乾球温度#℃#温度	FCUHPRP_室乾球温度	℃	入口
24	FCUHPRP_室絶対湿度#g/g#湿度	FCUHPRP_室絶対湿度	g/g	入口
25	FCUHPRP_吸込乾球温度#℃#温度	FCUHPRP_吸込乾球温度	℃	入口
26	FCUHPRP_吸込絶対湿度#g/g#湿度	FCUHPRP_吸込絶対湿度	g/g	入口
27	FCUHPRP_調整冷却能力#W#熱量	FCUHPRP_調整冷却能力	W	調整
28	FCUHPRP_調整加熱能力#W#熱量	FCUHPRP_調整加熱能力	W	調整
29	FCUHPRP_調整冷却台数#-#台数	FCUHPRP_調整冷却台数	—	調整
30	FCUHPRP_調整加熱台数#-#台数	FCUHPRP_調整加熱台数	—	調整
31	FCUHPRP_調整ステップ平均冷却台数#-#台数	FCUHPRP_調整ステップ平均冷却台数	—	調整

32	FCUHPRP_調整ステップ平均加熱台数#-#台数	FCUHPRP_調整ステップ平均加熱台数	—	調整
33				

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.essentials.utility.Psychrometrics;
import jp.or.ibec.best.log.BestLogger;
import jp.or.ibec.best.message.BestEngineMessageHandler;
import jp.or.ibec.best.message.BestEngineMessageParam;
import jp.or.ibec.best.message.constants.SystemMessageConstants;

/**
 * @author HIROSHI NINOMIYA/201508
 *
 * FCU+水冷EHP(PAFMAC)+放射パネル
 *
 * 201508
 *
 */
public class FCUwithEHPRADModule2015 extends AbstractBestModule implements
IBestMetaModule  {

   /**
    * 論理アドレス
    */
   private final String moduleName="(FCUwithEHPRADModule2015)";

   private final String S_NODE_airInOA  = "L0_airInOA";//外気
   private final String S_NODE_airOutEA = "L0_airOutEA";//外気
   private final String S_NODE_watInC   = "L0_watInC";   //FCU入口冷水
   private final String S_NODE_watInH   = "L0_watInH";   //FCU入口温水
   private final String S_NODE_watOutC  = "L0_watOutC";//FCU出口冷水
   private final String S_NODE_watOutH  = "L0_watOutH";//FCU出口温水
   //private final String S_NODE_watIn    = "L0_watInHS";//熱源水入口
   //private final String S_NODE_watOut   = "L0_watOutHS";//熱源水出口

   private final String S_NODE_airInRA  = "L0_airInRA";//室内吸込口
   private final String S_NODE_airOutSA = "L0_airOutSA";//室内吹出し

   private final String S_NODE_watOutD  = "L0_watOutD";//コイル出口ドレン
   private final String S_NODE_watInCW  = "L0_watInCW";//watInCW---090407追加**************

   private final String S_NODE_valInPID = "L0_valInCtrl";//供給熱量 from PID           : in

   private final String S_NODE_eleIn    = "L0_eleIn";//電力

   private final String C_NODE_swcIn    = "L1_swcIn";//運転状態：off/on
   private final String C_NODE_swcInOA  = "L1_swcInOA";//外気運転状態：off/on
   private final String C_NODE_modIn    = "L1_modIn";//空調モード：停止/冷却/加熱
   private final String C_NODE_modInPID = "L1_modInPID";//空調モード：停止/冷却/加熱
   private final String C_NODE_modInPIPE= "L1_modInPIPE";//冷水／温水切替

   private final String R_NODE = "L2_recOut";
```

```
    /**
     * 外部定義
     */
    private final String SPEC_name          = "名称";

    private final String SPEC_grzName  = "室グループ／室／ゾーン";

    private final String SPEC_m_no          = "機器番号";
    private final String SPEC_m_ty          = "機器種別";
    private final String SPEC_m_kt          = "機器型式";

    private final String SPEC_de_Qc_FCU_RAD = "定格冷房能力[W]";//FCU+RAD
    private final String SPEC_de_Qc_HP_RAD  = "温水時冷房能力[W]";//HP+RAD
    private final String SPEC_de_Qc_RAD     = "放射冷房能力[W]";
    private final String SPEC_de_Qh_FCU_RAD = "定格暖房能力[W]";//FCU+RAD
    private final String SPEC_de_Qh_HP_RAD  = "冷水時暖房能力[W]";//HP+RAD
    private final String SPEC_de_Qh_RAD     = "放射暖房能力[W]";
//  private final String SPEC_out_Qh_L      = "低温暖房能力[W]";//kW

    private final String SPEC_de_PPEc_FCU_RAD = "定格冷房入力(電力)[W]";
    private final String SPEC_de_PPEc_HP_RAD  = "温水時冷房入力(電力)[W]";
    private final String SPEC_de_PPEc_RAD     = "放射冷房入力(電力)[W]";
    private final String SPEC_de_PPEh_FCU_RAD = "定格暖房入力(電力)[W]";
    private final String SPEC_de_PPEh_HP_RAD  = "冷水時暖房入力(電力)[W]";
    private final String SPEC_de_PPEh_RAD     = "放射暖房入力(電力)[W]";
//  private final String SPEC_out_PPEh_L    = "低温暖房入力(電力)[W]";//kW
/*
    private final String SPEC_out_CLEa_R    = "クランクケースヒータ(運転時)";//W
    private final String SPEC_out_CLEa_S    = "クランクケースヒータ(停止時)";//W
    private final String SPEC_out_TKEa_R    = "定格待機電力(運転時)";//W
    private final String SPEC_out_TKEa_T    = "定格待機電力(待機時)";//W
    private final String SPEC_out_TKEa_S    = "定格待機電力(停止時)";//W
*/
    private final String SPEC_de_fRate_watInHS   = "定格冷温水流量[g/s]";
    private final String SPEC_de_fRate_watRAD    = "放射定格水量[g/s]";

    private final String SPEC_in_run_stop  = "機器起動停止負荷率[-]";//[%]
    private final String SPEC_Phase       = "相数[-]";
    private final String SPEC_Voltage     = "電圧[V]";
    private final String SPEC_Frequency   = "周波数[Hz]";
    private final String SPEC_PowerFactor = "力率[-]";
    //
    private final String SPEC_isGVisible = "グラフを表示する";
    private final String SPEC_maxItemCount = "最大同時表示ステップ数";
    private final String SPEC_isRecord = "記録を有効とする";

    private final String SPEC_CalcTypeLowerRangeLoad = "低負荷領域の計算方法";

    private final String SPEC_isAdjust2012   = "台数を調整する";
    private final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";

    private StringBuffer message= new StringBuffer();

    /**
     * 外部定義
     */

    private final String SPEC_de_dai       = "台数[-]";

    private final String SPEC_de_Va_S         = "定格風量[g/s]";//m3/h
    private final String SPEC_de_PPEa_S    = "定格ファン消費電力[W]";//W
//  private final String SPEC_in_TKEa_R     = "定格待機電力(運転時)";//W
//  private final String SPEC_in_THEa_S     = "定格待機電力(停止時)";//W

    private final String SPEC_de_HUM_mx       = "定格加湿能力[g/s]";//kg/h
    private final String SPEC_de_HUM_rt       = "加湿飽和効率[-]";//[%]
    private final String SPEC_de_HUM_on       = "加湿On・Off設定値[-]";//[%]
    private final String SPEC_de_OA_S         = "取入外気量[g/s]";//m3/h
    private final String SPEC_de_EX_S         = "全熱交換器効率[-]";//[%]
    private final String SPEC_isHexbypass     = "全熱交換器バイパスあり[-]";
```

```
    private final String SPEC_de_EX_PE       = "全熱交換器消費電力[W]";


    /**
     * 変数（外部定義に対応）
     */

    private String name             ;// "機器名称";
    private String m_no             ;//= "機器番号";
    private String m_ty             ;//= "機器種別";
    private String m_kt             ;//= "機器型式";
    private double de_Qc_FCU_RAD    ;//= "定格冷房能力";//kW
    private double de_Qc_HP_RAD     ;//= "温水時冷房能力";//kW
    private double de_Qc_RAD        ;//= "放射冷房能力";//kW
    private double de_PPEc_FCU_RAD  ;//= "定格冷房入力(電力)";//kW
    private double de_PPEc_HP_RAD   ;//= "温水時冷房入力(電力)";//kW
    private double de_PPEc_RAD      ;//= "放射冷房入力(電力)";//kW
    private double de_Qh_FCU_RAD    ;//= "定格暖房能力";//kW
    private double de_Qh_HP_RAD     ;//= "冷水時暖房能力";//kW
    private double de_Qh_RAD        ;//= "放射暖房能力";//kW
// private double out_Qh_L     ;//= "低温暖房能力";//kW
    private double de_PPEh_FCU_RAD  ;//= "定格暖房入力(電力)";//kW
    private double de_PPEh_HP_RAD   ;//= "冷水時暖房入力(電力)";//kW
    private double de_PPEh_RAD      ;//= "放射暖房入力(電力)";//kW
// private double out_PPEh_L    ;//= "低温暖房入力(電力)";//kW
/*
    private double out_CLEa_R=0.0    ;//= "クランクケースヒータ(運転時)";//W
    private double out_CLEa_S=0.0    ;//= "クランクケースヒータ(停止時)";//W
    private double out_TKEa_R =0.0   ;//= "定格待機電力(運転時)";//W
    private double out_TKEa_T =0.0   ;//= "定格待機電力(待機時)";//W
    private double out_TKEa_S =0.0  ;//= "定格待機電力(停止時)";//W
*/
    private double t_watInHS;// = this.watInHS.getTemp();

    private double fRate_watInHS ;//熱源水水量[g/s]
    private double de_fRate_watInHS ;//熱源水定格水量[g/s]
    private double de_fRate_watRAD ;//放射定格水量[g/s]

    private double in_run_stop    ;//= "機器起動停止負荷率";//[%]
    private int phase;          //相数[-]
    private double voltage;     //電圧[V]
    private double frequency;  //周波数[Hz]
    private double powerFactor;   //力率[-]

    private boolean isGVisible = false;//このグラフを表示する=true
    private int  maxItemCount = 100;
    private boolean isRecord   = false;//記録を有効とする=true


    /**
     * 変数（外部定義に対応）
     */


    private double in_dai          ;//= "台数";

    private double de_Va_S         ;//= "定格風量";//m3/h
    private double de_PPEa_S       ;//= "定格ファン消費電力";//W
    private double de_TKEa_R=0.0  ;//= "定格待機電力(運転時)";//W
    private double de_TKEa_S=0.0  ;//= "定格待機電力(停止時)";//W

    private double de_HUM_mx          ;//= "定格加湿能力";//kg/h
    private double de_HUM_rt          ;//= "加湿飽和効率";//[%]
    private double de_HUM_on          ;//= "加湿On・Off設定値";//[%]
    private double de_OA_S         ;//= "取入外気量";//m3/h
    private double de_EX_S         ;//= "全熱交換器効率";//[%]
    private boolean isHexbypass = false;//熱交バイパスあり=true
    private double de_EX_PE    ;//= "全熱交換器消費電力[W]";

    private BestAir airInOA  = null;
    private BestAir airOutEA   = null;
```

```
    private BestWater watInC  = null;
    private BestWater watOutC = null;
    private BestWater watInH  = null;
    private BestWater watOutH = null;
    private BestWater watInHS = null;
    private BestWater watOutHS = null;

    private BestElectricity eleIn = null;  //電力

    private PACDBManager pacDB=null;

    private int modIn  ;
    private int modInPID  ;
    private int modInPIPE ;
    private int swcIn ;
    private int swcInOA ;
    private int mState;  //停止、冷房、暖房フラグ



    private BestAir airInRA   = null;
    private BestAir airOutSA  = null;         //給気
    private BestWater watOutD = null;
    private BestWater watInCW = null;

    private BestValue valInPID = null;

// private PACDBManager pacDB=null;


    /**
     * 出力（建物）
     */
    private final String RECORD_message      = "FCUHPRP_Message#-#-";
    private final String RECORD_opeLoadS     = "FCUHPRP_処理顕熱量合計#W#熱量";
    private final String RECORD_opeLoadT     = "FCUHPRP_処理全熱量合計#W#熱量";
    private final String RECORD_coolingLoadT = "FCUHPRP_冷却要求全熱量合計#W#熱量";
    private final String RECORD_heatingLoadT = "FCUHPRP_加熱要求全熱量合計#W#熱量";

    private final String RECORD_ID4        = "FCUHPRP_消費電力#W#消費電力";
    private final String RECORD_capCooling = "FCUHPRP_冷却可能熱量/最大#W#熱量";
    private final String RECORD_capHeating = "FCUHPRP_加熱可能熱量/最大#W#熱量";

    private final String RECORD_qT         = "FCUHPRP_処理全熱量合計負荷#W#-";
    private final String RECORD_pidVal     = "FCUHPRP_PID操作量#-#-";

    private final String RECORD_T_watInC   = "FCUHPRP_watInC温度#℃#温度";
    private final String RECORD_T_watInH   = "FCUHPRP_watInH温度#℃#温度";
    private final String RECORD_T_watOutC  = "FCUHPRP_watOutC温度#℃#温度";
    private final String RECORD_T_watOutH  = "FCUHPRP_watOutH温度#℃#温度";
    private final String RECORD_M_watOutC  = "FCUHPRP_watOutC流量#g/s#質量流量";
    private final String RECORD_M_watOutH  = "FCUHPRP_watOutH流量#g/s#質量流量";

    private final String RECORD_COP = "FCUHPRP_COP#-#COP";


    /**
     * 出力（建物）
     */

    private final String RECORD_DBout     = "FCUHPRP_室吹出乾球温度#℃#温度";
    private final String RECORD_XGout     = "FCUHPRP_室吹出絶対湿度#g/g#湿度";
    private final String RECORD_GW        = "FCUHPRP_室吹出風量#g/s#質量流量";
    private final String RECORD_D_Wrate   = "FCUHPRP_ドレン量#g/s#質量流量";
    private final String RECORD_ID8i      = "FCUHPRP_加湿給水量#g/s#質量流量";

    private final String RECORD_DBairInRM = "FCUHPRP_室乾球温度#℃#温度";
    private final String RECORD_XairInRM  = "FCUHPRP_室絶対湿度#g/g#湿度";
    private final String RECORD_DBairIn   = "FCUHPRP_吸込乾球温度#℃#温度";
    private final String RECORD_XairIn    = "FCUHPRP_吸込絶対湿度#g/g#湿度";
```

```
    private final String RECORD_out_Qc_Adjust = "FCUHPRP_調整冷却能力#W#熱量";
    private final String RECORD_out_Qh_Adjust = "FCUHPRP_調整加熱能力#W#熱量";
    private final String RECORD_in_daicAdjust = "FCUHPRP_調整冷却台数#-#台数";
    private final String RECORD_in_daihAdjust = "FCUHPRP_調整加熱台数#-#台数";
    private final String RECORD_avecdaiAdjust = "FCUHPRP_調整ステップ平均冷却台数#-#台数";
    private final String RECORD_avehdaiAdjust = "FCUHPRP_調整ステップ平均加熱台数#-#台数";

    private double DBin;          //室内乾球温度[℃]
    private double WBin;          //室内湿球温度[℃]
    private double XGin;
    private double IAin;
    private double RHin;

    private double DBoa;          //外気乾球温度[℃]
    private double WBoa;          //外気湿球温度[℃]
    private double XGoa;
    private double IAoa;

    private double DBra;          //室内機吸込乾球温度[℃]
    private double WBra;          //室内機吸込湿球温度[℃]
    private double XGra;
    private double IAra;

    private double DBout;            //乾球温度[℃]
    private double XGout;            //絶対湿度[kg/kg.DA]

    private double GW;               //風量(g/s)
    private double D_Wrate;       //ドレン水量(g/s)
    private double CW_Wrate;         //加湿水量(g/s)

    private double PPE_out;
    private double E_PPE_out;
    private double PPE_in;
    private double E_PPE_in;

    private double opeLoadS;//処理熱量（顕熱）[W]
    private double opeLoadT;//処理熱量（全熱）[W]
    private double opeCoolingLoadS;//処理熱量（顕熱）[W]
    private double opeCoolingLoadT;//処理熱量（全熱）[W]
    private double opeCoolingLoadAIR;//送風処理熱量（全熱）[W]
    private double opeCoolingLoadRAD;//放射処理熱量（全熱）[W]
    private double opeHeatingLoadS;//処理熱量（顕熱）[W]
    private double opeHeatingLoadT;//処理熱量（全熱）[W]
    private double opeHeatingLoadAIR;//送風処理熱量（全熱）[W]
    private double opeHeatingLoadRAD;//放射処理熱量（全熱）[W]
//******************090725-s
    private double saijohatsu;
// private double T_C_heat;
// private int u40=0;

//******************090725-e
    private int dbflg=0;
    private String path="EHP";
    private String[] filenames= new String[1];
    private String equipmentName;
    //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    private int HEXselect=1;//1=エンタルピ基準  2=温度基準
    //+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    //低負荷領域の計算方法
    private String calcTypeLowerRangeLoad = null;//"低負荷領域の計算方法";
    private int num_calcTypeLowerRangeLoad;//"低負荷領域の計算方法";

    /**
     * その他変数
     */

    private double Rc;               //部分負荷率
    private double Rh;               //部分負荷率

    private double capCooling;//処理可能熱量（冷却全熱）[W]
```

```
    private double capHeating;//処理可能熱量（加熱全熱）[W]

    private double coolingLoadT;//要求処理熱量（冷却全熱）[W]
    private double heatingLoadT;//要求処理熱量（加熱全熱）[W]

    private String kiki_file;

    private double COP;  //成績係数(-)

    //仮設調整モード2012

    private boolean isAdjust2012  = false;//true="台数を調整する";//
    //private boolean isAdjust2012 = false;//20120406nino 建築学会大会空衛学会OS論文検討用＝true
    private int numAdjustSteps;//"調整の計算ステップ数";
    private LinkedList<Double> daiListc = null;//必要台数
    private LinkedList<Double> daiListh = null;//必要台数
    private LinkedList<Double> rcList = null;//必要台数
    private LinkedList<Double> rhList = null;//必要台数
    private double maxcAdjustdai = 1;
    private double maxhAdjustdai = 1;
    private double avecdaiAdjust = 0;
    private double avehdaiAdjust = 0;
    private double in_daicAdjust = 1;//= "調整台数";
    private double in_daihAdjust = 1;//= "調整台数";
    private double in_run_stop_Num;//= "機器起動停止負荷率"台数補正;

// private double max_rcAdjust = 1.;
// private double max_rhAdjust = 1.;
// private double ave_rcAdjust = 0;
// private double ave_rhAdjust = 0;
// private double out_rcAdjust = 1.;//= "調整率冷房";
// private double out_rhAdjust = 1.;//= "調整率暖房";

    private double de_Qc_FCU_RAD_1   ;//= "定格冷房能力";//kW
    private double de_Qc_HP_RAD_1    ;//= "温水時冷房能力";//kW
    private double de_Qc_RAD_1       ;//= "放射冷房能力";//kW
    private double de_PPEc_FCU_RAD_1 ;//= "定格冷房入力(電力)";//kW
    private double de_PPEc_HP_RAD_1  ;//= "温水時冷房入力(電力)";//kW
    private double de_PPEc_RAD_1     ;//= "放射冷房入力(電力)";//kW
    private double de_Qh_FCU_RAD_1   ;//= "定格暖房能力";//kW
    private double de_Qh_HP_RAD_1    ;//= "冷水時暖房能力";//kW
    private double de_Qh_RAD_1       ;//= "放射暖房能力";//kW
// private double out_Qh_L_1        ;//= "低温暖房能力";//kW
    private double de_PPEh_FCU_RAD_1 ;//= "定格暖房入力(電力)";//kW
    private double de_PPEh_HP_RAD_1  ;//= "冷水時暖房入力(電力)";//kW
    private double de_PPEh_RAD_1     ;//= "放射暖房入力(電力)";//kW
// private double out_PPEh_L_1      ;//= "低温暖房入力(電力)";//kW

// private double out_Qc_Adjust;//= "調整冷却能力";
// private double out_Qh_Adjust;//= "調整加熱能力";

    private double fRate_watInHSc ;//水量[g/s]
    private double fRate_watInHSh ;//水量[g/s]

    private double de_Va_S_1      ;//= "定格風量";//m3/h
    private double de_PPEa_S_1    ;//= "定格ファン消費電力";//W

    private double de_HUM_mx_1    ;//= "定格加湿能力";//kg/h

    private ZoneAirforSSinsideModule201502 zoneAirforSSinside = null;
    private String grzName = null;
    private boolean isUseZoneAir = false;

    private double t_watInHS_UpperLimitC;
    private double t_watInHS_LowerLimitC;
    private double wb_airInRA_UpperLimitC;
    private double wb_airInHS_LowerLimitC;

    private double t_watInHS_UpperLimitH;
    private double t_watInHS_LowerLimitH;
    private double db_airInRA_UpperLimitH;
    private double db_airInRA_LowerLimitH;
```

```
    private RadPanelModule2014 radPanel = null;

    //グラフ表示など
    private GraphCommonJFrame2015 gGCJF = null;
    private GraphCommonData2015[] gData = null;

    private boolean isCPipe = false;//冷水配管有効の時true

    @Override
    public void setProfile(BestSpecs spec) {
//    外部定義項目取得
      if(spec == null) {
         return;
      }
      Map<String,String> map=spec.getSpec();
      if(map == null) {
         return;
      }

      // 機器名称
      this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

      this.grzName = spec.getSpecValue( this.SPEC_grzName, "" );
      if( !this.grzName.equals( "" )){
         this.isUseZoneAir = true;
      }

      if( this.isUseZoneAir ){
         this.zoneAirforSSinside = new ZoneAirforSSinsideModule201502();
         this.zoneAirforSSinside._setProfile( this.name, this.grzName );
      }

      // 機器番号
    // this.m_no = spec.getSpecValue( this.SPEC_m_no, "" );
      if(null!=map.get(this.SPEC_m_no)){
         this.m_no = (String)map.get(this.SPEC_m_no);
         //******************090725-s
         try{
            if(m_no.substring(0 , 1).equals("+")==true){
               saijohatsu=Double.parseDouble(m_no.substring(1 , 3));
            }else{
               saijohatsu=50.0;
            }
         }
         catch(NumberFormatException e){
            saijohatsu=50.0;
         }
         //******************090725-s
      } else {
         ///System.out.println("(W)SPEC_機器番号がありません");
         saijohatsu=50.0;  //******************090725
      }

      //  機器種別  0_定速型、1_インバータ型
      this.m_ty = spec.getSpecValue( this.SPEC_m_ty, "FCU_EHP_RAD2015" );

      //kiki_file="EHP_WaterCooled20100625";
      this.kiki_file="FCU_EHP_RAD2015";
      //
      if(m_ty.equals("0_201303_定速型")){
         this.kiki_file="EHP_WaterCooled201303";
      }
      if(m_ty.equals("1_201303_インバータ型")){
         this.kiki_file="EHP_WaterCooledInv201303";
                        //EHP_WaterCooledInv201303
      }

      //
      if(m_ty.equals("0_定速型")){
         this.kiki_file="EHP_WaterCooled20100625";
      }
```

```
        if(m_ty.equals("1_インバータ型")){
           this.kiki_file="EHP_WaterCooledInv20100625";
        }


        // 機器型式
        this.m_kt = spec.getSpecValue( this.SPEC_m_kt, "" );

        // 台数
        this.in_dai = spec.getSpecValue( this.SPEC_de_dai, 1. );

        // 定格冷房能力
        this.de_Qc_FCU_RAD   = spec.getSpecValue( this.SPEC_de_Qc_FCU_RAD, 0. ) * in_dai;
        this.de_Qc_FCU_RAD_1 = spec.getSpecValue( this.SPEC_de_Qc_FCU_RAD, 0. );

        // 温水時冷房能力
        this.de_Qc_HP_RAD   = spec.getSpecValue( this.SPEC_de_Qc_HP_RAD, 0. ) * in_dai;
        this.de_Qc_HP_RAD_1 = spec.getSpecValue( this.SPEC_de_Qc_HP_RAD, 0. );

        // 放射冷房能力
        this.de_Qc_RAD   = spec.getSpecValue( this.SPEC_de_Qc_RAD, 0. ) * in_dai;
        this.de_Qc_RAD_1 = spec.getSpecValue( this.SPEC_de_Qc_RAD, 0. );

        // 定格冷房入力(電力)
        this.de_PPEc_FCU_RAD   = spec.getSpecValue( this.SPEC_de_PPEc_FCU_RAD, 0. ) * in_dai;
        this.de_PPEc_FCU_RAD_1 = spec.getSpecValue( this.SPEC_de_PPEc_FCU_RAD, 0. );

        // 温水時冷房入力(電力)
        this.de_PPEc_HP_RAD   = spec.getSpecValue( this.SPEC_de_PPEc_HP_RAD, 0. ) * in_dai;
        this.de_PPEc_HP_RAD_1 = spec.getSpecValue( this.SPEC_de_PPEc_HP_RAD, 0. );

        // 放射冷房入力(電力)
        this.de_PPEc_RAD   = spec.getSpecValue( this.SPEC_de_PPEc_RAD, 0. ) * in_dai;
        this.de_PPEc_RAD_1 = spec.getSpecValue( this.SPEC_de_PPEc_RAD, 0. );


        // 定格暖房能力
        this.de_Qh_FCU_RAD   = spec.getSpecValue( this.SPEC_de_Qh_FCU_RAD, 0. ) * in_dai;
        this.de_Qh_FCU_RAD_1 = spec.getSpecValue( this.SPEC_de_Qh_FCU_RAD, 0. );

        //  冷水時暖房能力
        this.de_Qh_HP_RAD   = spec.getSpecValue( this.SPEC_de_Qh_HP_RAD, 0. ) * in_dai;
        this.de_Qh_HP_RAD_1 = spec.getSpecValue( this.SPEC_de_Qh_HP_RAD, 0. );

        // 放射暖房能力
        this.de_Qh_RAD   = spec.getSpecValue( this.SPEC_de_Qh_RAD, 0. ) * in_dai;
        this.de_Qh_RAD_1 = spec.getSpecValue( this.SPEC_de_Qh_RAD, 0. );

/*      // 低温暖房能力
        this.out_Qh_L   = spec.getSpecValue( this.SPEC_out_Qh_L, 0. ) * in_dai;
        this.out_Qh_L_1 = spec.getSpecValue( this.SPEC_out_Qh_L, 0. ) * in_dai;
*/
        // 定格暖房入力(電力)
        this.de_PPEh_FCU_RAD   = spec.getSpecValue( this.SPEC_de_PPEh_FCU_RAD, 0. ) * in_dai;
        this.de_PPEh_FCU_RAD_1 = spec.getSpecValue( this.SPEC_de_PPEh_FCU_RAD, 0. );

        // 冷水時暖房入力(電力)
        this.de_PPEh_HP_RAD   = spec.getSpecValue( this.SPEC_de_PPEh_HP_RAD, 0. ) * in_dai;
        this.de_PPEh_HP_RAD_1 = spec.getSpecValue( this.SPEC_de_PPEh_HP_RAD, 0. );

        // 放射暖房入力(電力)
        this.de_PPEh_RAD   = spec.getSpecValue( this.SPEC_de_PPEh_RAD, 0. ) * in_dai;
        this.de_PPEh_RAD_1 = spec.getSpecValue( this.SPEC_de_PPEh_RAD, 0. );

/*      // 低温暖房入力(電力)
        this.out_PPEout_PPEh_Lh_C   = spec.getSpecValue( this.SPEC_out_PPEh_L, 0. ) * in_dai;
        this.out_PPEout_PPEh_Lh_C_1 = spec.getSpecValue( this.SPEC_out_PPEh_L, 0. );
*/
/*
        // クランクケースヒータ(運転時)
        this.out_CLEa_R   = spec.getSpecValue( this.SPEC_out_CLEa_R, 0. ) * in_dai;
        this.out_CLEa_R_1 = spec.getSpecValue( this.SPEC_out_CLEa_R, 0. );
```

```
        // クランクケースヒータ(停止時)
        this.out_CLEa_S   = spec.getSpecValue( this.SPEC_out_CLEa_S, 0. ) * in_dai;
        this.out_CLEa_S_1 = spec.getSpecValue( this.SPEC_out_CLEa_S, 0. );

        // 待機電力(運転時)
        this.out_TKEa_R   = spec.getSpecValue( this.SPEC_out_TKEa_R, 0. ) * in_dai;
        this.out_TKEa_R_1 = spec.getSpecValue( this.SPEC_out_TKEa_R, 0. );

        // 待機電力(待機時)
        this.out_TKEa_T   = spec.getSpecValue( this.SPEC_out_TKEa_T, 0. ) * in_dai;
        this.out_TKEa_T_1 = spec.getSpecValue( this.SPEC_out_TKEa_T, 0. );

        // 待機電力(停止時)
        this.out_TKEa_S   = spec.getSpecValue( this.SPEC_out_TKEa_S, 0. ) * in_dai;
        this.out_TKEa_S_1 = spec.getSpecValue( this.SPEC_out_TKEa_S, 0. );
*/

        //SPEC_de_fRate_watInHS   = "熱源水定格水量[g/s]";
        this.de_fRate_watInHS = spec.getSpecValue( this.SPEC_de_fRate_watInHS, 0. );

        if( this.de_fRate_watInHS <= 0 ){
           BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                 SystemMessageConstants.CAL_ERROR2 ,
                 new String[]{ "( "+de_fRate_watInHS+"/"+moduleName+" )[ PAC EHP水熱源 ]", "熱源水定格水量を設定してくださ
い"} ));
           this.de_fRate_watInHS = this.de_Qc_FCU_RAD * 0.11;//20110505nino
        }

        //SPEC_de_fRate_watRAD   = "放射定格水量[g/s]";
        this.de_fRate_watRAD = spec.getSpecValue( this.SPEC_de_fRate_watRAD, 0. );

        if( this.de_fRate_watRAD <= 0 ){
           BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                 SystemMessageConstants.CAL_ERROR2 ,
                 new String[]{ "( "+de_fRate_watRAD+"/"+moduleName+" )[ PAC EHP水熱源 ]", "放射定格水量を設定してください
"} ));
           this.de_fRate_watRAD = this.de_Qc_FCU_RAD * 0.11 * 0.72;
        }

/*      // 機器起動停止負荷率
        if(null!=map.get(this.SPEC_in_run_stop)){
           this.in_run_stop = Double.parseDouble((String)map.get(this.SPEC_in_run_stop))*100.0;
        } else {
           System.out.println(this.moduleName +"(E)SPEC_機器起動停止負荷率がありません");
           in_run_stop=30.0;
        }
*/
        // 相数
        this.phase = spec.getSpecValue( this.SPEC_Phase, 3 );

        // 電圧
        this.voltage = spec.getSpecValue( this.SPEC_Voltage, 200. );

        // 周波数
        this.frequency = spec.getSpecValue( this.SPEC_Frequency, 50. );

        // 力率
        this.powerFactor = spec.getSpecValue( this.SPEC_PowerFactor, 0.8 );



        // 定格風量
        this.de_Va_S   = spec.getSpecValue( this.SPEC_de_Va_S, 0. ) * in_dai;
        this.de_Va_S_1 = spec.getSpecValue( this.SPEC_de_Va_S, 0. );

        if( this.de_Va_S <= 0 ){
           BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                 SystemMessageConstants.CAL_ERROR2 ,
                 new String[]{ "( "+de_Va_S+"/"+moduleName+" )[ PAC EHP水熱源 ]", "風量を設定してください"} ));

        }
```

```
        // 定格ファン消費電力
        this.de_PPEa_S   = spec.getSpecValue( this.SPEC_de_PPEa_S, 0. ) * in_dai;
        this.de_PPEa_S_1 = spec.getSpecValue( this.SPEC_de_PPEa_S, 0. );

/*
        // 待機電力(運転時)
        this.in_TKEa_R   = spec.getSpecValue( this.SPEC_in_TKEa_R, 0. ) * in_dai;
        this.in_TKEa_R_1 = spec.getSpecValue( this.SPEC_in_TKEa_R, 0. );

        // 待機電力(停止時)
        this.in_TKEa_S   = spec.getSpecValue( this.SPEC_in_THEa_S, 0. ) * in_dai;
        this.in_TKEa_S_1 = spec.getSpecValue( this.SPEC_in_THEa_S, 0. );
*/

        // 機器起動停止負荷率
        this.in_run_stop = spec.getSpecValue( this.SPEC_in_run_stop, 0.3 );

        this.in_run_stop_Num = this.in_run_stop;

        // 定格加湿能力
        this.de_HUM_mx   = spec.getSpecValue( this.SPEC_de_HUM_mx, 0. ) * in_dai;
        this.de_HUM_mx_1 = spec.getSpecValue( this.SPEC_de_HUM_mx, 0. );

        // 加湿飽和効率
        this.de_HUM_rt   = spec.getSpecValue( this.SPEC_de_HUM_rt, 0.7 ) * 100.;

        // 加湿器ONOFF湿度
        this.de_HUM_on   = spec.getSpecValue( this.SPEC_de_HUM_on, 0.4 ) * 100.;

        // 取入外気量
        this.de_OA_S   = spec.getSpecValue( this.SPEC_de_OA_S, 0. ) * in_dai;
        //this.in_OA_S_1 = spec.getSpecValue( this.SPEC_in_OA_S, 0. );

        // 全熱交換器効率
        this.de_EX_S = spec.getSpecValue( this.SPEC_de_EX_S, 0.6 );

        //SPEC_isHexbypass = "熱交バイパスあり";
        this.isHexbypass = spec.getSpecValue( this.SPEC_isHexbypass, false );

        // 全熱交換器消費電力[W]
        this.de_EX_PE   = spec.getSpecValue( this.SPEC_de_EX_PE, 0. ) * in_dai;
        //this.in_EX_PE_1 = spec.getSpecValue( this.SPEC_in_EX_PE, 0. );


        //isGVisibleを取得
        this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

        // 最大同時表示ステップ数
        this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );


        //低負荷領域の計算方法
        this.calcTypeLowerRangeLoad = spec.getSpecValue( this.SPEC_CalcTypeLowerRangeLoad, "0_発停運転" );
        //calcTypeLowerRangeLoad
        //  0_発停運転
        //  1_下限入力値固定
        //  2_下限COP値固定
        //  3_下限入力値と中間切片
        if( this.calcTypeLowerRangeLoad.equals( "0_発停運転" )){
           this.num_calcTypeLowerRangeLoad = 0;
        }else if(this.calcTypeLowerRangeLoad.equals( "1_下限入力値固定" )){
           this.num_calcTypeLowerRangeLoad = 1;
        }else if(this.calcTypeLowerRangeLoad.equals( "2_下限COP値固定" )){
           this.num_calcTypeLowerRangeLoad = 2;
        }else if(this.calcTypeLowerRangeLoad.equals( "3_下限入力値と中間切片" )){
           this.num_calcTypeLowerRangeLoad = 3;
        }else{
           this.num_calcTypeLowerRangeLoad = 0;
```

```
    }

    //isAdjust2012 = "台数を調整する";//
    this.isAdjust2012 = spec.getSpecValue( this.SPEC_isAdjust2012, false );

    // 調整の計算ステップ数
    this.numAdjustSteps = spec.getSpecValue( this.SPEC_NumAdjustSteps, 12 );


    this.daiListc = new LinkedList<Double>();
    this.daiListh = new LinkedList<Double>();
    for( int i=0; i<this.numAdjustSteps; i++ ){
       this.daiListc.add( 0. );
       this.daiListh.add( 0. );
    }
    this.rcList = new LinkedList<Double>();
    this.rhList = new LinkedList<Double>();
    for( int i=0; i<this.numAdjustSteps; i++ ){
       this.rcList.add( 0. );
       this.rhList.add( 0. );
    }

    this.fRate_watInHSc = this.de_fRate_watInHS;
    this.fRate_watInHSh = this.de_fRate_watInHS;


    //
    this.radPanel = new RadPanelModule2014();
    this.radPanel.setProfile(spec);
}



@Override
public void initialize(IBestStateMessage stateNodes,
      IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
      IBestDomainSchedule schedule) {
    //System.out.println(this.moduleName + " BuillMultiOutイニシャライズ");
    //状態ノードを受け取る
    super.sm = stateNodes;
    //制御ノードを受け取る
    super.cm = commandNodes;
    //記録ノードを受け取る
    super.rm = recordNodes;

    //eleIn
    this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

    //airInOA
    this.airInOA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInOA );
    this.airInOA.setMaxFlowRate( this.de_OA_S );//20150423

    //airOutEA
    this.airOutEA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutEA );
    this.airOutEA.setMaxFlowRate( this.de_OA_S );//20150423

    //watInC H
    this.watInC = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInC );
    this.watInH = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInH );

    //watOutC H
    this.watOutC = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutC );
    this.watOutH = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutH );

    this.watInHS  = new  BestWater();
    this.watOutHS = new  BestWater();

    //watInHS
    //this.watInHS = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watIn );

    //watOutHS
    //this.watOutHS = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOut );
```

```
//    rmlinemap= (Map<String,BM_EHPdata>)super.sm.getState(super.getConnectionNode(this.S_NODE_valInLine));
//    if(null == rmlinemap) {
//       rmlinemap = new HashMap<String,BM_EHPdata>();
//       System.out.println("out=NULL ");
//    } else {
//       num = 0;
//       Set<String> LineEntry = this.rmlinemap.keySet();
//       for(Iterator i = LineEntry.iterator();i.hasNext();){
//          String key = (String) i.next();
////           System.out.println(" name="+name+" key="+key);
//          RMdata = rmlinemap.get(key);
//          rmnm[num]=key;
////           System.out.println("key="+key+" 0="+rmdata[0]+" 1="+rmdata[1]+" 2="+rmdata[2]+" 3="+rmdata[3]+"
4="+rmdata[4]+" 5="+rmdata[5]+" 6="+rmdata[6]);
//          num++;
//       }
//       //出力設定
//       for(int i=0;i<num;i++){
//          RMdata = rmlinemap.get(rmnm[i]);
//          RMdata.set_kiki(kiki_file);
//          rmlinemap.put(rmnm[i],RMdata);
//       }
//       super.sm.setState( super.getConnectionNode( this.S_NODE_valInLine), rmlinemap);
//    }


      filenames[0]=kiki_file;
      equipmentName=kiki_file;
      pacDB=new PACDBManager(path,filenames);
      pacDB.setEquipmentName(equipmentName);

      System.out.println( "filenames =" +filenames[0] + "  equipmentName="+equipmentName);
      //201509
      this.t_watInHS_UpperLimitC = this.pacDB.getformulaRangeMaxRaw( this.pacDB.getFACKcQtwi() );
      this.t_watInHS_LowerLimitC = this.pacDB.getformulaRangeMinRaw( this.pacDB.getFACKcQtwi() );
      this.wb_airInRA_UpperLimitC = this.pacDB.getformulaRangeMaxCol( this.pacDB.getFACKcQtwi() );
      this.wb_airInHS_LowerLimitC = this.pacDB.getformulaRangeMinCol( this.pacDB.getFACKcQtwi() );

      this.t_watInHS_UpperLimitH = this.pacDB.getformulaRangeMaxRaw( this.pacDB.getFACKhQtwi() );
      this.t_watInHS_LowerLimitH = this.pacDB.getformulaRangeMinRaw( this.pacDB.getFACKhQtwi() );
      this.db_airInRA_UpperLimitH = this.pacDB.getformulaRangeMaxCol( this.pacDB.getFACKhQtwi() );
      this.db_airInRA_LowerLimitH = this.pacDB.getformulaRangeMinCol( this.pacDB.getFACKhQtwi() );


      //airOutRM
      this.airOutSA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutSA );
      this.airOutSA.setMaxFlowRate( this.de_Va_S );//20150423
      if( this.isUseZoneAir ){
         this.zoneAirforSSinside.checkNumberAirChange( this.de_Va_S, "in_Va_S", this.moduleName, "initialize()",
this.name, this.isRecord, this.message);
         this.zoneAirforSSinside._initialize( );
      }

      //**************090407追加**************
      //watInCW
      this.watInCW = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInCW );

      //***************************************


      //watOutD
      this.watOutD = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutD );

      //出口


      //接続ノード　入口
      //S_NODE_airInRA
      this.airInRA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInRA );
      this.airInRA.setTempDB(24.0);
```

```
    this.airInRA.setHumi(0.01);
    this.airInRA.setMaxFlowRate( this.de_Va_S );//20150423

    //S_NODE_valInPID
    this.valInPID = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInPID );

    //
    this.radPanel.initialize( );

    //グラフ表示の準備
    if( this.isGVisible ){
       this.gData = new GraphCommonData2015[13];

       int i = 0;

       this.gData[i++] = new GraphCommonData2015( "M_watIn[g/s]", this.maxItemCount, 0, 50, 0 );
       this.gData[i++] = new GraphCommonData2015( "M_watInCW[g/s]", this.maxItemCount, 0, 50, 0 );
       this.gData[i++] = new GraphCommonData2015( "M_watOutD[g/s]", this.maxItemCount, 0, 50, 0 );

       this.gData[i++] = new GraphCommonData2015( "T_watIn[℃]", this.maxItemCount, 0, 50, 1 );
       this.gData[i++] = new GraphCommonData2015( "T_watOut[℃]", this.maxItemCount, 0, 50, 1 );
       this.gData[i++] = new GraphCommonData2015( "T_watInCW[℃]", this.maxItemCount, 0, 50, 1 );
       this.gData[i++] = new GraphCommonData2015( "T_airIn[℃]", this.maxItemCount, 0, 50, 1 );
       this.gData[i++] = new GraphCommonData2015( "T_airOut[℃]", this.maxItemCount, 0, 50, 1 );

       this.gData[i++] = new GraphCommonData2015( "Q_opeLoadT[W]", this.maxItemCount, 0, 1000, 2 );
       this.gData[i++] = new GraphCommonData2015( "Q_opeCapaT[W]", this.maxItemCount, 0, 1000, 2 );
       this.gData[i++] = new GraphCommonData2015( "P_eleIn[W]", this.maxItemCount, 0, 1000, 2 );

       this.gData[i++] = new GraphCommonData2015( "COP[-]", this.maxItemCount, 0, 10, 3 );
       this.gData[i++] = new GraphCommonData2015( "PID_valIn[-]", this.maxItemCount, 0, 10, 3 );

       String[] yName = { "質量流量[g/s]", "温度[℃]", "熱量・電力[W]","COP・PID[-]" };

       gGCJF = new GraphCommonJFrame2015( this.name+"_FCUHP", this.gData, yName );
    }

}


/**
 * 水冷EHP計算及び結果出力/各室内機よりも先に動きます。
 */
@Override
public void outputs() {

    if (null == super.cm || null == super.sm ){
       return;
    }

    /**
     * 入力
     */
    //---20060211-----------------------------------------------
    this.PPE_out = 0;
    this.opeLoadS = 0;
    this.opeLoadT = 0;
    this.coolingLoadT = 0.0;//要求処理熱量（冷却全熱）[W]
    this.heatingLoadT = 0.0;//要求処理熱量（加熱全熱）[W]
    this.opeHeatingLoadS = 0.0;//室内機処理熱量（加熱顕熱）[W]
    this.opeCoolingLoadS = 0.0;//室内機処理熱量（冷却顕熱）[W]
    this.opeHeatingLoadT = 0.0;//室内機処理熱量（加熱全熱）[W]
    this.opeCoolingLoadT = 0.0;//室内機処理熱量（冷却全熱）[W]


    if(dbflg==0){
       //filenames[0]=RMdata.get_kiki();
       //equipmentName=RMdata.get_kiki();
       //pacDB=new PACDBManager(path,filenames);
       //pacDB.setEquipmentName(equipmentName);
    }
    dbflg = 1;
```

```
//
this.airInRA = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airInRA));
if( this.isUseZoneAir ){
   this.zoneAirforSSinside._outputSetRA( this.airInRA );
}

this.airInOA = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airInOA));

this.watInC = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watInC));
if( Double.isNaN( watInC.getTemp() ) ){//20150305
   BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
         SystemMessageConstants.WARNING ,
         new String[]{ "(E)入口水温が異常値", this.name }));
   BestLogger.info( "(E)入口水温が異常値"
         + "__"+this.name+"__("+moduleName+"_outputs_watInC)" + AirSystemControl.getTimeStr() );
}

this.watInH = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watInH));
if( Double.isNaN( watInH.getTemp() ) ){//20150305
   BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
         SystemMessageConstants.WARNING ,
         new String[]{ "(E)入口水温が異常値", this.name }));
   BestLogger.info( "(E)入口水温が異常値"
         + "__"+this.name+"__("+moduleName+"_outputs_watInH)" + AirSystemControl.getTimeStr() );
}

//this.watInHS = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watIn ));
//if( Double.isNaN( watInHS.getTemp() ) ){//20150305
// BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
//       SystemMessageConstants.WARNING ,
//       new String[]{ "(E)入口水温が異常値", this.name }));
// BestLogger.info( "(E)入口水温が異常値"
//       + "__"+this.name+"__("+moduleName+"_outputs_watInHS)" + AirSystemControl.getTimeStr() );
//}


//停止中も熱源水は入口の状態で出ていくものとする
//this.watOutHS.copyAllState( this.watInHS );//20130110

this.swcIn   = super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcIn));
this.swcInOA = super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcInOA));
this.modIn   = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modIn));
this.modInPID= super.cm.getCommand(super.getConnectionNode(this.C_NODE_modInPID));
this.modInPIPE = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modInPIPE));

//PID操作量
this.valInPID = (BestValue) super.sm.getState( super.getConnectionNode( this.S_NODE_valInPID));


//
if( Airswc.isOFF( this.swcIn )){
   //停止
   //swcInで停止のときのみ入口流量を０とする
   this.fRate_watInHS = 0;//定格流量で定流量
   this.mState = 0;

}else if( Airswc.isON( this.swcIn )){
   //運転
   if( Airmod.isCOOL( this.modIn ) && Airmod.isHEAT( this.modIn )){
      //冷暖房同時の時はmodInPIPEで冷水と温水の切り替えを行う。
      //冷水・温水配管切替 入口
      if( Airmod.isCOOL( this.modInPIPE ) ){
         if( this.isRecord ){
            this.message.append( "(C)modIn冷暖modPIPE冷→冷水" );
         }
         //冷暖房、冷水使用
         this.fRate_watInHS = this.fRate_watInHSc;//定格流量で定流量
         this.watInC.setFlowRate( this.fRate_watInHS );
         this.watInH.setFlowRate( 0 );
         this.watInHS.copyAllState( this.watInC );
         this.isCPipe = true;
```

```
    }else if( Airmod.isHEAT( this.modInPIPE )){
       if( this.isRecord ){
          this.message.append( "(C)modIn冷暖modPIPE暖→温水" );
       }
       //冷暖房、温水使用
       this.fRate_watInHS = this.fRate_watInHSh;//定格流量で定流量
       this.watInC.setFlowRate( 0 );
       this.watInH.setFlowRate( this.fRate_watInHS );
       this.watInHS.copyAllState( this.watInH );
       this.isCPipe = false;

    }else{
       //modInPIPEが冷房でも暖房でもないときは　冷房として扱う
       if( this.isRecord ){
          this.message.append( "(C)modIn冷暖modPIPEなし→冷水" );
       }
       //冷暖房、冷水使用
       this.fRate_watInHS = this.fRate_watInHSc;//定格流量で定流量
       this.watInC.setFlowRate( this.fRate_watInHS );
       this.watInH.setFlowRate( 0 );
       this.watInHS.copyAllState( this.watInC );
       this.isCPipe = true;

    }

    //計算が冷房か暖房かは　modInPID　で判断する
    if( Airmod.isCOOL( this.modInPID )){
       if( this.isRecord ){
          this.message.append( "(C)modInPID冷" );
       }
       //冷房運転
       this.mState = 1;
    }else{
       if( this.isRecord ){
          this.message.append( "(C)modInPID暖" );
       }
       //暖房運転
       this.mState = 2;
    }

 }else if( Airmod.isCOOL( this.modIn )){
    //冷房専用の時
    if( this.isRecord ){
       this.message.append( "(C)modIn冷→冷水" );
    }
    //冷房運転、冷水使用
    this.fRate_watInHS = this.fRate_watInHSc;//定格流量で定流量
    this.watInC.setFlowRate( this.fRate_watInHS );
    this.watInH.setFlowRate( 0 );
    this.watInHS.copyAllState( this.watInC );
    this.isCPipe = true;
    this.mState = 1;

 }else if( Airmod.isHEAT( this.modIn )){
    //暖房専用の時
    if( this.isRecord ){
       this.message.append( "(C)modIn暖→温水" );
    }
    //暖房運転、温水使用
    this.fRate_watInHS = this.fRate_watInHSh;//定格流量で定流量
    this.watInC.setFlowRate( 0 );
    this.watInH.setFlowRate( this.fRate_watInHS );
    this.watInHS.copyAllState( this.watInH );
    this.isCPipe = false;
    this.mState = 2;

 }else if( Airmod.isVENTILATE( this.modIn )){
    if( this.isRecord ){
       this.message.append( "(C)modIn換" );
    }
    //換気モード
```

```
            this.mState = 3;
            this.fRate_watInHS = 0;
            this.watInC.setFlowRate( 0 );
            this.watInH.setFlowRate( 0 );
            this.watInHS.copyAllState( this.watInC );

         }else{
            //停止
            this.mState = 0;
         }
         BestAir.checkOpeData( this.airInRA, "airInRA", this.moduleName, "outputs()", this.name, this.isRecord,
this.message );
      }


      /**
       * 入力
       */

      //状態ノードから室内空気を設定
      //室内乾球温度を設定
      this.DBra = airInRA.getTempDB();
      this.WBra = airInRA.getTempWB();

      //室内絶対湿度を設定
      this.XGra = airInRA.getHumi();

      if( Double.isInfinite( this.DBra ) || Double.isNaN( this.DBra )){
         if( this.isRecord ){
            this.message.append( "(E)DBra異常値→0℃に設定");
         }
         this.mState= -1;
         this.DBra = 0;
      }
      if( Double.isInfinite( this.XGra ) || Double.isNaN( this.XGra ) || this.XGra < 0 ){
         if( this.isRecord ){
            this.message.append( "(E)XGra異常値→0.004g/gに設定");
         }
         this.mState= -1;
         this.XGra = 0.004;
      }

      //室内湿球温度を設定
      this.WBra = Psychrometrics.FNWbtx(this.DBra, this.XGra);

      //    System.out.println("DBin="+DBin+" "+XGin+"  "+WBin+"  ");

      //外気乾球温度設定
      this.DBoa = airInOA.getTempDB();

      //外気絶対湿度を設定
      this.XGoa = airInOA.getHumi();

      //外気湿球温度を設定
      this.WBoa = Psychrometrics.FNWbtx(this.DBoa, this.XGoa);
      if( WBoa < -9000.0 ) WBoa =WB_cal(this.DBoa, this.XGoa);


      //熱源水入口温度　流量
      this.t_watInHS = this.watInHS.getTemp();
      //this.fRate_watInHS = this.watInHS.getFlowRate();



      //    System.out.println("mState="+mState+" "+onOff+"  "+mode);
      //    System.out.println("DBoa="+DBoa+" "+XGoa+"  "+WBoa);

      //計算の切替

      //室内機側の計算
      switch(this.mState){
      case -1:
```

```
    //異常停止
    if( this.isRecord ){
       this.message.append("(C)異常停止");
    }
    this.fRate_watInHS = 0;
    this.calc_Stop_In0();
    this.watInC.setFlowRate( 0 );
    this.watInH.setFlowRate( 0 );
    this.watInHS.setFlowRate( 0 );
    this.watOutC.copyAllState( this.watInC );
    this.watOutH.copyAllState( this.watInH );
    break;
case 0:
    //停止
    if( this.isRecord ){
       this.message.append("(C)停止処理");
    }
    this.calc_Stop_In0();
    this.watInC.setFlowRate( 0 );
    this.watInH.setFlowRate( 0 );
    this.watInHS.setFlowRate( 0 );
    this.watOutC.copyAllState( this.watInC );
    this.watOutH.copyAllState( this.watInH );
    break;
case 1:
    //冷房
    if( this.de_Qc_FCU_RAD_1 > 0 ){
       if( this.isRecord ){
          this.message.append("(C)冷房運転");
       }
       this.GW = this.de_Va_S;
       this.cooling_cal();
       this.PPE_in = this.de_PPEa_S + this.de_TKEa_R + this.de_EX_PE;//20130717
    }else{
       if( this.isRecord ){
          this.message.append("(C)冷能力=0停止");//20130625
       }
       this.calc_Stop_In0();
       this.mState = 0;
    }
    //冷水・温水配管切替 出口
    if( this.isCPipe ){
       this.watOutC.copyAllState( this.watOutHS );
       this.watOutH.copyAllState( this.watInH );

    }else if( !this.isCPipe ){
       this.watOutC.copyAllState( this.watInC );
       this.watOutH.copyAllState( this.watOutHS );

    }

    break;
case 2:
    //暖房
    if( this.de_Qh_FCU_RAD_1 > 0 ){
       if( this.isRecord ){
          this.message.append("(C)暖房運転");
       }
       this.GW = this.de_Va_S;
       this.heating_cal();
       this.PPE_in = this.de_PPEa_S + this.de_TKEa_R + this.de_EX_PE;//20130717
    }else{
       if( this.isRecord ){
          this.message.append("(C)暖能力=0停止");//20130625
       }
       this.calc_Stop_In0();
       this.mState = 0;
    }
    //冷水・温水配管切替 出口
    if( this.isCPipe ){
       this.watOutC.copyAllState( this.watOutHS );
       this.watOutH.copyAllState( this.watInH );
```

```
            }else if( !this.isCPipe ){
               this.watOutC.copyAllState( this.watInC );
               this.watOutH.copyAllState( this.watOutHS );

            }

            break;
         case 3:
            //換気
            if( this.isRecord ){
               this.message.append("(C)換気運転");
            }
            this.GW = this.de_Va_S;
            this.ventilation_cal();
            this.PPE_in = this.de_PPEa_S + this.de_TKEa_R + this.de_EX_PE;//20130717
            //
            this.valInPID.setValue( 0. );
            this.watInC.setFlowRate( 0 );
            this.watInH.setFlowRate( 0 );
            this.watInHS.setFlowRate( 0 );
            this.watOutC.copyAllState( this.watInC );
            this.watOutH.copyAllState( this.watInH );
            break;
         default:
            if( this.isRecord ){
               this.message.append("(C)運転モード？停止");//20130625
            }
            this.fRate_watInHS = 0;
            this.calc_Stop_In0();
            this.watInC.setFlowRate( 0 );
            this.watInH.setFlowRate( 0 );
            this.watInHS.setFlowRate( 0 );
            this.watOutC.copyAllState( this.watInC );
            this.watOutH.copyAllState( this.watInH );
            this.mState = 0;
            System.out.println(this.moduleName + ">>Error<< onOff*modeが範囲外");
         }

//       this.allSc_heat = this.coolingLoadS;//要求冷房顕熱量
//       this.allSh_heat = this.Sh_Load;//要求暖房顕熱量
//       this.coolingLoadT = this.Tc_Load;//要求冷房全熱量
//       this.heatingLoadT = this.Th_Load;//要求暖房全熱量

         if( this.PPE_in + this.PPE_out <= 0 ){
            this.COP = 0;
         }else{
            this.COP = this.opeLoadT / ( this.PPE_in + this.PPE_out );
         }

         if( this.isAdjust2012 ){
            if( mState == 1 ){
               this.fRate_watInHS = this.fRate_watInHSc;
            }else if( mState == 2 ){
               this.fRate_watInHS = this.fRate_watInHSh;
            }
         }

         //*************************----------------
         //super.sm.setState( super.getConnectionNode( this.S_NODE_watOut ), this.watOutHS );
         super.sm.setState( super.getConnectionNode( this.S_NODE_watOutC ), this.watOutC );
         super.sm.setState( super.getConnectionNode( this.S_NODE_watOutH ), this.watOutH );

         this.E_PPE_in = this.PPE_in * Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 );
         this.E_PPE_out= this.PPE_out* Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 );
         this.eleIn.setActivePower( this.PPE_out + this.PPE_in );
         this.eleIn.setReactivePower(this.E_PPE_out + this.E_PPE_in );
         this.eleIn.setPhase( this.phase );
         this.eleIn.setVoltage( this.voltage );
         this.eleIn.setFrequency( this.frequency );
```

```
//    System.out.println("CAPrate="+CAPrate+" "+S_Load_RM+" "+S_Load+ " "+mState);
this.airOutSA.setTempDB( this.DBout );
this.airOutSA.setHumi( this.XGout );
this.airOutSA.setFlowRate( this.GW );

this.watOutD.setFlowRate( this.D_Wrate );
this.watOutD.setTemp( this.DBout );
this.watInCW.setFlowRate( this.CW_Wrate );
//    System.out.println("DBout="+DBout+" "+XGout+"  "+airOutRM.getFlowRate());

//出力側ノード
super.sm.setState( super.getConnectionNode( this.S_NODE_airOutSA ), this.airOutSA);
super.sm.setState( super.getConnectionNode( this.S_NODE_eleIn ), this.eleIn);
super.sm.setState( super.getConnectionNode( this.S_NODE_watOutD ), this.watOutD);
super.sm.setState( super.getConnectionNode( this.S_NODE_watInCW ), this.watInCW);//090407追加

//出力設定

//グラフデータ追加
if( this.isGVisible ){
   //とりあえずwatOutCW[0]のみグラフ化
   //gTank.addData( BestTimeManager.getDateWeatherTime(),this.watInCW_GT, this.watOutCW[0], this.m_watMy,
this.m_start, this.m_stop, this.m_capacity );

   double[] gData = new double[13];

/* this.gData[i++] = new GraphCommonData2015( "M_watIn[g/s]", this.maxItemCount, 0, 50, 0 );
   this.gData[i++] = new GraphCommonData2015( "M_watInCW[g/s]", this.maxItemCount, 0, 50, 0 );
   this.gData[i++] = new GraphCommonData2015( "M_watOutD[g/s]", this.maxItemCount, 0, 50, 0 );

   this.gData[i++] = new GraphCommonData2015( "T_watIn[g/s]", this.maxItemCount, 0, 50, 1 );
   this.gData[i++] = new GraphCommonData2015( "T_watOut[g/s]", this.maxItemCount, 0, 50, 1 );
   this.gData[i++] = new GraphCommonData2015( "T_watInCW[g/s]", this.maxItemCount, 0, 50, 1 );
   this.gData[i++] = new GraphCommonData2015( "T_airIn[g/s]", this.maxItemCount, 0, 50, 1 );
   this.gData[i++] = new GraphCommonData2015( "T_airOut[g/s]", this.maxItemCount, 0, 50, 1 );

   this.gData[i++] = new GraphCommonData2015( "Q_opeLoadT[W]", this.maxItemCount, 0, 1000, 2 );
   this.gData[i++] = new GraphCommonData2015( "Q_opeCapaT[W]", this.maxItemCount, 0, 1000, 2 );
   this.gData[i++] = new GraphCommonData2015( "P_eleIn[W]", this.maxItemCount, 0, 1000, 2 );

   this.gData[i++] = new GraphCommonData2015( "COP[-]", this.maxItemCount, 0, 10, 3 );
   this.gData[i++] = new GraphCommonData2015( "Ctrl_valIn[-]", this.maxItemCount, 0, 10, 3 );
*/

   int i=0;
   gData[i++] = this.watInHS.getFlowRate();
   gData[i++] = this.watInCW.getFlowRate();
   gData[i++] = this.watOutD.getFlowRate();

   gData[i++] = this.watInHS.getTemp();
   gData[i++] = this.watOutHS.getTemp();
   gData[i++] = this.watInCW.getTemp();
   gData[i++] = this.airInRA.getTempDB();
   gData[i++] = this.airOutSA.getTempDB();

   gData[i++] = this.opeLoadT;
   gData[i++] = this.capCooling + this.capHeating;
   gData[i++] = this.eleIn.getActivePower();

   gData[i++] = this.COP;
   gData[i++] = this.valInPID.getValue();

   this.gGCJF.addData( BestTimeManager.getDateWeatherTime(), gData );
}

//記録ノード
if( this.isRecord && super.rm != null ){
   this.record();

   this.radPanel.record( super.rm, this );
}
```

```
message.setLength(0);


//*************************----------------
if( this.isAdjust2012 ){
   //移動平均の最大値で調整していく
   if( true ){
      this.daiListc.removeLast();
      this.daiListc.addFirst( this.in_daicAdjust * this.valInPID.getValue() );
      //
      double sum_in_dai = 0;
      for( int i=0; i<this.numAdjustSteps; i++ ){
         sum_in_dai += this.daiListc.get( i );
      }
      this.avecdaiAdjust = sum_in_dai / this.numAdjustSteps;
      //
      if( sum_in_dai > this.maxcAdjustdai * this.numAdjustSteps ){
         this.maxcAdjustdai = this.avecdaiAdjust;

         this.in_daicAdjust = this.maxcAdjustdai;
         this.in_run_stop_Num = this.in_run_stop / this.in_daicAdjust;

         this.de_Qc_FCU_RAD   = this.de_Qc_FCU_RAD_1   * this.in_daicAdjust;//= "定格冷房能力";//kW
         this.de_Qc_RAD = this.de_Qc_RAD_1 * this.in_daicAdjust;//= "中間冷房能力";//kW
         this.de_PPEc_FCU_RAD = this.de_PPEc_FCU_RAD_1 * this.in_daicAdjust;//= "定格冷房入力(電力)";//kW
         this.de_PPEc_RAD = this.de_PPEc_RAD_1 * this.in_daicAdjust;//= "中間冷房入力(電力)";//kW

         this.fRate_watInHSc = this.de_fRate_watInHS * this.in_daicAdjust;//水量[g/s]

         this.de_Va_S   = this.de_Va_S_1   * this.in_daicAdjust;//= "定格風量";//m3/h
         this.de_PPEa_S = this.de_PPEa_S_1 * this.in_daicAdjust;//= "定格ファン消費電力";//W

         this.de_HUM_mx = this.de_HUM_mx_1 * this.in_daicAdjust;//= "定格加湿能力";//kg/h

         this.airInRA.setFlowRate(de_Va_S);
         this.airOutSA.setFlowRate(de_Va_S);
      }
   }


   if( true ){
      this.daiListh.removeLast();
      this.daiListh.addFirst( this.in_daihAdjust * this.valInPID.getValue() );
      //
      double sum_in_dai = 0;
      for( int i=0; i<this.numAdjustSteps; i++ ){
         sum_in_dai += this.daiListh.get( i );
      }
      this.avehdaiAdjust = sum_in_dai / this.numAdjustSteps;
      //
      if( sum_in_dai > this.maxhAdjustdai * this.numAdjustSteps ){
         this.maxhAdjustdai = this.avehdaiAdjust;

         this.in_daihAdjust = this.maxhAdjustdai;
         this.in_run_stop_Num = this.in_run_stop / this.in_daihAdjust;

         this.de_Qh_FCU_RAD   = this.de_Qh_FCU_RAD_1   * this.in_daihAdjust;//= "定格暖房能力";
         this.de_Qh_RAD   = this.de_Qh_RAD_1   * this.in_daihAdjust;//= "FCU暖房能力";
//          this.in_Qh_L   = this.in_Qh_L_1   * this.in_daihAdjust;//= "低温暖房能力";
         this.de_PPEh_FCU_RAD = this.de_PPEh_FCU_RAD_1 * this.in_daihAdjust;//= "定格暖房入力(電力)";
         this.de_PPEh_RAD = this.de_PPEh_RAD_1 * this.in_daihAdjust;//= "FCU暖房入力(電力)";
//          this.in_PPEh_L = this.in_PPEh_L_1 * this.in_daihAdjust;//= "低温暖房入力(電力)";

         this.fRate_watInHSh = this.de_fRate_watInHS * this.in_daihAdjust;//水量[g/s]

         this.de_Va_S   = this.de_Va_S_1   * this.in_daihAdjust;//= "定格風量";//m3/h
         this.de_PPEa_S = this.de_PPEa_S_1 * this.in_daihAdjust;//= "定格ファン消費電力";//W

         this.de_HUM_mx = this.de_HUM_mx_1 * this.in_daihAdjust;//= "定格加湿能力";//kg/h

         this.airInRA.setFlowRate(de_Va_S);
```

```
                this.airOutSA.setFlowRate(de_Va_S);
            }

        }
    }
    //*************************----------------

}

private void calc_Stop_In0(){
    //出口空気の状態は入口空気の状態とする
    this.airOutSA.copyAllvalState( this.airInRA );
    this.airOutSA.setFlowRate(0.0);

    this.PPE_in = this.de_TKEa_S;
    this.DBin = this.DBra;
    this.XGin = this.XGra;
    this.WBin = this.WBra;
    //
    this.DBout = this.DBin;//乾球温度
    this.XGout = this.XGin;//絶対湿度
    this.GW = 0.0;
    this.D_Wrate = 0.0;
    this.CW_Wrate = 0.0;
    this.COP = 0.0;
    //
    this.valInPID.setValue( 0. );
    //
    this.watOutHS.setFlowRate( this.fRate_watInHS );

    this.opeCoolingLoadS = 0.0;
    this.opeCoolingLoadT = 0.0;
    this.opeCoolingLoadAIR = 0.0;
    this.opeCoolingLoadRAD = 0.0;
    this.opeHeatingLoadS = 0.0;
    this.opeHeatingLoadT = 0.0;
    this.opeHeatingLoadAIR = 0.0;
    this.opeHeatingLoadRAD = 0.0;
    this.opeLoadS = 0.0;
    this.opeLoadT = 0.0;
    this.capCooling = 0.0;
    this.capHeating = 0.0;


}




/**
 * 記録
 */
private void record(){
    if( super.rm != null ){
        if( CheckPrintModule.isPrintMessage ){
            //message
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_message, this.name, message.toString() );
        }

        if( CheckPrintModule.isPrintEnergy ){
            //記録ノードに室外機消費電力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID4, this.name,
AirFormat.df_2(this.eleIn.getActivePower()));
        }

        if( CheckPrintModule.isPrintLoad ){
            //記録ノードに室内機処理熱量（全熱）を設定
            if( this.opeLoadT > 0 ){
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT,  this.name, AirFormat.df_2(-
this.opeLoadT ));//処理全熱量合計#W#熱量
            }else{
```

```
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT,  this.name,
AirFormat.df_2( this.opeLoadT ));//処理全熱量合計#W#熱量
            }
        }

        if( CheckPrintModule.isPrintStateOut ){
            //記録ノードに吹出口乾球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBout, this.name,
AirFormat.df_2( this.airOutSA.getTempDB() ));
            //記録ノードに吹出口絶対湿度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_XGout, this.name,
AirFormat.df_5( this.airOutSA.getHumi() ));
            //記録ノードに吹出口風量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_GW, this.name,
AirFormat.df_2( this.airOutSA.getFlowRate() ));
            //記録ノードにドレン量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_D_Wrate, this.name,
AirFormat.df_2(D_Wrate));
            //記録ノードに熱源水出口流量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_M_watOutC, this.name,
AirFormat.df_2(this.watOutC.getFlowRate()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_M_watOutH, this.name,
AirFormat.df_2(this.watOutH.getFlowRate()));
            //記録ノードに熱源水出口温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_T_watOutC, this.name,
AirFormat.df_2(this.watOutC.getTemp()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_T_watOutH, this.name,
AirFormat.df_2(this.watOutH.getTemp()));
        }

        if( CheckPrintModule.isPrintStateMy ){

            //記録ノードに要求熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_coolingLoadT, this.name,
AirFormat.df_2(this.coolingLoadT));//冷却要求全熱量合計#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_heatingLoadT, this.name,
AirFormat.df_2(this.heatingLoadT));//加熱要求全熱量合計#W#熱量

            //記録ノードに処理可能熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_capCooling, this.name,
AirFormat.df_2(this.capCooling));//冷却可能熱量/最大#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_capHeating, this.name,
AirFormat.df_2(this.capHeating));//加熱可能熱量/最大#W#熱量

            //記録ノードに室内機処理熱量（顕熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_opeLoadS,  this.name,
AirFormat.df_2(opeLoadS));//処理顕熱量合計#W#熱量
            //記録ノードに室内機処理熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_opeLoadT,  this.name,
AirFormat.df_2(opeLoadT));//処理全熱量合計#W#熱量

            //COP
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_COP, this.name, AirFormat.df_3(this.COP));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_pidVal, this.name,
AirFormat.df_4(this.valInPID.getValue()));
        }


        if( CheckPrintModule.isPrintStateIn ){
            //記録ノードに加湿給水量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID8i, this.name,
AirFormat.df_2(CW_Wrate));
            //記録ノードに入口乾球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBairIn, this.name,
AirFormat.df_2( DBin ));
            //記録ノードに入口絶対湿度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_XairIn, this.name,
AirFormat.df_5( XGin ));
            //記録ノードに入口乾球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBairInRM, this.name,
AirFormat.df_2( this.airInRA.getTempDB() ));
```

```
            //記録ノードに入口絶対湿度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_XairInRM, this.name,
AirFormat.df_5( this.airInRA.getHumi() ));

            //記録ノードに熱源水出口温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_T_watInC, this.name,
AirFormat.df_2(this.watInC.getTemp()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_T_watInH, this.name,
AirFormat.df_2(this.watInH.getTemp()));
         }

         if( CheckPrintModule.isPrintAdjust ){
            //
            if( this.isAdjust2012 ){
               //記録ノードに室外機調整能力を設定
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qc_Adjust, this.name,
this.de_Qc_FCU_RAD );//室外機調整能力[W]out_Qc_Adjust
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qh_Adjust, this.name,
this.de_Qh_FCU_RAD );//室外機調整能力[W]out_Qh_Adjust
               //記録ノードに室外機調整能力を設定
               //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rcAdjust", this.name,
this.out_rcAdjust );//室外機調整率[-]
               //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rhAdjust", this.name,
this.out_rhAdjust );//室外機調整率[-]
               //記録ノードに調整台数を設定
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_in_daicAdjust, this.name,
AirFormat.df_2(this.in_daicAdjust) );
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_in_daihAdjust, this.name,
AirFormat.df_2(this.in_daihAdjust) );
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_avecdaiAdjust, this.name,
AirFormat.df_2(this.avecdaiAdjust) );
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_avehdaiAdjust, this.name,
AirFormat.df_2(this.avehdaiAdjust) );
            }
         }
      }
   }

   /**
    * 水冷EHP計算/各室内機の集計後
    */

   @Override
   public void update() {
      if( this.isUseZoneAir ){
         this.zoneAirforSSinside._update( this.airOutSA );
      }
   }

   public Object viewInternal(TestCommand cmd) {

      ArrayList<Object> result = new ArrayList<Object>();
      result.add(super.sm);
      result.add(super.cm);
      result.add(super.rm);
      //外部定義
      result.add(this.name);
      result.add(this.m_kt);

      result.add(this.airOutEA.getTempDB());

      result.add(this.de_Qc_FCU_RAD);
      result.add(this.de_Qc_RAD);
      result.add(this.de_PPEc_FCU_RAD);
      result.add(this.de_PPEc_RAD);
      result.add(this.de_Qh_FCU_RAD);
      result.add(this.de_Qh_RAD);
//    result.add(this.in_Qh_L);
      result.add(this.de_PPEh_FCU_RAD);
      result.add(this.de_PPEh_RAD);
//    result.add(this.in_PPEh_L);
```

```
    result.add(this.DBoa);
    result.add(this.XGoa);
    result.add(this.DBra);
    result.add(this.XGra);

    return result;
}




private double WB_cal(double ddb,double xx){
    double wwb,x1,de=1;
    int kk=-1,j=0;
    wwb=ddb;

    do{
        j++;
        x1=Psychrometrics.FNXtw(ddb,wwb);
        if(Math.abs(x1-xx)<0.000003) {
            //          System.out.println(" j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
            return wwb;
        }
        if(x1<xx) {
            if(kk==-1) de=de/2.0;
            wwb=wwb+de;
            kk=1;
        }
            if(x1>=xx) {
            if(kk==1) de=de/2.0;
            wwb=wwb-de;
            kk=-1;
        }

    }while(j<1000);
    //    System.out.println("*j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
    return wwb;

}



private void cooling_cal(){
    //室側の冷房計算
    double heatCapacity = 0.0;
    double heatCapacityHPFCU=0.0;
    double heatCapacityRAD =0.0;
    double eleHPFCU = 0.0;
    double eleFCU = 0.0;
    double ele = 0.0;
    double rCode0 = 0.0,rCode1 = 0.0;
    double IAin,IAout1,DBout1,XGout1;
    double HEXrate;
    double loadRAD=0.0;//RAD要求負荷

    HEXrate = this.de_EX_S;

    //外気取り入れありの時の吸い込み状態の計算
    //バイパス制御 追加 エンタルピで判断
    if( this.isHexbypass == true ){  //20101212nino
        //全熱交換器バイパス有
        this.IAra = Psychrometrics.FNH( this.DBra, this.XGra );//20130109
        this.IAoa = Psychrometrics.FNH( this.DBoa, this.XGoa );

        if( this.IAra > this.IAoa && this.valInPID.getValue() > 0.001 && this.HEXselect==1 ){//20130109
            //エンタルピ基準
            HEXrate = 0.0;
        }
        if( this.DBra > this.DBoa && this.valInPID.getValue() > 0.001 && this.HEXselect==2 ){//20130109
            //温度/顕熱基準
            HEXrate = 0.0;
```

```
        }
      }
      if( Airswc.isON( this.swcInOA )){//20101212nino
        //吸込み状態を求める
        this.DBin = (( this.de_Va_S - this.de_OA_S ) * this.DBra + (( this.DBoa - this.DBra ) * ( 1.0 - HEXrate ) +
this.DBra ) * this.de_OA_S ) / this.de_Va_S;//20130109
        this.XGin = (( this.de_Va_S - this.de_OA_S ) * this.XGra + (( this.XGoa - this.XGra ) * ( 1.0 - HEXrate ) +
this.XGra ) * this.de_OA_S ) / this.de_Va_S;//20130109
      }else{//20101212nino
        DBin = DBra;//20130109
        XGin = XGra;//20130109
      }
      this.WBin = Psychrometrics.FNWbtx( this.DBin, this.XGin );

      // System.out.println("C/DBin="+DBin+" "+XGin+"  ");
      // System.out.println("C/DBoa="+DBoa+" "+XGoa+"  ");
      // System.out.println("C/DBra="+DBra+" "+XGra+"  ");
      // System.out.println("  ");
      // System.out.println("DBra="+DBra+" "+XGra+"  ");

      //特性の上下限のチェック 20120821nino
      double twatIn = this.t_watInHS;
      double wbIn = this.WBin;

      if( twatIn > this.t_watInHS_UpperLimitC ){
        //熱源水入口温度＞上限の時停止
        if( this.isRecord ){
          this.message.append( "(C)熱源水入口温度＞"+this.t_watInHS_UpperLimitC+"上限→停止");
        }

        this.mState = 3;//能力＝０→換気モードとする20130415
        this.calc_Stop_In0();
        return;
      }
      if( twatIn < this.t_watInHS_LowerLimitC ){
        //熱源水入口温度＜下限の時　下限値の特性で運転
        if( this.isRecord ){
          this.message.append( "(C)熱源水入口温度＜"+this.t_watInHS_LowerLimitC+"下限→下限値で特性計算");
        }
        twatIn = this.t_watInHS_LowerLimitC;
      }
      if( wbIn > this.wb_airInRA_UpperLimitC ){
        //室内湿球温度＞上限の時　上限値の特性で運転
        if( this.isRecord ){
          this.message.append( "(C)室WB＞"+this.wb_airInRA_UpperLimitC+"上限→上限値で特性計算");
        }
        wbIn = this.wb_airInRA_UpperLimitC;
      }
      if( wbIn < this.wb_airInHS_LowerLimitC ){
        //室内湿球温度＜下限の時　下限値の特性で運転
        if( this.isRecord ){
          this.message.append( "(C)室WB＜"+this.wb_airInHS_LowerLimitC+"下限→下限値で特性計算");
        }
        wbIn = this.wb_airInHS_LowerLimitC;
      }


      //能力補正
      rCode0 = this.pacDB.getKcQtwi( twatIn, wbIn );//水温WB補正
      rCode1 = this.pacDB.getKcf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
      //FCU+RAD可能能力
      heatCapacityHPFCU = rCode0 * rCode1 * this.de_Qc_FCU_RAD;

      rCode0 = this.pacDB.getKcQtwir( twatIn, wbIn );//水温WB補正
      //rCode1 = this.pacDB.getKcf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
      //RAD可能能力
      heatCapacityRAD =  rCode0 * rCode1 * this.de_Qc_RAD;

      //要求負荷　処理負荷
      this.coolingLoadT = heatCapacityHPFCU * this.valInPID.getValue();
      this.opeCoolingLoadT = this.coolingLoadT;
```

```
//RAD負荷
loadRAD = ( this.radPanel.watOut.getTemp() - 16. ) * this.radPanel.watOut.getFlowRate() * 4.18605;

//放射パネルへの送水温度、流量
if( this.coolingLoadT <= 0 ){
   this.opeCoolingLoadRAD = 0;
   this.radPanel.watIn.setFlowRate( 0. );

}else if( loadRAD > heatCapacityRAD ){
   this.opeCoolingLoadRAD = heatCapacityRAD;
   this.radPanel.watIn.setFlowRate( this.de_fRate_watRAD );

}else{
   this.opeCoolingLoadRAD = loadRAD;
   this.radPanel.watIn.setFlowRate( this.de_fRate_watRAD );

}
this.radPanel.watIn.setTemp( this.radPanel.watOut.getTemp() + this.opeCoolingLoadRAD / this.de_fRate_watRAD );


//ele
rCode0 = this.pacDB.getKcWtwi( twatIn, wbIn );//水温WB補正
rCode1 = this.pacDB.getKcwf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
eleHPFCU = rCode0 * rCode1 * this.de_PPEc_FCU_RAD;

rCode0 = this.pacDB.getKcWtwif( twatIn, wbIn );//水温WB補正
//rCode1 = this.pacDB.getKcwf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正
eleFCU = rCode0 * rCode1 * this.de_PPEc_RAD;

if( heatCapacityHPFCU < this.coolingLoadT ){
   //要求負荷が能力を超えているとき
   heatCapacity = heatCapacityHPFCU;
   this.opeCoolingLoadT = heatCapacityHPFCU;
   this.opeCoolingLoadAIR = this.opeCoolingLoadT - this.opeCoolingLoadRAD;
   ele = eleHPFCU;

}else if( heatCapacityRAD >= this.coolingLoadT ){
   //FCUだけで能力が足りている
   heatCapacity = heatCapacityRAD;
   this.opeCoolingLoadAIR = 0;
   //ele
   ele = eleFCU;

}else{
   //HPの運転が必要
   heatCapacity = heatCapacityHPFCU;
   this.opeCoolingLoadAIR = this.opeCoolingLoadT - this.opeCoolingLoadRAD;
   //ele
   ele = eleHPFCU;

}

this.capCooling = heatCapacity;


//吹き出し状態の計算
IAin   = Psychrometrics.FNH(this.DBin, this.XGin);
IAout1 = IAin - this.opeCoolingLoadAIR / this.de_Va_S * 1000.0;
DBout1 = Psychrometrics.FNDbrh( 90.0, IAout1 );//乾球温度
XGout1 = Psychrometrics.FNxtr( DBout1, 90.0);//絶対湿度
// System.out.println("IAra="+IAra+" "+IAout1+"  "+MHP_in_Vc_S);

if( this.XGin < XGout1 ){
   XGout1 = this.XGin;
   DBout1 = this.DBin - this.opeCoolingLoadT / this.de_Va_S;
   this.opeCoolingLoadS = this.opeCoolingLoadT;

}else{
   this.opeCoolingLoadS = ( this.DBin - DBout1 ) * this.de_Va_S;
}

this.opeCoolingLoadS += this.opeCoolingLoadRAD;
```

```
      this.DBout = DBout1;//乾球温度
      this.XGout = XGout1;//絶対湿度

      /*
         if(saijohatsu>0.0 && Psychrometrics.FNRhtx(DBin, XGin)<40.0){
            System.out.println("40以下"+(++u40)+"S_Load_RM="+S_Load_RM+" T_Load=_RM"+T_Load_RM+" sRc="+sRc+"
SS_rm="+SS_rm);
            System.out.println("   DBout0="+(DBra-S_Load_RM/in_Va_S)+"  DBout1="+(DBra-SS_rm/in_Va_S));
            System.out.println("   XXout0="+Psychrometrics.FNXth((DBra-S_Load_RM/in_Va_S), IAout)+"
XXout1="+Psychrometrics.FNXth((DBra-SS_rm/in_Va_S), IAout));
            System.out.println("   DBin  ="+DBin+"  XGin  ="+XGin+"  RHin  ="+Psychrometrics.FNRhtx(DBin, XGin));
            System.out.println("   DBoa  ="+DBoa+"  XGoa  ="+XGoa+"  RHoa  ="+Psychrometrics.FNRhtx(DBoa, XGoa));
            System.out.println("   DBra  ="+DBra+"  XGra  ="+XGra+"  RHra  ="+Psychrometrics.FNRhtx(DBra, XGra));

         }
       */

      // System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate);


      //////++++++++++++++++++++++++++++++++
      this.radPanel.watIn.setFlowRate( 12000. /  60. );
      this.radPanel.watIn.setTemp( 16. );

      this.radPanel.outputs_Object(this.swcIn, this.modIn, super.rm );


      //////++++++++++++++++++++++++++++++++


      double Rp;

      if( this.capCooling <= 0. ){
         this.Rc = 0;

      }else{
         this.Rc = this.coolingLoadT / this.capCooling;

      }

      if(this.Rc > 1.0 ) {
         this.Rc = 1.0;
      }

      //低負荷領域の計算仕分け:Rh
      if( this.Rc < this.in_run_stop ){
         switch( this.num_calcTypeLowerRangeLoad){
         case 0://  0_発停運転
            Rp = 0;
            break;

         case 1://  1_下限入力値固定
         case 2://  2_下限COP値固定
         case 3://  3_下限入力値と中間切片
            Rp = this.in_run_stop;
            break;

         default:
            Rp = 0;
         }
      }else{
         Rp = this.Rc;
      }


      //低負荷領域の計算仕分け:PPE
      if( this.Rc < this.in_run_stop && Rp > 0 ){
         switch( this.num_calcTypeLowerRangeLoad){
         case 0://  0_発停運転
            ele = 0.0;
            break;
```

```
            case 1://  1_下限入力値固定
               //何もしない
               break;

            case 2://  2_下限COP値固定
               ele *= ( this.Rc / Rp );
               break;

            case 3://  3_下限入力値と中間切片
               ele *= ( 0.5 + this.Rc / Rp / 2. );
               break;

            default:
            }
         }else{
            //何もしない
         }

         this.PPE_out = ele;

         //airOutEA
         //this.airOutEA.setFlowRate( this.fRate_watInHS );
         //this.airOutEA.setTempDB( ( this.T_Loadout + this.PPE_out ) / this.fRate_watInHS * 1000. +
this.airInOA.getTempDB() );

         //watOutHS
         this.watOutHS.setFlowRate( this.fRate_watInHS );
         this.watOutHS.setTemp( ( this.opeCoolingLoadT + this.PPE_out ) / this.fRate_watInHS / 4.18605 + this.t_watInHS );

         if( this.Rc < 0.0001 ){
            this.opeCoolingLoadS = 0.0;//実処理熱量（冷却顕熱）[W]
            this.opeCoolingLoadT = 0.0;//実処理熱量[W]
            this.capCooling      = 0.0;//処理可能熱量[W]
            this.opeHeatingLoadS = 0.0;//実処理熱量（加熱顕熱）[W]
            this.opeHeatingLoadT = 0.0;//実処理熱量[W]
            this.capHeating      = 0.0;//処理可能熱量[W]
            this.PPE_out = 0.0;
            this.watOutHS.copyAllState( this.watInHS );//20130110
            if( this.isRecord ){
               this.message.append( "(C)負荷率0で停止");
            }
            this.mState = 3;//能力＝０→換気モードとする20130415

         }else if( this.Rc < this.in_run_stop ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
               this.opeCoolingLoadS = 0.0;//実処理熱量（冷却顕熱）[W]
               this.opeCoolingLoadT = 0.0;//実処理熱量（冷却全熱）[W]
               this.capCooling      = 0.0;//処理可能熱量（冷却全熱）[W]
               this.opeHeatingLoadS = 0.0;//実処理熱量（加熱顕熱）[W]
               this.opeHeatingLoadT = 0.0;//実処理熱量（加熱全熱）[W]
               this.capHeating      = 0.0;//処理可能熱量（加熱全熱）[W]
               this.PPE_out = 0.0;
               this.watOutHS.copyAllState( this.watInHS );//20130110
               if( this.isRecord ){
                  this.message.append( "(C)負荷率＜停止負荷率で停止");
               }
               this.mState = 3;//能力＝０→換気モードとする20130415
               break;

            case 1://  1_下限入力値固定
               break;

            case 2://  2_下限COP値固定
               break;

            case 3://  3_下限入力値と中間切片
               break;

            default:
            }
```

```
    }

    this.opeLoadS = this.opeCoolingLoadS;
    this.opeLoadT = this.opeCoolingLoadT;

    //       System.out.println(" PPE="+PPE+" "+Rc+" "+rCode1+" "+rCode2+" "+rCode3);
  }



  private void heating_cal(){
    //室側の暖房計算
    double heatCapacity=0.0;
    double heatCapacityHPFCU=0.0;//全加熱能力
    double heatCapacityRAD=0.0;//放射の加熱能力
    double rCode0 = 0.0, rCode1 = 0.0;
    double HEXrate;
    double ele=0;
    double eleHPFCU=0;
    double eleFCU=0;
    double loadRAD =0.0;//RAD要求負荷

    HEXrate=de_EX_S;

    //バイパス制御 追加 20101212nino
    if( this.isHexbypass == true ){
      this.IAra = Psychrometrics.FNH( this.DBra, this.XGra );//20130109
      this.IAoa = Psychrometrics.FNH( this.DBoa, this.XGoa );
      if( this.IAra < this.IAoa && this.valInPID.getValue() > 0.001 && this.HEXselect == 1 ){//20130109
        HEXrate = 0.0;
      }
      if( this.DBra < this.DBoa && this.valInPID.getValue() > 0.001 && this.HEXselect == 2 ){//20130109
        HEXrate = 0.0;
      }
      if( this.IAra > this.IAoa && this.valInPID.getValue() < 0.001 && this.HEXselect == 1 ){//20130109
        HEXrate = 0.0;
      }
      if( this.DBra > this.DBoa && this.valInPID.getValue() < 0.001 && this.HEXselect == 2 ){//20130109
        HEXrate = 0.0;
      }
    }
    if( Airswc.isON( this.swcInOA )){//20101212nino
      this.DBin = (( this.de_Va_S - this.de_OA_S ) * this.DBra + ( this.DBra - ( this.DBra - this.DBoa ) * ( 1.0 -
HEXrate )) * this.de_OA_S ) / this.de_Va_S;//20130109
      this.XGin = (( this.de_Va_S - this.de_OA_S ) * this.XGra + ( this.XGra - ( this.XGra - this.XGoa ) * ( 1.0 -
HEXrate )) * this.de_OA_S ) / this.de_Va_S;//20130109
    }else{//20101212nino
      this.DBin = this.DBra;//20130109
      this.XGin = this.XGra;//20130109
    }
    this.WBin = Psychrometrics.FNWbtx(this.DBin,this.XGin);//20130109

    // System.out.println("H/DBin="+DBin+" "+XGin+"  ");
    // System.out.println("H/DBoa="+DBoa+" "+XGoa+"  ");
    // System.out.println("H/DBra="+DBra+" "+XGra+"  ");
    // System.out.println("  ");

    //上下限のチェック 20120821nino
    double twatIn = this.t_watInHS;
    double dbIn = this.DBin;

    if( twatIn > this.t_watInHS_UpperLimitH ){
      //熱源水入口温度＞上限の時　上限値の特性で運転
      if( this.isRecord ){
        this.message.append( "(C)熱源水入口温度＞"+this.t_watInHS_UpperLimitH+"上限→上限値で特性計算");
      }
      twatIn = this.t_watInHS_UpperLimitH;
    }
    if( twatIn < this.t_watInHS_LowerLimitH ){
      //熱源水入口温度＜下限の時　停止
      if( this.isRecord ){
```

```
        this.message.append( "(C)熱源水入口温度<"+this.t_watInHS_LowerLimitH+"下限→停止");
    }

    this.mState = 3;//能力＝０→換気モードとする20130415
    this.calc_Stop_In0();
    return;
}
if( dbIn > this.db_airInRA_UpperLimitH ){
    //室内乾球温度＞上限の時　上限値の特性で運転
    if( this.isRecord ){
        this.message.append( "(C)室DB>"+this.db_airInRA_UpperLimitH+"上限→上限値で特性計算");
    }
    dbIn = this.db_airInRA_UpperLimitH;
}
if( dbIn < this.db_airInRA_LowerLimitH ){
    //室内乾球温度＜下限の時　下限値の特性で運転
    if( this.isRecord ){
        this.message.append( "(C)室DB<"+this.db_airInRA_LowerLimitH+"下限→下限値で特性計算");
    }
    dbIn = this.db_airInRA_LowerLimitH;
}


//能力補正
rCode0 = this.pacDB.getKhQtwi( twatIn, dbIn );//水温DB補正
rCode1 = this.pacDB.getKhf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正

//FCU+RAD可能能力
heatCapacityHPFCU = rCode0 * rCode1 * this.de_Qh_FCU_RAD;

rCode0 = this.pacDB.getKhQtwir( twatIn, dbIn );//水温DB補正
//rCode1 = this.pacDB.getKhf( this.fRate_watInHS / this.fRate_watInHSc );//水量補正

//RAD可能能力
heatCapacityRAD = rCode0 * rCode1 * this.de_Qh_RAD;

//要求負荷　処理負荷
this.heatingLoadT = heatCapacityHPFCU * this.valInPID.getValue();
this.opeHeatingLoadT = this.heatingLoadT;

//RAD負荷
loadRAD = ( 32 - this.radPanel.watOut.getTemp() ) * this.radPanel.watOut.getFlowRate() * 4.18605;

//放射パネルへの送水温度、流量
if( this.heatingLoadT <= 0 ){
    this.opeHeatingLoadRAD = 0;
    this.radPanel.watIn.setFlowRate( 0. );

}else if( loadRAD > heatCapacityRAD ){
    this.opeHeatingLoadRAD = heatCapacityRAD;
    this.radPanel.watIn.setFlowRate( this.de_fRate_watRAD );

}else{
    this.opeHeatingLoadRAD = loadRAD;
    this.radPanel.watIn.setFlowRate( this.de_fRate_watRAD );

}
this.radPanel.watIn.setTemp( this.radPanel.watOut.getTemp() + this.opeHeatingLoadRAD / this.de_fRate_watRAD );

//ele
rCode0 = this.pacDB.getKhWtwi( twatIn, dbIn );//水温DB補正
rCode1 = this.pacDB.getKhwf( this.fRate_watInHS / this.fRate_watInHSh );//水量補正
eleHPFCU = rCode0 * rCode1 * this.de_PPEh_FCU_RAD;

rCode0 = this.pacDB.getKhWtwif( twatIn, dbIn );//水温DB補正
//rCode1 = this.pacDB.getKhwf( this.fRate_watInHS / this.fRate_watInHSh );//水量補正
eleFCU = rCode0 * rCode1 * this.de_PPEh_RAD;

if( heatCapacityHPFCU < this.heatingLoadT ){
    //要求負荷が能力を超えているとき
    heatCapacity = heatCapacityHPFCU;
    this.opeHeatingLoadT = heatCapacityHPFCU;
```

```
    this.opeHeatingLoadAIR = this.opeHeatingLoadT - this.opeHeatingLoadRAD;
    ele = eleHPFCU;


}else if( heatCapacityRAD >= this.heatingLoadT ){
    //放射だけで能力が足りている
    heatCapacity = heatCapacityRAD;
    this.opeHeatingLoadAIR = 0;
    //ele
    ele = eleFCU;


}else{
    //HPの運転が必要
    heatCapacity = heatCapacityHPFCU;
    this.opeHeatingLoadAIR = this.opeHeatingLoadT - this.opeHeatingLoadRAD;
    //ele
    ele = eleHPFCU;

}

this.capHeating = heatCapacity;
this.opeHeatingLoadS = this.opeHeatingLoadT;


//吹き出し状態の計算

// System.out.println("heatCapacity ="+rCode1+" "+rCode2+" "+heatCapacity+"  ");

//IAin   = Psychrometrics.FNH( this.DBin, this.XGin );
//IAout1 = IAin + heatCapacity / this.in_Va_S * 1000.0;
//XGout1 = this.XGin;
//DBout1 = Psychrometrics.FNDbxh( XGout1, IAout1 );
double IAout;

this.RHin = Psychrometrics.FNRhtx( this.DBin, this.XGin );

if( this.de_HUM_on <= this.RHin || this.de_HUM_mx < 0.00001 ){
    IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
    IAout = IAin +this.opeHeatingLoadAIR / this.de_Va_S * 1000.0;
    this.DBout = this.DBin + this.opeHeatingLoadAIR / this.de_Va_S;//乾球温度
    this.XGout = Psychrometrics.FNXth( this.DBout, IAout );//絶対湿度
    this.CW_Wrate = 0.0;
    //    System.out.println("1 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
    //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
} else {
    IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
    IAout = IAin + this.opeHeatingLoadAIR / this.de_Va_S*1000.0;
    this.DBout = Psychrometrics.FNDbrh( this.de_HUM_rt, IAout);
    this.XGout = Psychrometrics.FNxtr( this.DBout, this.de_HUM_rt);
    this.CW_Wrate = ( this.XGout - this.XGin ) * this.GW;

    //定格加湿量をオーバーするときの処理
    if( this.CW_Wrate > this.de_HUM_mx ){
        this.XGout = this.de_HUM_mx / this.GW+ this.XGin;
        this.DBout =Psychrometrics.FNDbxh( this.XGout, IAout);
        this.CW_Wrate = this.de_HUM_mx;
    }
    if( this.XGout < this.XGin){
        IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
        IAout = IAin + this.opeHeatingLoadAIR / this.de_Va_S * 1000.0;
        this.DBout = this.DBin + this.opeHeatingLoadAIR / this.de_Va_S;//乾球温度
        this.XGout = Psychrometrics.FNXth( this.DBout, IAout);//絶対湿度
        this.CW_Wrate = 0.0;
    }
    //    System.out.println("2 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
    //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
}

if( this.capHeating <= 0. ){
    this.Rh = 0.;
```

```
    }else{
       this.Rh = this.heatingLoadT / this.capHeating;

    }

    if( this.Rh > 1.0 ) {
       this.Rh = 1.0;
    }

    double Rp;

    //       System.out.println("Rh="+Rh);
    //低負荷領域の計算仕分け:Rh
    if( this.Rh < this.in_run_stop ){
       switch( this.num_calcTypeLowerRangeLoad){
       case 0://  0_発停運転
          Rp = 0;
          break;

       case 1://  1_下限入力値固定
       case 2://  2_下限COP値固定
       case 3://  3_下限入力値と中間切片
          Rp = this.in_run_stop;
          break;

       default:
          Rp = 0;
       }
    }else{
       Rp = this.Rh;
    }


    //入力
    this.PPE_out = ele;

    //低負荷領域の計算仕分け:PPE
    if( this.Rh < this.in_run_stop && Rp > 0 ){
       switch( this.num_calcTypeLowerRangeLoad){
       case 0://  0_発停運転
          this.PPE_out = 0.0;
          break;

       case 1://  1_下限入力値固定
          //何もしない
          break;

       case 2://  2_下限COP値固定
          this.PPE_out *= ( this.Rh / Rp );
          break;

       case 3://  3_下限入力値と中間切片
          this.PPE_out *= ( 0.5 + this.Rh / Rp / 2. );
          break;

       default:
       }
    }else{
       //何もしない
    }


    //       System.out.println("DBoa="+DBoa+" WBoa="+WBoa+" "+rCode0+" "+rCode1+" "+rCode2+" "+rCode3+" "+PPE);
    //System.out.println(" PAC 負荷率="+this.Rh+"    負荷律係数="+rCode1);


    //airOutEA
    //this.airOutEA.setFlowRate( this.fRate_watInHS );
    //this.airOutEA.setTempDB( ( -this.T_Loadout + this.PPE_out ) / this.fRate_watInHS * 1000. +
this.airInOA.getTempDB() );
    //watOutHS
    this.watOutHS.setFlowRate( this.fRate_watInHS );
```

```
      this.watOutHS.setTemp( ( -this.opeHeatingLoadT + this.PPE_out ) / this.fRate_watInHS / 4.18605 + this.t_watInHS );

      if( this.Rh < 0.0001 ){
         this.opeCoolingLoadS = 0.0;//実処理熱量（冷却顕熱）[W]
         this.opeCoolingLoadT = 0.0;//実処理熱量[W]
         this.capCooling      = 0.0;//処理可能熱量[W]
         this.opeHeatingLoadS = 0.0;//実処理熱量（加熱顕熱）[W]
         this.opeHeatingLoadT = 0.0;//実処理熱量[W]
         this.capHeating      = 0.0;//処理可能熱量[W]
         this.PPE_out = 0.0;
         this.watOutHS.copyAllState( this.watInHS );//20130110
         if( this.isRecord ){
            this.message.append( "(C)負荷率0で停止");
         }
         this.mState = 3;//能力＝０→換気モードとする20130415

      }else if( this.Rh < this.in_run_stop ){
         switch( this.num_calcTypeLowerRangeLoad){
         case 0://  0_発停運転
            this.opeCoolingLoadS = 0.0;//実処理熱量（冷却顕熱）[W]
            this.opeCoolingLoadT = 0.0;//実処理熱量（冷却全熱）[W]
            this.capCooling      = 0.0;//処理可能熱量（冷却全熱）[W]
            this.opeHeatingLoadS = 0.0;//実処理熱量（加熱顕熱）[W]
            this.opeHeatingLoadT = 0.0;//実処理熱量（加熱全熱）[W]
            this.capHeating      = 0.0;//処理可能熱量（加熱全熱）[W]
            this.PPE_out = 0.0;
            this.watOutHS.copyAllState( this.watInHS );//20130110
            if( this.isRecord ){
               this.message.append( "(C)負荷率＜停止負荷率で停止");
            }
            this.mState = 3;//能力＝０→換気モードとする20130415
            break;

         case 1://  1_下限入力値固定
            break;

         case 2://  2_下限COP値固定
            break;

         case 3://  3_下限入力値と中間切片
            break;

         default:
         }

      }

      //    this.watInCW.setFlowRate(CW_Wrate);
      this.opeLoadS = this.opeHeatingLoadS;
      this.opeLoadT = this.opeHeatingLoadT;
   }

   private void ventilation_cal(){
      //double heatCapacity=0.0,rCode1 = 0.0,rCode2 = 0.0,rCode3 = 0.0,rCode4 = 0.0,rCode5 = 0.0,rCode6 = 0.0,rCode7 =
0.0,rCode8 = 0.0;
      //double IAra,IAout1,DBout1,XGout1,S_heat,T_heat;
      double HEXrate;

      HEXrate = this.de_EX_S;

      //換気の時は全熱交換器は停止とする
      HEXrate = 0.0;

      //外気と室空気の混合空気の計算
      if( Airswc.isON( this.swcInOA )){//20101212nino
         this.DBin = (( this.de_Va_S - this.de_OA_S ) * this.DBra + (( this.DBoa - this.DBra ) * ( 1.0 - HEXrate ) +
this.DBra ) * this.de_OA_S ) / this.de_Va_S;
         this.XGin = (( this.de_Va_S - this.de_OA_S ) * this.XGra + (( this.XGoa - this.XGra ) * ( 1.0 - HEXrate ) +
this.XGra ) * this.de_OA_S ) / this.de_Va_S;
      }else{//20101212nino
         DBin = DBra;
         XGin = XGra;
```

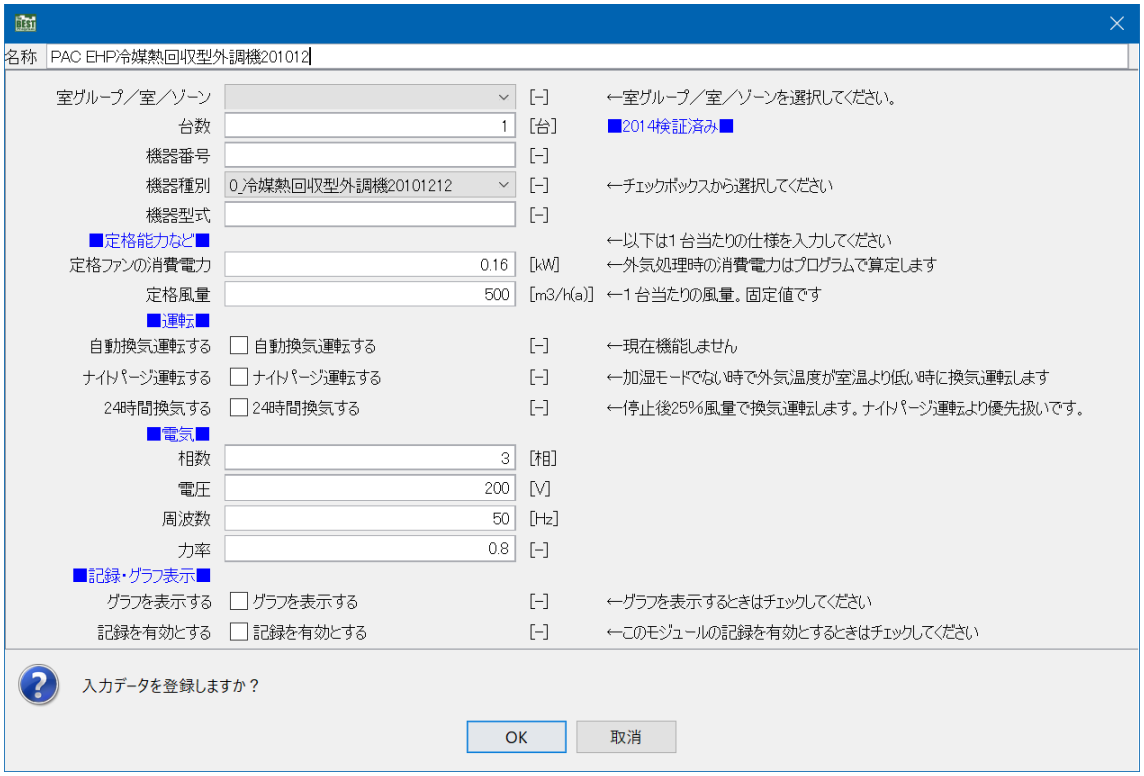
```
        }
        this.WBin = Psychrometrics.FNWbtx( this.DBin, this.XGin );
        //    System.out.println("C/DBin="+DBin+" "+XGin+"  ");
        //    System.out.println("C/DBoa="+DBoa+" "+XGoa+"  ");
        //    System.out.println("C/DBra="+DBra+" "+XGra+"  ");
        //    System.out.println("  ");

        //    System.out.println("DBra="+DBra+" "+XGra+"  ");

        this.opeCoolingLoadS = 0.0;
        this.opeCoolingLoadT = 0.0;
        this.opeCoolingLoadAIR = 0.0;
        this.opeCoolingLoadRAD = 0.0;
        this.opeHeatingLoadS = 0.0;
        this.opeHeatingLoadT = 0.0;
        this.opeHeatingLoadAIR = 0.0;
        this.opeHeatingLoadRAD = 0.0;
        this.opeLoadS = 0.0;
        this.opeLoadT = 0.0;

        this.DBout = this.DBin;//乾球温度
        this.XGout = this.XGin;//絶対湿度
        this.CW_Wrate = 0.0;

    }


}
```

「PAC EHP 冷媒熱回収型外調機 201012」（場所：設備 2015／個別分散 2015／）

モジュール名	PAC EHP 冷媒熱回収型外調機 201012
クラス	DesicaEHPModule20101212

（1）入力画面

・スペック



（2）モジュールの概要

冷媒熱回収型外調機 PAC のモジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

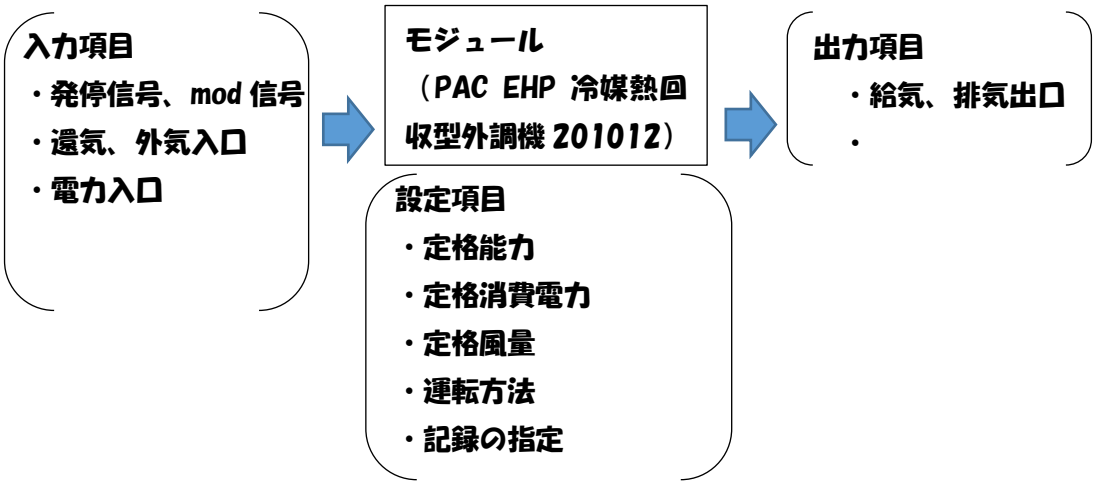


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	室グループ／室／ゾーン	String	zoneAirforSSinside	#RoomGroup	[-]	ー	ー		←室グループ／室／ゾーンを選択してください。
2	台数	double	in_dai	1	[台]	ー	0		
3	機器番号	String	m_no		[-]	ー	ー		
4	機器種別	String	m_ty	0_冷媒熱回収型外調機 20101212	[-]	ー	ー		←チェックボックスから選択してください
5	機器型式	String	m_kt		[-]	ー	ー		
6	■定格能力など■					ー	ー		←以下は１台当たりの仕様を入力してください
7	定格ファンの消費電力	double	in_PPEa_S	0.16	[kW]	ー	0		←外気処理時の消費電力はプログラムで算定します
8	定格風量	double	in_Va_S	500	[m3/h(a)]	ー	0		←１台当たりの風量。固定値です
9	■運転■					ー	ー		
10	＊自動換気運転する	boolean		FALSE	[-]	ー	ー		←現在機能しません
11	ナイトパージ運転する	boolean	isNightPurge	FALSE	[-]	ー	ー		←加湿モードでない時で外気温度が室温より低い時に換気運転します
12	24 時間換気する	boolean	isVent24	FALSE	[-]	ー	ー		←停止後 25%風量で換気運転します。ナイトパージ運転より優先扱いです。
13	■電気■					ー	ー		
14	相数	double	phase	3	[相]	ー	1		
15	電圧	double	voltage	200	[V]	ー	100		

16	周波数	double	frequency	50	[Hz]	60	50		
17	力率	double	powerFactor	0.8	[-]	1	0		
18	■記録・グラフ表示■					—	—		
19	グラフを表示する	boolean	isGVisible	FALSE	[-]	—	—		←グラフを表示するときはチェックしてください
20	記録を有効とする	boolean	isRecord	FALSE	[-]	—	—		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	外気導入信号入口	L1_swcInOA	swcInOA	-	制御	OnOff 信号	入口	
4	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
5	PID モード信号入口	L1_modInPID	modInPID	-	制御	制御モード	入口	
6	外気入口	L0_airInOA	airInOA	-	状態	空気	入口	
7	排気出口	L0_airOutEA	airOutEA	-	状態	空気	出口	
8	還気入口	L0_airInRA	airInRA	-	状態	空気	入口	
9	給気出口	L0_airOutSA	airOutSA	-	状態	空気	出口	
10	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	
11	操作量入口	L0_valInCtrl	valInCtrl	-	値	値	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	室内機 Message#-#-	メッセージ	—	メッセージ
2	冷却処理顕熱量合計#W#熱量	冷却処理顕熱量合計	W	My
3	冷却処理潜熱量合計#W#熱量	冷却処理潜熱量合計	W	My
4	冷却処理全熱量合計#W#熱量	冷却処理全熱量合計	W	My
5	加熱処理顕熱量合計#W#熱量	加熱処理顕熱量合計	W	My
6	加熱処理全熱量合計#W#熱量	加熱処理全熱量合計	W	My
7	加熱処理潜熱量合計#W#熱量	加熱処理潜熱量合計	W	My
8	処理全熱量合計負荷#W#-	処理全熱量合計負荷	W	負荷
9	消費電力#W#消費電力	消費電力	W	エネルギー消費
10	(ファン消費電力)#W#消費電力	(ファン消費電力)	W	My
11	室吹出乾球温度#℃#温度	室吹出乾球温度	℃	出口
12	室吹出絶対湿度#g/g#湿度	室吹出絶対湿度	g/s	出口
13	室吹出風量#g/s#質量流量	室吹出風量	g/s	出口
14	室吸込乾球温度#℃#温度	室吸込乾球温度	℃	入口
15	室吸込絶対湿度#g/g#湿度	室吸込絶対湿度	g/s	入口
16	外気乾球温度#℃#温度	外気乾球温度	℃	入口
17	外気絶対湿度#g/g#湿度	外気絶対湿度	g/g	入口
18	排気乾球温度#℃#温度	排気乾球温度	℃	出口
19	排気絶対湿度#g/g#湿度	排気絶対湿度	g/g	出口
20	COP#-#COP	COP	—	My
21	放熱量#W#放熱量	放熱量	W	My

（7）計算フロー・計算内容

省略・

（8）データ範囲と範囲外の取扱い

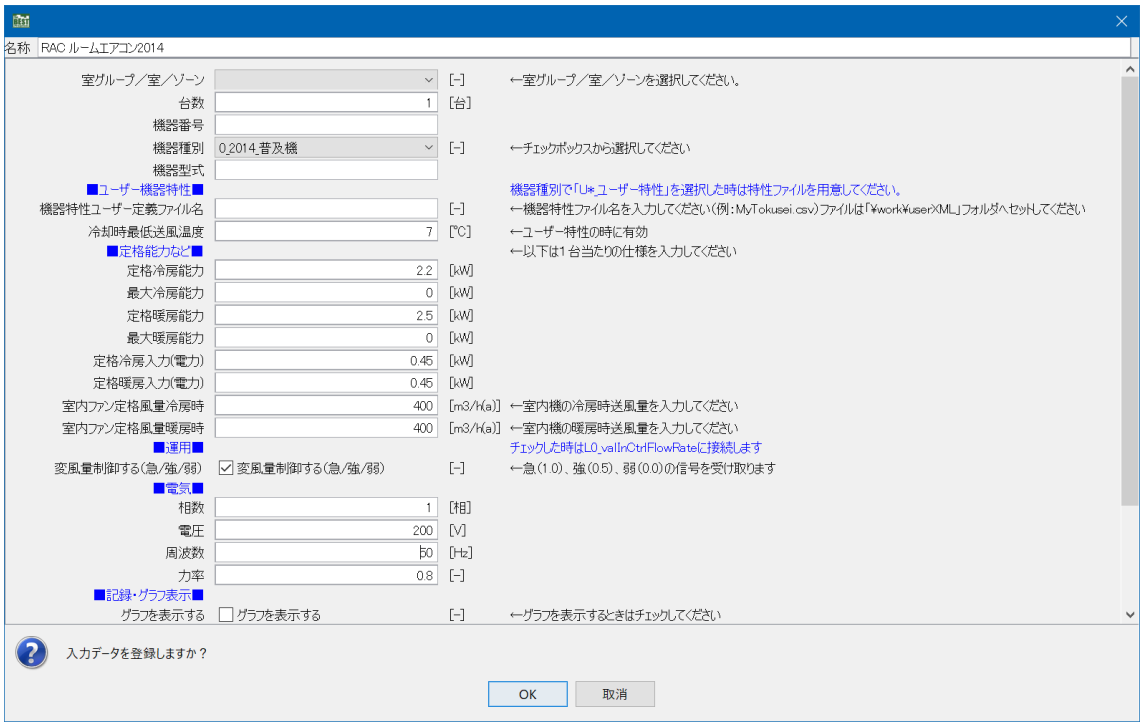
省略

「RAC ルームエアコン 2014」（場所：設備 2 015／個別分散 2015／）

モジュール名	RAC ルームエアコン 2014
クラス	RACModule201407

（1）入力画面

・スペック



（2）モジュールの概要

ルームエアコンのモジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

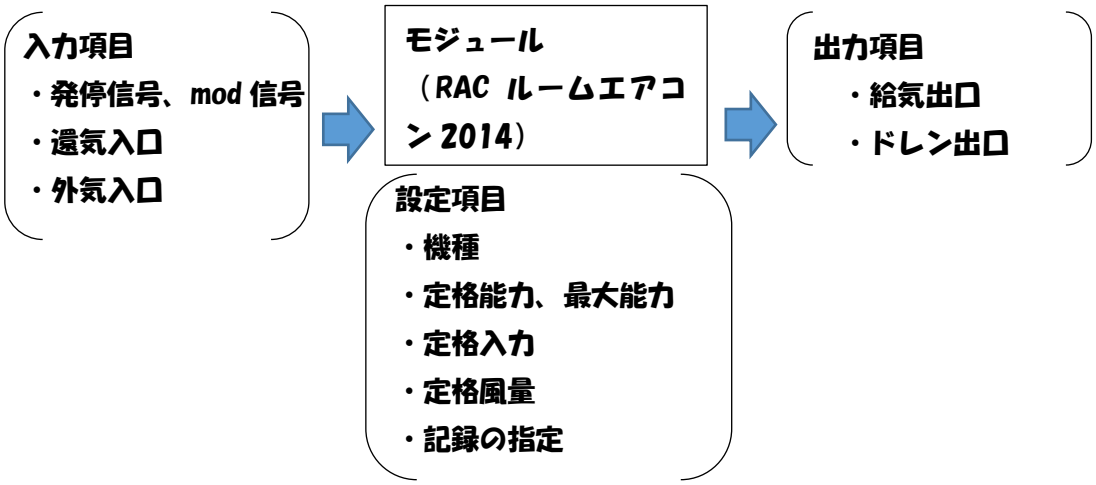


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	室グループ／室／ゾーン	String	zoneAirforSSinside	#RoomGroup	[-]	ー	ー		←室グループ／室／ゾーンを選択してください。
2	台数	double	in_dai	1	[台]	ー	0		
3	機器番号	String	m_no			ー	ー		
4	機器種別	String	m_ty	0_2014_普及機、1_2014_高性能機、U_ユーザー特性 Type1、U_ユーザー特性 Type2	[-]	ー	ー		←チェックボックスから選択してください
5	機器型式	String	m_kt			ー	ー		
6	■ユーザー機器特性■					ー	ー		
7	機器特性ユーザー定義ファイル名	String	uTokusei		[-]	ー	ー		←機器特性ファイル名を入力してください（例:MyTokusei.csv）ファイルは「¥work¥userXML」フォルダへセットしてください
8	冷却時最低送風温度	double	t_airOut_Lower	7	[℃]	ー	ー		←ユーザー特性の時に有効
9	■定格能力など■					ー	ー		←以下は１台当たりの仕様を入力してください
10	定格冷房能力	double	out_Qc_S	2.2	[kW]	ー	0		
11	最大冷房能力	double	out_Qc_Smax	0	[kW]	ー	0		
12	定格暖房能力	double	out_Qh_S	2.5	[kW]	ー	0		
13	最大暖房能力	double	out_Qh_Smax	0	[kW]	ー	0		

14	定格冷房入力(電力)	double	out_PPEc_S	0.45	[kW]	ー	0		
15	定格暖房入力(電力)	double	out_PPEh_S	0.45	[kW]	ー	0		
16	機器起動停止負荷率	double		10	[%]	100	0		←部分負荷率を入力してください
17	室内ファン定格風量冷房時	double	in_VaC_S	400	[m3/h(a)]	ー	0		←室内機の冷房時送風量を入力してください
18	室内ファン定格風量暖房時	double	in_VaH_S	400	[m3/h(a)]	ー	0		←室内機の暖房時送風量を入力してください
19	■運用■					ー	ー		
20	変風量制御する（急/強/弱）	boolean	isSVAV	TRUE	[-]	ー	ー		←急（1.0)、強（0.5)、弱（0.0）の信号を受け取ります
21	■電気■					ー	ー		
22	相数	double	phase	1	[相]	ー	1		
23	電圧	double	voltage	200	[V]	ー	100		
24	周波数	double	frequency	50	[Hz]	60	50		
25	力率	double	powerFactor	0.8	[-]	1	0		
26	■記録・グラフ表示■					ー	ー		
27	グラフを表示する	boolean	isGVisible	FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
28	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください
29	■機器特性■					ー	ー		
30	低負荷領域の計算方法	String	calcTypeLowerRangeLoad	0_発停運転、1_下限入力値固	[-]	ー	ー		←チェックボックスから選択してください

				定、2_下限COP 値 固定、3_下限入力値と中間切片					
31	■仮設調整■					ー	ー		
32	台数を調整する	boolean	isAdjust2012	FALSE	[-]	ー	ー		←台数を仮設調整するときはチェックしてください
33	調整の計算ステップ数	int	numAdjustSteps	18	[-]	ー	6		←仮設調整する計算ステップ数を入力してください
34									

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	外気入口	L0_airInOA	airInOA	-	状態	空気	入口	
5	排気出口	L0_airOutEA	airOutEA	-	状態	空気	出口	
6	還気入口	L0_airInRA	airInRA	-	状態	空気	入口	
7	給気出口	L0_airOutSA	airOutSA	-	状態	空気	出口	
8	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	
9	ドレイン出口	L0_watOutD	watOutD	-	状態	水	出口	
10	操作量入口	L0_valInCtrl	valInCtrl	-	値	値	入口	
11	制御風量入口	L0_valInCtrlFlowRate	valInCtrlFlowRate	-	値	値	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	RAC_Message#-#-	メッセージ	—	メッセージ
2	RAC_冷却処理顕熱量合計#W#熱量	冷却処理顕熱量の合計	W	負荷
3	RAC_加熱処理顕熱量合計#W#熱量	加熱処理顕熱量の合計	W	負荷
4	RAC_処理顕熱量#W#熱量";//加熱が正値	処理顕熱量	W	My
5	RAC_処理潜熱量#W#熱量";//加湿が正値	処理潜熱量	W	My
6	RAC_冷却処理全熱量合計#W#熱量	冷却処理全熱量の合計	W	My
7	RAC_加熱処理全熱量合計#W#熱量	加熱処理全熱量の合計	W	My
8	RAC_消費電力#W#消費電力	消費電力	W	エネルギー消費量
9	RAC_処理全熱量合計負荷#W#-	処理全熱量の合計負荷	W	My
10	RAC_COP#-#-	COP	—	My
11	RAC_COPc#-#-	COP（冷却）	—	My
12	RAC_COPh#-#-	COP（加熱）	—	My
13	RAC_PID 操作量#-#-	PID からの操作量	—	My
14	RAC_in 吹出乾球温度#℃#温度	吹き出し空気の乾球温度	℃	出口
15	RAC_in 吹出絶対湿度#g/g#湿度	吹き出し空気の絶対湿度	g/g	出口
16	RAC_in 吹出風量#g/s#質量流量	吹き出し空気の風量	g/s	出口
17	RAC_in ドレン量#g/s#質量流量	ドレン量	g/s	出口
18	RAC_in 還気乾球温度#℃#温度	還気の乾球温度	℃	入口
19	RAC_in 還気絶対湿度#g/g#湿度	還気の絶対湿度	g/g	入口
20	RAC_in 還気湿球温度#℃#温度	還気の湿球温度	℃	入口
21	RAC_out 外気乾球温度#℃#温度	外気の乾球温度	℃	My
22	RAC_out 外気絶対湿度#g/g#湿度	外気の絶対湿度	g/g	My
23	RAC_out 外気湿球温度#℃#温度	外気の湿球温度	℃	My
24	RAC_調整冷却能力#W#熱量	調整冷却能力	W	調整
25	RAC_調整加熱能力#W#熱量	調整加熱能力	W	調整
26	RAC_調整冷却台数#-#台数	調整冷却台数	—	調整
27	RAC_調整加熱台数#-#台数	調整加熱台数	—	調整
28	RAC_調整ステップ平均冷却台数#-#台数	調整ステップの平均冷却台数	—	調整
29	RAC_調整ステップ平均加熱台数#-#台数	調整ステップの平均加熱台数	—	調整

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.client.bestgui.file.service.FileConstants;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.essentials.utility.Psychrometrics;
import jp.or.ibec.best.log.BestLogger;
import jp.or.ibec.best.message.BestEngineMessageHandler;
import jp.or.ibec.best.message.BestEngineMessageParam;
import jp.or.ibec.best.message.constants.SystemMessageConstants;

/**
 * @author HIROSHI NINOMIYA/20140729
 *
 * ルームエアコン
 *
 * 201407機器特性
 *
 */
public class RACModule201407 extends AbstractBestModule implements
IBestMetaModule  {

   /**
    * 論理アドレス
    */
   private final String moduleName="(RACModule201407)";

   private final String S_NODE_airInOA  = "L0_airInOA";//外気
   private final String S_NODE_airOutEA = "L0_airOutEA";//外気

   private final String S_NODE_airInRA  = "L0_airInRA";//室内吸込口
   private final String S_NODE_airOutSA = "L0_airOutSA";//室内吹出し

   private final String S_NODE_watOutD  = "L0_watOutD";//コイル出口ドレン

   private final String S_NODE_valInPID = "L0_valInCtrl";//供給熱量 from PID           : in
   private final String S_NODE_valInFlowRate = "L0_valInCtrlFlowRate";//風量制御

   private final String S_NODE_eleIn    = "L0_eleIn";//電力

   private final String C_NODE_swcIn    = "L1_swcIn";//運転状態：off/on
   private final String C_NODE_swcInOA  = "L1_swcInOA";//OA状態：off/on
   private final String C_NODE_modIn    = "L1_modIn";//空調モード：停止/冷却/加熱
   private final String C_NODE_modInPID = "L1_modInPID";//PID空調モード：停止/冷却/加熱

   private final String R_NODE = "L2_recOut";



   /**
    * 外部定義
    */
   private final String SPEC_name        = "名称";
```

```
    private final String SPEC_grzName  = "室グループ／室／ゾーン";

    private final String SPEC_m_no        = "機器番号[-]";
    private final String SPEC_m_ty        = "機器種別[-]";
    private final String SPEC_m_kt        = "機器型式[-]";

    private final String SPEC_uTokusei    = "機器特性UserFile名";
    private final String SPEC_T_airOut_Lower = "冷却時最低送風温度[℃]";//20181119nino

    private final String SPEC_out_Qc_S    = "定格冷房能力[W]";
    private final String SPEC_out_Qh_S    = "定格暖房能力[W]";

    private final String SPEC_out_Qc_Smax = "最大冷房能力[W]";
    private final String SPEC_out_Qh_Smax = "最大暖房能力[W]";

    private final String SPEC_out_PPEc_S  = "定格冷房入力(電力)[W]";
    private final String SPEC_out_PPEh_S  = "定格暖房入力(電力)[W]";

//  private final String SPEC_fRate_airEA   = "風量[g/s]";

//  private final String SPEC_in_run_stop  = "機器起動停止負荷率[-]";//[%]

    private final String SPEC_Phase       = "相数[-]";
    private final String SPEC_Voltage     = "電圧[V]";
    private final String SPEC_Frequency   = "周波数[Hz]";
    private final String SPEC_PowerFactor = "力率[-]";
    //
    private final String SPEC_isGVisible = "グラフを表示する";
    private final String SPEC_isRecord = "記録を有効とする";

    private final String SPEC_CalcTypeLowerRangeLoad = "低負荷領域の計算方法";

    private final String SPEC_isAdjust2012   = "台数を調整する";
    private final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";

    private StringBuffer message= new StringBuffer();

    /**
     * 外部定義
     */

    private final String SPEC_in_dai    = "台数[-]";


//  private final String SPEC_in_VaCH_S = "定格風量[g/s]";
    private final String SPEC_in_VaC_S     = "定格風量冷房[g/s]";
    private final String SPEC_in_VaH_S     = "定格風量暖房[g/s]";
    private final String SPEC_isSVAV    = "変風量制御する";


//  private final String SPEC_in_PPEa_S = "定格ファン消費電力[W]";

//  private final String SPEC_in_Lp     = "冷媒管長[m]";
//  private final String SPEC_in_Lh     = "冷媒管高低差[m]";
//  private final String SPEC_in_HUM_mx    = "定格加湿能力[g/s]";
//  private final String SPEC_in_HUM_rt    = "加湿飽和効率[-]";
//  private final String SPEC_in_HUM_on    = "加湿On・Off設定値[-]";
//  private final String SPEC_in_OA_S      = "取入外気量[g/s]";


    /**
     * 変数（外部定義に対応）
     */

    private String name          ;// "機器名称";
    private String m_no          ;//= "機器番号";
    private String m_ty          ;//= "機器種別";
    private String m_kt          ;//= "機器型式";

    private double out_Qc_S      ;//= "定格冷房能力";//W
    private double out_Qc_Smax   ;//= "定格冷房能力";//W
```

```
    private double out_PPEc_S    ;//= "定格冷房入力(電力)";//W

    private double out_Qh_S      ;//= "定格暖房能力";//W
    private double out_Qh_Smax   ;//= "定格暖房能力";//W
    private double out_PPEh_S    ;//= "定格暖房入力(電力)";//W

// private double fRate_airEA ;//風量[g/s]

    private int phase;         //相数[-]
    private double voltage;    //電圧[V]
    private double frequency;  //周波数[Hz]
    private double powerFactor;   //力率[-]

    private boolean isGVisible = false;//このグラフを表示する=true
    private boolean isRecord   = false;//記録を有効とする=true


    /**
     * 変数（外部定義に対応）
     */

    private double in_dai         ;//= "台数";

// private double in_VaCH_S      ;//= "定格風量[g/s]";
    private double in_VaC_S    ;//= "定格風量冷房[g/s]";
    private double in_VaH_S    ;//= "定格風量暖房[g/s]";
// private double in_PPEa_S      ;//= "定格ファン消費電力";//W

    private double in_run_stop       ;//= "機器起動停止負荷率";//[%]

/* private double in_Lp          ;//= "冷媒管長";//m
    private double in_Lh          ;//= "冷媒管高低差";//m
*/
    private double in_HUM_mx          ;//= "定格加湿能力";//kg/h
    private double in_HUM_rt          ;//= "加湿飽和効率";//[%]
    private double in_HUM_on          ;//= "加湿On・Off設定値";//[%]
    private double in_OA_S =0.0;//= "取入外気量";//m3/h

    private BestAir airInOA  = null;
    private BestAir airOutEA = null;

    private BestElectricity eleIn = null;  //電力
// private Double CAPrateC = null;//室内機能力補正比率[-]冷房側
// private Double CAPrateH = null;//室内機能力補正比率[-]暖房側

    private PACDBManager pacDB=null;

    private int modIn  ;
    private int modInPID;
    private int swcIn ;
    private int swcInOA ;
    private int mState;  //停止、冷房、暖房フラグ



    private BestAir airInRA   = null;
    private BestAir airOutSA  = null;         //給気
    private BestWater watOutD = null;

    private BestValue PIDrate = null;
    private BestValue sVAVrate = null;
    private double vavRate = 1.0;//風量割合
// private Double LPrate = null;


    /**
     * 出力（建物）
     */
    private final String RECORD_message = "RAC_Message#-#-";

    private final String RECORD_ID1c    = "RAC_冷却処理顕熱量合計#W#熱量";
    private final String RECORD_ID1h    = "RAC_加熱処理顕熱量合計#W#熱量";
```

```
    private final String RECORD_ID_S    = "RAC_処理顕熱量#W#熱量";//加熱が正値
    private final String RECORD_ID_L    = "RAC_処理潜熱量#W#熱量";//加湿が正値

    private final String RECORD_ID2c    = "RAC_冷却処理全熱量合計#W#熱量";
    private final String RECORD_ID2h    = "RAC_加熱処理全熱量合計#W#熱量";
    private final String RECORD_ID4     = "RAC_消費電力#W#消費電力";


    private final String RECORD_qT      = "RAC_処理全熱量合計負荷#W#-";

    private final String RECORD_cop     = "RAC_COP#-#-";
    private final String RECORD_copC    = "RAC_COPc#-#-";
    private final String RECORD_copH    = "RAC_COPh#-#-";

    private final String RECORD_pidVal  = "RAC_PID操作量#-#-";


    /**
     * 出力（建物）
     */

    private final String RECORD_ID4i = "RAC_in吹出乾球温度#℃#温度";
    private final String RECORD_ID5i = "RAC_in吹出絶対湿度#g/g#湿度";
    private final String RECORD_ID6i = "RAC_in吹出風量#g/s#質量流量";
    private final String RECORD_ID7i = "RAC_inドレン量#g/s#質量流量";
    private final String RECORD_ID8i = "RAC_in加湿給水量#g/s#質量流量";

    private final String RECORD_DBra = "RAC_in還気乾球温度#℃#温度";
    private final String RECORD_Xra  = "RAC_in還気絶対湿度#g/g#湿度";
    private final String RECORD_WBra = "RAC_in還気湿球温度#℃#温度";

    private final String RECORD_DBin = "RAC_in入口乾球温度#℃#温度";
    private final String RECORD_Xin  = "RAC_in入口絶対湿度#g/g#湿度";
    private final String RECORD_WBin = "RAC_in入口湿球温度#℃#温度";

    private final String RECORD_DBoa = "RAC_out外気乾球温度#℃#温度";
    private final String RECORD_Xoa  = "RAC_out外気絶対湿度#g/g#湿度";
    private final String RECORD_WBoa = "RAC_out外気湿球温度#℃#温度";



    private final String RECORD_out_Qc_Adjust = "RAC_調整冷却能力#W#熱量";
    private final String RECORD_out_Qh_Adjust = "RAC_調整加熱能力#W#熱量";
    private final String RECORD_in_daicAdjust = "RAC_調整冷却台数#-#台数";
    private final String RECORD_in_daihAdjust = "RAC_調整加熱台数#-#台数";
    private final String RECORD_avecdaiAdjust = "RAC_調整ステップ平均冷却台数#-#台数";
    private final String RECORD_avehdaiAdjust = "RAC_調整ステップ平均加熱台数#-#台数";

    private double DBra;          //室内乾球温度[℃]
    private double WBra;          //室内湿球温度[℃]
    private double XGra;
//  private double IAra;
    private double RHra;
    private double CO2ppmra;

    private double DBoa;          //外気乾球温度[℃]
    private double WBoa;          //外気湿球温度[℃]
    private double XGoa;
//  private double IAoa;
    private double CO2ppmoa;

    private double DBin;          //室内機吸込乾球温度[℃]
    private double WBin;          //室内機吸込湿球温度[℃]
    private double XGin;
    private double IAin;
    private double CO2ppmin;

    private double DBout;           //乾球温度[℃]
    private double XGout;           //絶対湿度[kg/kg.DA]
    private double CO2ppmout;
```

```
    private double GW;              //風量(g/s)
    private double D_Wrate;       //ドレン水量(g/s)
    private double CW_Wrate;        //加湿水量(g/s)

    private double PPE_out;
    private double E_PPE_out;
    private double PPE_in;
    private double E_PPE_in;
// private double S_Load;
// private double T_Load;
    private double Sc_Load;
    private double Tc_Load;
    private double Sh_Load;
    private double Th_Load;

    private double Sc_Load_RM;//室内機処理熱量（顕熱）[W]
    private double Tc_Load_RM;//室内機処理熱量（全熱）[W]
    private double Sh_Load_RM;//室内機処理熱量（顕熱）[W]
    private double Th_Load_RM;//室内機処理熱量（全熱）[W]

    private double saijohatsu;
    private double T_C_heat;


//*****************090725-e
    private int dbflg=0;
    private String path="EHP";
    private String[] filenames= new String[1];
    private String equipmentName;


    //グラフ表示など
    private GraphJFrameBuilMultiIn_S20101212 gBMIn = null;

    //低負荷領域の計算方法
    private String calcTypeLowerRangeLoad = null;//"低負荷領域の計算方法";
    private int num_calcTypeLowerRangeLoad;//"低負荷領域の計算方法";

    /**
     * その他変数
     */

// private double Rc;              //部分負荷率
// private double Rh;              //部分負荷率

    private double Sc_Loadout;//室外機処理可能熱量（冷却全熱）[W]
    private double Sh_Loadout;//室外機処理可能熱量（加熱全熱）[W]
// private double Tc_Loadout;//室外機実処理熱量（冷却全熱）[W]
// private double Th_Loadout;//室外機実処理熱量（加熱全熱）[W]
    private double allSc_heat;//室内機処理熱量（冷却顕熱）[W]
    private double allSh_heat;//室内機処理熱量（加熱顕熱）[W]
    private double allTc_heat;//室内機処理熱量（冷却全熱）[W]
    private double allTh_heat;//室内機処理熱量（加熱全熱）[W]

    private String kiki_file;
    private final double CP = 1.006;//空気比熱[J/gK]

// private double Lratemin;
// private double fmc;
// private double pmc;
// private double fmh;
// private double pmh;

    private int iType = 0;
    private final int iType_STANDARD = 0;
    private final int iType_HIGHEFI = 1;

    private String uTokusei = null;//     = "機器特性UserFile名";
    private boolean isUserTokusei = false;//ユーザー定義の機器特性の場合=true

    private double t_airOut_Lower;//20181119nino
```

```
    //グラフ表示など
    private GraphJFrameBuilMultiOut_S20101212 gBMout = null;

    //仮設調整モード2012

    private boolean isAdjust2012  = false;//true="台数を調整する";//
    //private boolean isAdjust2012 = false;//20120406nino 建築学会大会空衛学会OS論文検討用＝true
    private int numAdjustSteps;//"調整の計算ステップ数";
    private LinkedList<Double> daiListc = null;//必要台数
    private LinkedList<Double> daiListh = null;//必要台数
    private double maxcAdjustdai = 1;
    private double maxhAdjustdai = 1;
    private double avecdaiAdjust = 0;
    private double avehdaiAdjust = 0;
    private double in_daicAdjust = 1;//= "調整台数";
    private double in_daihAdjust = 1;//= "調整台数";
    private double in_run_stop_Num;//= "機器起動停止負荷率"台数補正;

    private double out_Qc_S_1      ;//= "定格冷房能力";//kW
    private double out_Qc_Smax_1      ;//= "定格冷房能力";//kW
    private double out_PPEc_S_1    ;//= "定格冷房入力(電力)";//kW
    private double out_Qh_S_1      ;//= "定格暖房能力";//kW
    private double out_Qh_Smax_1      ;//= "定格暖房能力";//kW
    private double out_PPEh_S_1    ;//= "定格暖房入力(電力)";//kW
    private double BF;//バイパスファクター

    private double in_VaC_S_1     ;//= "定格風量冷房[g/s]";//
    private double in_VaH_S_1     ;//= "定格風量暖房[g/s]";//

    private boolean isSVAV;//多段　変風量制御する
// private double in_PPEa_S_1    ;//= "定格ファン消費電力";//W

// private double in_HUM_mx_1    ;//= "定格加湿能力";//kg/h

    private double max_Qc_rate;
    private double max_Qh_rate;

    private double max_vavRate = 1.0;//急風の風量比率
    private double mid_vavRate;//強風の風量比率
    private double min_vavRate;//弱風の風量比率



    double[][] pLRListC = {
          {      0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1., 1.1 },
          { 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1., 1.1, 1.2, 1.3 }
          }; //[0]普及機[1]高効率機

    String[][] pLRstrListC = {
          {             "RACKcw02", "RACKcw03", "RACKcw04", "RACKcw05", "RACKcw07", "RACKcw08", "RACKcw09", "RACKcw10",
"RACKcw11" },
          { "RACKcw01", "RACKcw02", "RACKcw03", "RACKcw04", "RACKcw05", "RACKcw07", "RACKcw08", "RACKcw09", "RACKcw10",
"RACKcw11", "RACKcw12", "RACKcw13" }
          }; //[0]普及機[1]高効率機

    double[][] pLRListH = {
          {      0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1., 1.1, 1.2,       1.5 },
          { 0.1, 0.2, 0.3, 0.4, 0.5, 0.7, 0.8, 0.9, 1., 1.1, 1.2,       1.5, 1.8, 2., 2.2 }
          }; //[0]普及機[1]高効率機

    String[][] pLRstrListH = {
          {             "RACKhw02", "RACKhw03", "RACKhw04", "RACKhw05", "RACKhw07", "RACKhw08", "RACKhw09", "RACKhw10",
"RACKhw11", "RACKhw12",  "RACKhw15"},
          { "RACKhw01", "RACKhw02", "RACKhw03", "RACKhw04", "RACKhw05", "RACKhw07", "RACKhw08", "RACKhw09", "RACKhw10",
"RACKhw11", "RACKhw12",  "RACKhw15", "RACKhw18", "RACKhw20", "RACKhw22" }
          }; //[0]普及機[1]高効率機

    int n_pLR = 0;
    double cop = 0;
    double copC = 0;
    double copH = 0;
```

```
    private ZoneAirforSSinsideModule201502 zoneAirforSSinside = null;
    private String grzName = null;
    private boolean isUseZoneAir = false;


    @Override
    public void setProfile(BestSpecs spec) {
//    外部定義項目取得
      if(spec == null) {
         return;
      }
      Map<String,String> map=spec.getSpec();
      if(map == null) {
         return;
      }

      // 機器名称
      this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

      this.grzName = spec.getSpecValue( this.SPEC_grzName, "" );
      if( !this.grzName.equals( "" )){
         this.isUseZoneAir = true;
      }

      if( this.isUseZoneAir ){
         this.zoneAirforSSinside = new ZoneAirforSSinsideModule201502();
         this.zoneAirforSSinside._setProfile( this.name, this.grzName );
      }

      // 機器番号
      this.m_no = spec.getSpecValue( this.SPEC_m_no, "" );

      //  機器種別  0_標準型、1_インバータ
      this.m_ty = spec.getSpecValue( this.SPEC_m_ty, "0_2014_普及機" );

      // SPEC_uTokusei      = "機器特性UserFile名";
      this.uTokusei = spec.getSpecValue( this.SPEC_uTokusei, "" );

      //
      if( this.m_ty.equals( "0_2014_普及機" )){
         this.iType = this.iType_STANDARD;
         this.kiki_file ="RACstnd2014";
         this.n_pLR = 0;
         this.isUserTokusei = false;
      }else if( this.m_ty.equals( "1_2014_高性能機" ) || this.m_ty.equals( "1_2014_高効率機" )){
         this.iType = this.iType_HIGHEFI;
         this.kiki_file ="RAChigh2014";
         this.n_pLR = 1;
         this.isUserTokusei = false;
      }
      //機器特性追加20181108  ユーザー定義の機器特性ファイルを使用する
      if( this.m_ty.equals("U_ユーザー特性Type1")){
         this.iType = this.iType_STANDARD;
         this.kiki_file = this.uTokusei;
         this.n_pLR = 0;
         this.isUserTokusei = true;

         //外部定義項目が初期化されていなければ何もしない
         if(this.uTokusei == null || this.uTokusei.equals("")){
            System.out.println( "(E)["+this.name+"]ファイルが読めません /" + this.uTokusei );
            if( this.isRecord ){
               this.message.append( "(E)ファイルが読めません /" + this.uTokusei );
            }
            BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                  SystemMessageConstants.CAL_ERROR2 ,
                  new String[]{ this.name, "(E)ファイル["+ this.uTokusei+"]が読めません" } ));
            BestLogger.info( "(E)ファイル["+ this.uTokusei +"]が読めません" );
            return;
         }

           this.kiki_file = FileConstants.DIR_USER + "/userXML/"+this.uTokusei;
```

```
}
//
if( this.m_ty.equals("U_ユーザー特性Type2")){
   this.iType = this.iType_HIGHEFI;
   this.kiki_file = this.uTokusei;
   this.n_pLR = 1;
   this.isUserTokusei = true;

   //外部定義項目が初期化されていなければ何もしない
   if(this.uTokusei == null || this.uTokusei.equals("")){
      System.out.println( "(E)["+this.name+"]ファイルが読めません /" + this.uTokusei );
      if( this.isRecord ){
         this.message.append( "(E)ファイルが読めません /" + this.uTokusei );
      }
      BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
            SystemMessageConstants.CAL_ERROR2 ,
            new String[]{ this.name, "(E)ファイル["+ this.uTokusei+"]が読めません" } ));
      BestLogger.info( "(E)ファイル["+ this.uTokusei +"]が読めません" );
      return;
   }

     this.kiki_file = FileConstants.DIR_USER + "/userXML/"+this.uTokusei;

}
if( this.isUserTokusei ) {
   this.t_airOut_Lower = spec.getSpecValue( this.SPEC_T_airOut_Lower, 7. );//20181119nino
}else {
   this.t_airOut_Lower = 7.;
}

//
this.max_Qc_rate = this.pLRListC[ this.n_pLR ][ this.pLRListC[ this.n_pLR ].length - 1 ];
this.max_Qh_rate = this.pLRListH[ this.n_pLR ][ this.pLRListH[ this.n_pLR ].length - 1 ];

System.out.println(" max_Qc_rate ="+max_Qc_rate+", max_Qh_rate ="+max_Qh_rate );

// 機器型式
this.m_kt = spec.getSpecValue( this.SPEC_m_kt, "" );

//    台数
this.in_dai = spec.getSpecValue( this.SPEC_in_dai, 1 );

// 定格冷房能力
this.out_Qc_S   = spec.getSpecValue( this.SPEC_out_Qc_S, 0. ) * in_dai;
this.out_Qc_S_1 = spec.getSpecValue( this.SPEC_out_Qc_S, 0. );
this.out_Qc_Smax  = spec.getSpecValue( this.SPEC_out_Qc_Smax, 0. ) * in_dai;
this.out_Qc_Smax_1= spec.getSpecValue( this.SPEC_out_Qc_Smax, 0. );

if( this.out_Qc_Smax == 0 ){
   this.out_Qc_Smax   = this.out_Qc_S * this.max_Qc_rate;
   this.out_Qc_Smax_1 = this.out_Qc_S_1 * this.max_Qc_rate;
}

// 定格冷房入力(電力)
this.out_PPEc_S   = spec.getSpecValue( this.SPEC_out_PPEc_S, 0. ) * in_dai;
this.out_PPEc_S_1 = spec.getSpecValue( this.SPEC_out_PPEc_S, 0. );

// 定格暖房能力
this.out_Qh_S   = spec.getSpecValue( this.SPEC_out_Qh_S, 0. ) * in_dai;
this.out_Qh_S_1 = spec.getSpecValue( this.SPEC_out_Qh_S, 0. );
this.out_Qh_Smax   = spec.getSpecValue( this.SPEC_out_Qh_Smax, 0. ) * in_dai;
this.out_Qh_Smax_1 = spec.getSpecValue( this.SPEC_out_Qh_Smax, 0. );

if( this.out_Qh_Smax == 0 ){
   this.out_Qh_Smax   = this.out_Qh_S * this.max_Qh_rate;
   this.out_Qh_Smax_1 = this.out_Qh_S_1 * this.max_Qh_rate;
}

// 定格暖房入力(電力)
this.out_PPEh_S   = spec.getSpecValue( this.SPEC_out_PPEh_S, 0. ) * in_dai;
this.out_PPEh_S_1 = spec.getSpecValue( this.SPEC_out_PPEh_S, 0. );
```

```
        // 相数
        this.phase = spec.getSpecValue( this.SPEC_Phase, 3 );

        // 電圧
        this.voltage = spec.getSpecValue( this.SPEC_Voltage, 200. );

        // 周波数
        this.frequency = spec.getSpecValue( this.SPEC_Frequency, 50. );

        // 力率
        this.powerFactor = spec.getSpecValue( this.SPEC_PowerFactor, 0.8 );


        //    機器名称
        //this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

        // 定格風量
//      this.in_VaCH_S  = spec.getSpecValue( this.SPEC_in_VaCH_S, 0. );
        this.in_VaC_S   = spec.getSpecValue( this.SPEC_in_VaC_S, 0.) * in_dai;
        this.in_VaC_S_1 = spec.getSpecValue( this.SPEC_in_VaC_S, 0.);
        this.in_VaH_S   = spec.getSpecValue( this.SPEC_in_VaH_S, 0.) * in_dai;
        this.in_VaH_S_1 = spec.getSpecValue( this.SPEC_in_VaH_S, 0.);

        if( this.in_VaC_S <= 0 && this.out_Qc_S > 0){
           BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                 SystemMessageConstants.CAL_ERROR2 ,
                 new String[]{ "("+this.moduleName+"_outputs)["+this.name+"]",  "冷房風量を設定してください"} ));

        }
        if( this.in_VaH_S <= 0 && this.out_Qh_S > 0){
           BestEngineMessageHandler.addMessage( new BestEngineMessageParam(
                 SystemMessageConstants.CAL_ERROR2 ,
                 new String[]{ "("+this.moduleName+"_outputs)["+this.name+"]",  "暖房風量を設定してください"} ));

        }

        //変風量制御「急」「強」「弱」する
        this.isSVAV = spec.getSpecValue(this.SPEC_isSVAV, false);

           this.mid_vavRate = 0.87;// 強風の風量比率
           this.min_vavRate = 0.73;// 弱風の風量比率

           // バイパスファクター
        double QS_bs_upper =7.1;//kW 上限
        double QS_bs_lower = 2.2;//kW 下限
        double QS_bs;//kW
        if(this.out_Qc_S / 1000<QS_bs_lower) {
           QS_bs = QS_bs_lower;
        }else if(QS_bs_upper<this.out_Qc_S / 1000) {
           QS_bs = QS_bs_upper;
        }else {
           QS_bs = this.out_Qc_S / 1000;
        }
        this.BF = 0.0035 * Math.pow(QS_bs, 2) - 0.0346 * QS_bs + 0.118;// 川津さんより


        // 定格ファン消費電力
//      this.in_PPEa_S   = spec.getSpecValue( this.SPEC_in_PPEa_S, 0. ) * in_dai;
//      this.in_PPEa_S_1 = spec.getSpecValue( this.SPEC_in_PPEa_S, 0. );

        // 定格加湿能力
//      this.in_HUM_mx   = spec.getSpecValue( this.SPEC_in_HUM_mx, 0. ) * in_dai;
//      this.in_HUM_mx_1 = spec.getSpecValue( this.SPEC_in_HUM_mx, 0. );

        // 加湿飽和効率
//      this.in_HUM_rt = spec.getSpecValue( this.SPEC_in_HUM_rt, 0.7 )*100.0;

        // 加湿器ONOFF湿度
//      this.in_HUM_on = spec.getSpecValue( this.SPEC_in_HUM_on, 0.4 )*100.0;
```

```
        // 取入外気量
//      this.in_OA_S = spec.getSpecValue( this.SPEC_in_OA_S, 0. ) * in_dai;

        //isGVisibleを取得
        this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

        //低負荷領域の計算方法
        this.calcTypeLowerRangeLoad = spec.getSpecValue( this.SPEC_CalcTypeLowerRangeLoad, "1_下限入力値固定" );
        //calcTypeLowerRangeLoad
        //  0_発停運転
        //  1_下限入力値固定
        //  2_下限COP値固定
        //  3_下限入力値と中間切片
        if( this.calcTypeLowerRangeLoad.equals( "0_発停運転" )){
           this.num_calcTypeLowerRangeLoad = 0;
        }else if(this.calcTypeLowerRangeLoad.equals( "1_下限入力値固定" )){
           this.num_calcTypeLowerRangeLoad = 1;
        }else if(this.calcTypeLowerRangeLoad.equals( "2_下限COP値固定" )){
           this.num_calcTypeLowerRangeLoad = 2;
        }else if(this.calcTypeLowerRangeLoad.equals( "3_下限入力値と中間切片" )){
           this.num_calcTypeLowerRangeLoad = 3;
        }else{
           this.num_calcTypeLowerRangeLoad = 0;
        }

        //isAdjust2012 = "台数を調整する";//
        this.isAdjust2012 = spec.getSpecValue( this.SPEC_isAdjust2012, false );

        // 調整の計算ステップ数
        this.numAdjustSteps = spec.getSpecValue( this.SPEC_NumAdjustSteps, 12 );

        this.daiListc = new LinkedList<Double>();
        this.daiListh = new LinkedList<Double>();
        for( int i=0; i<this.numAdjustSteps; i++ ){
           this.daiListc.add( 0. );
           this.daiListh.add( 0. );
        }


    }

    @Override
    public void initialize(IBestStateMessage stateNodes,
          IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
          IBestDomainSchedule schedule) {
        //System.out.println(this.moduleName + " BuillMultiOutイニシャライズ");
        //状態ノードを受け取る
        super.sm = stateNodes;
        //制御ノードを受け取る
        super.cm = commandNodes;
        //記録ノードを受け取る
        super.rm = recordNodes;

        //eleIn
        this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

        //airInOA
        this.airInOA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInOA );
        this.airInOA.setMaxFlowRate( this.in_OA_S );//20150423

        //airOutEA
        this.airOutEA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutEA );
        this.airOutEA.setMaxFlowRate( this.in_OA_S );//20150423

        //グラフ表示の準備
        if( this.isGVisible ){
           gBMout = new GraphJFrameBuilMultiOut_S20101212( this.name, 300,  out_Qc_S * 1.5, 0 );
        }
```

```
      this.filenames[0] = this.kiki_file;
      this.equipmentName = this.kiki_file;
      this.pacDB=new PACDBManager(this.path, this.filenames, this.isUserTokusei);
      this.pacDB.setEquipmentName(this.equipmentName);

      //airOutRM
      this.airOutSA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutSA );
      this.airOutSA.setMaxFlowRate( Math.max( this.in_VaC_S, this.in_VaH_S ));//20150423
      if( this.isUseZoneAir ){
         this.zoneAirforSSinside.checkNumberAirChange( this.in_VaC_S, "in_VaC_S", this.moduleName, "initialize()",
this.name, this.isRecord, this.message);
         this.zoneAirforSSinside.checkNumberAirChange( this.in_VaH_S, "in_VaH_S", this.moduleName, "initialize()",
this.name, this.isRecord, this.message);
         this.zoneAirforSSinside._initialize( );
      }

      //watOutD
      this.watOutD = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutD );


      this.airInRA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInRA );
      this.airInRA.setMaxFlowRate( Math.max( this.in_VaC_S, this.in_VaH_S ));//20150423

      //S_NODE_valInPID
      this.PIDrate = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInPID );

      //S_NODE_valInFlowRate
      this.sVAVrate = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInFlowRate );



      //グラフ表示の準備
      if( this.isGVisible ){
         gBMIn = new GraphJFrameBuilMultiIn_S20101212( this.name, 300, this.out_Qc_S * 1.5, Math.max( this.in_VaC_S,
this.in_VaH_S ) * 1.5 );
      }

   }


   /**
    * EHP計算及び結果出力/各室内機よりも先に動きます。
    */
   @Override
   public void outputs() {

      if (null == super.cm || null == super.sm ){
         return;
      }


      /**
       * 入力
       */
      //---20060211-----------------------------------------------
      this.allSc_heat = 0.0;//室内機処理熱量（冷却顕熱）[W]
      this.allTc_heat = 0.0;//室内機処理熱量（冷却全熱）[W]
      this.allSh_heat = 0.0;//室内機処理熱量（加熱顕熱）[W]
      this.allTh_heat = 0.0;//室内機処理熱量（加熱全熱）[W]

      this.PPE_in = 0;
      this.PPE_out = 0;

      if(dbflg==0){
         //filenames[0]=RMdata.get_kiki();
         //equipmentName=RMdata.get_kiki();
         //pacDB=new PACDBManager(path,filenames);
         //pacDB.setEquipmentName(equipmentName);
      }
      dbflg = 1;
```

```
//
this.airInRA = (BestAir)super.sm.getState(super.getConnectionNode( this.S_NODE_airInRA ));
if( this.isUseZoneAir ){
   this.zoneAirforSSinside._outputSetRA( this.airInRA );
}

this.airInOA = (BestAir)super.sm.getState(super.getConnectionNode( this.S_NODE_airInOA ));

this.swcIn    = super.cm.getCommand(super.getConnectionNode( this.C_NODE_swcIn ));
this.swcInOA  = super.cm.getCommand(super.getConnectionNode( this.C_NODE_swcInOA ));
this.modIn    = super.cm.getCommand(super.getConnectionNode( this.C_NODE_modIn ));
this.modInPID = super.cm.getCommand(super.getConnectionNode( this.C_NODE_modInPID ));

/**
 * 入力
 */

//状態ノードから室内空気を設定
//室内乾球温度を設定
this.DBra = airInRA.getTempDB();

//室内絶対湿度を設定
this.XGra = airInRA.getHumi();

//室内湿球温度を設定
this.WBra = airInRA.getTempWB();

//室内CO2濃度を設定
this.CO2ppmra = airInRA.getCO2ppm();

//    System.out.println("DBin="+DBin+" "+XGin+"  "+WBin+"  ");

//外気乾球温度設定
this.DBoa = airInOA.getTempDB();

//外気絶対湿度を設定
this.XGoa = airInOA.getHumi();

//外気湿球温度を設定
this.WBoa = airInOA.getTempWB();
if( WBoa < -9000.0 ) WBoa =WB_cal(this.DBoa, this.XGoa);

//外気CO2濃度を設定
this.CO2ppmoa = airInOA.getCO2ppm();

//PID操作量
this.PIDrate = (BestValue)super.sm.getState( super.getConnectionNode( this.S_NODE_valInPID ));

//    System.out.println("mState="+mState+" "+onOff+"  "+mode);
//    System.out.println("DBoa="+DBoa+" "+XGoa+"  "+WBoa);

//多段変風量制御をする
if(this.isSVAV) {
   this.sVAVrate = (BestValue)super.sm.getState( super.getConnectionNode( this.S_NODE_valInFlowRate ));
   //弱風
   if(this.sVAVrate.getValue() ==0.0) {
      this.vavRate= this.min_vavRate;
   //強風
   }else if(this.sVAVrate.getValue() ==0.5) {
      this.vavRate= this.mid_vavRate;
   //急風
   }else if(this.sVAVrate.getValue() ==1.0) {
      this.vavRate= this.max_vavRate;
   }
}else {
   this.vavRate = this.max_vavRate;
}

//
if( Airswc.isOFF( this.swcIn )){
   //停止
```

```
            this.mState = 0;
            this.message.append( "(C)停止");
        }else if( Airswc.isON( this.swcIn )){
            //運転
            if( Airmod.isCOOL( this.modIn ) && Airmod.isHEAT( this.modIn ) ){
                //冷暖同時
                if( Airmod.isCOOL( this.modInPID ) ){
                    this.mState = 1;
                    this.message.append( "(C)PID冷暖同時の冷房");
                }else if(  Airmod.isHEAT( this.modInPID ) ){
                    this.mState = 2;
                    this.message.append( "(C)PID冷暖同時の暖房");
                }else{
                    this.mState = 1;
                }
            }else if( Airmod.isCOOL( this.modIn )){
                //冷房運転
                this.mState = 1;
                this.message.append( "(C)冷房");
            }else if( Airmod.isHEAT( this.modIn )){
                //暖房運転
                this.mState = 2;
                this.message.append( "(C)暖房");
            }else if( Airmod.isVENTILATE( this.modIn )){
                //換気モード
                this.mState = 3;
                this.message.append( "(C)換気");
            }else{
                //停止
                this.mState = 0;
                this.message.append( "(C)停止");
            }
            BestAir.checkOpeData( this.airInRA, "airInRA", this.moduleName, "outputs()", this.name, this.isRecord,
this.message );

        }



        //計算の切替

        //室内機側の計算
        switch(this.mState){
        case 0:
            //停止
            this.message.append("(C)停止");
            this.calc_Stop_In();
            //
            break;

        case 1:
            //冷房
            if( this.out_Qc_S_1 > 0 ){
                this.message.append("(C)冷房運転");
                this.cooling_cal_In();
                this.PPE_in = 0.;//this.in_PPEa_S;
            }else{
                this.message.append("(C)冷能力=0停止");
                this.calc_Stop_In();
                this.mState = 0;
            }
            break;

        case 2:
            //暖房
            if( this.out_Qh_S_1 > 0 ){
                this.message.append("(C)暖房運転");
                this.heating_cal_In();
                this.PPE_in = 0.;//this.in_PPEa_S;
            }else{
                this.message.append("(C)暖能力=0停止");
                this.calc_Stop_In();
```

```
      this.mState = 0;
   }
   break;

case 3:
   //換気
   this.message.append("(C)換気運転");
   this.ventilation_cal();
   this.PPE_in = 0.;//this.in_PPEa_S;
   break;

default:
   this.message.append("(C)運転モード？停止");
   this.calc_Stop_In();
   this.mState = 0;
   //System.out.println(this.moduleName + ">>Error<< onOff*modeが範囲外");
}

this.allSc_heat = this.Sc_Load;
this.allSh_heat = this.Sh_Load;
this.allTc_heat = this.Tc_Load;
this.allTh_heat = this.Th_Load;

//室外機側の計算
super.sm.setState( super.getConnectionNode( this.S_NODE_airOutEA ), this.airOutEA );


this.E_PPE_in = this.PPE_in * Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 );
this.E_PPE_out= this.PPE_out* Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 );
this.eleIn.setActivePower( this.PPE_out + this.PPE_in );
this.eleIn.setReactivePower(this.E_PPE_out + this.E_PPE_in );
this.eleIn.setPhase( this.phase );
this.eleIn.setVoltage( this.voltage );
this.eleIn.setFrequency( this.frequency );


this.Sc_Load_RM = this.Sc_Load;
this.Tc_Load_RM = this.Tc_Load;
this.Sh_Load_RM = this.Sh_Load;
this.Th_Load_RM = this.Th_Load;

//室内機側の再計算
switch( this.mState ){
case 0:
   //停止
   this.DBout = DBin;//乾球温度
   this.XGout = XGin;//絶対湿度
   this.GW = 0.0;
   this.D_Wrate = 0.0;
   this.CW_Wrate = 0.0;
   break;

case 1:
   //冷房
   this.GW = this.in_VaC_S*this.vavRate;
   this.cooling_supply();
   this.D_Wrate = ( this.XGin - this.XGout ) * this.GW;//g/s
   if( this.D_Wrate < 0.0 ){
      this.D_Wrate = 0.0;//追加090619
   }
   this.CW_Wrate = 0.0;
   break;

case 2:
   //暖房
   this.GW = this.in_VaH_S*this.vavRate;
   this.heating_supply();
   this.D_Wrate = 0.0;
   break;

case 3:
   //換気
```

```
        this.GW = this.in_VaC_S*this.vavRate;
        this.ventilation_supply();
        this.D_Wrate = 0.0;
        //this.S_Load_RM = 0.0;
        //this.T_Load_RM = 0.0;
        break;

    default:
        System.out.println( this.moduleName + ">>Error<< onOff*modeが範囲外");
    }
    //    System.out.println("DBout="+DBout);

    if( this.PPE_out > 0 ){
        if( this.Tc_Load > 0 ){
            this.cop = this.Tc_Load / this.PPE_out;
            this.copC = this.cop;
            this.copH = 0;
        }else{
            this.cop = this.Th_Load / this.PPE_out;
            this.copH = this.cop;
            this.copC = 0;
        }
    }else{
        this.cop = 0;
        this.copC = 0;
        this.copH = 0;
    }

    this.airOutSA.setTempDB( this.DBout );
    this.airOutSA.setHumi( this.XGout );
    this.airOutSA.setFlowRate(this.GW);
    this.airOutSA.setCO2ppm( this.CO2ppmout );//20131031
    this.airInRA.setFlowRate(this.GW);

    this.watOutD.setFlowRate( this.D_Wrate );
    this.watOutD.setTemp( this.DBout );
    //    System.out.println("DBout="+DBout+" "+XGout+"  "+airOutRM.getFlowRate());

    //出力側ノード
    super.sm.setState( super.getConnectionNode( this.S_NODE_airOutSA ), this.airOutSA);
    super.sm.setState( super.getConnectionNode( this.S_NODE_eleIn ), this.eleIn);
    super.sm.setState( super.getConnectionNode( this.S_NODE_watOutD ), this.watOutD);

    //グラフデータ追加
    if( this.isGVisible ){
        gBMIn.addData( BestTimeManager.getDateWeatherTime(),
                this.Sc_Load_RM,
                this.Tc_Load_RM,
                this.Sh_Load_RM,
                this.Th_Load_RM,
                this.PPE_in,
                this.DBoa,
                this.DBra,
                this.DBin,
                this.DBout,
                this.XGoa,
                this.XGra,
                this.XGin,
                this.XGout,
                this.GW,
                this.D_Wrate );
    }


    //出力設定

    //グラフデータ追加
    if( this.isGVisible ){
        gBMout.addData( BestTimeManager.getDateWeatherTime(),
        this.Sc_Loadout,//室外機処理可能熱量（冷却）[W]
        this.Sh_Loadout,//室外機処理可能熱量（加熱）[W]
        this.allTc_heat,//室内機処理要求熱量（冷却）[W]
```

```
            this.allTh_heat,//室内機処理要求熱量（加熱）[W]
            this.PPE_out,
            this.DBoa,
            this.DBin,
            this.XGoa,
            this.XGin,
            1. );
        }

        //記録ノード
        if( this.isRecord && super.rm != null ){
            this.record();
        }

        message.setLength(0);



        //*************************----------------
        if( this.isAdjust2012 ){
            //移動平均の最大値で調整していく
            if( true ){
                this.daiListc.removeLast();
                this.daiListc.addFirst( this.in_daicAdjust * this.PIDrate.getValue() );
                //
                double sum_in_dai = 0;
                for( int i=0; i<this.numAdjustSteps; i++ ){
                    sum_in_dai += this.daiListc.get( i );
                }
                this.avecdaiAdjust = sum_in_dai / this.numAdjustSteps;
                //
                if( sum_in_dai > this.maxcAdjustdai * this.numAdjustSteps ){
                    this.maxcAdjustdai = this.avecdaiAdjust;

                    this.in_daicAdjust = this.maxcAdjustdai;
                    this.in_run_stop_Num = this.in_run_stop / this.in_daicAdjust;

                    this.out_Qc_S   = this.out_Qc_S_1   * this.in_daicAdjust;//= "定格冷房能力";//W
                    this.out_Qc_Smax= this.out_Qc_Smax_1* this.in_daicAdjust;//= "定格冷房能力";//W
                    this.out_PPEc_S = this.out_PPEc_S_1 * this.in_daicAdjust;//= "定格冷房入力(電力)";//W

//                  private double fRate_airEAc ;//風量[g/s]
//                  private double fRate_airEAh ;//風量[g/s]

                    this.in_VaC_S   = this.in_VaC_S_1   * this.in_daicAdjust;//= "定格風量";//m3/h
//                  this.in_PPEa_S = this.in_PPEa_S_1 * this.in_daicAdjust;//= "定格ファン消費電力";//W

//                  this.in_HUM_mx = this.in_HUM_mx_1 * this.in_daicAdjust;//= "定格加湿能力";//kg/h

                    this.airInRA.setFlowRate(this.in_VaC_S * this.vavRate);
                    this.airOutSA.setFlowRate(this.in_VaC_S * this.vavRate);
                }
            }


            if( true ){
                this.daiListh.removeLast();
                this.daiListh.addFirst( this.in_daihAdjust * this.PIDrate.getValue() );
                //
                double sum_in_dai = 0;
                for( int i=0; i<this.numAdjustSteps; i++ ){
                    sum_in_dai += this.daiListh.get( i );
                }
                this.avehdaiAdjust = sum_in_dai / this.numAdjustSteps;
                //
                if( sum_in_dai > this.maxhAdjustdai * this.numAdjustSteps ){
                    this.maxhAdjustdai = this.avehdaiAdjust;

                    this.in_daihAdjust = this.maxhAdjustdai;
                    this.in_run_stop_Num = this.in_run_stop / this.in_daihAdjust;

                    this.out_Qh_S   = this.out_Qh_S_1   * this.in_daihAdjust;//= "定格暖房能力";//W
```

```
                this.out_Qh_Smax= this.out_Qh_Smax_1* this.in_daihAdjust;//= "定格暖房能力";//W
                this.out_PPEh_S = this.out_PPEh_S_1 * this.in_daihAdjust;//= "定格暖房入力(電力)";//W

                this.in_VaH_S   = this.in_VaH_S_1   * this.in_daihAdjust;//= "定格風量";//m3/h
//              this.in_PPEa_S = this.in_PPEa_S_1 * this.in_daihAdjust;//= "定格ファン消費電力";//W

//              this.in_HUM_mx = this.in_HUM_mx_1 * this.in_daihAdjust;//= "定格加湿能力";//kg/h

                this.airInRA.setFlowRate(this.in_VaH_S * this.vavRate);
                this.airOutSA.setFlowRate(this.in_VaH_S * this.vavRate);
            }

        }
    }
    //*************************----------------

}


private void calc_Stop_In(){
    //出口空気の状態は入口空気の状態とする
    this.airOutSA.copyAllvalState( this.airInRA );
    this.airOutSA.setFlowRate(0.0);
    this.Sc_Load = 0.0;
    this.Sh_Load = 0.0;
    this.Tc_Load = 0.0;
    this.Th_Load = 0.0;
    //LPrate=1.0;
    this.PPE_in = 0.;
    this.DBin = this.DBra;
    this.XGin = this.XGra;
    this.WBin = this.WBra;
    this.CO2ppmin = this.CO2ppmra;//20131031
}


/**
 * 記録
 */
private void record(){

    if( super.rm != null ){
        if( CheckPrintModule.isPrintMessage ){
            //message
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_message, this.name, message.toString() );
        }

        if( CheckPrintModule.isPrintEnergy ){
            //記録ノードに室外機消費電力を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID4, this.name,
AirFormat.df_2(this.PPE_out));
        }

        if( CheckPrintModule.isPrintLoad ){
            //記録ノードに室外機処理熱量（顕熱・潜熱）を設定
            double S_Load = - this.Sc_Load + this.Sh_Load;//加熱が正値
            double L_Load = (-this.Tc_Load + this.Th_Load) - S_Load;//加湿が正値
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID_S, this.name,
AirFormat.df_2(S_Load));//処理顕熱量#W#熱量
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID_L, this.name,
AirFormat.df_2(L_Load));//処理潜熱量#W#熱量
        }

        if( CheckPrintModule.isPrintStateOut ){
            //記録ノードに吹出口乾球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID4i, this.name,
AirFormat.df_2(this.airOutSA.getTempDB()));
            //記録ノードに吹出口絶対湿度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID5i, this.name,
AirFormat.df_5(this.airOutSA.getHumi()));
            //記録ノードに吹出口風量を設定
```

```
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID6i, this.name,
AirFormat.df_2(this.airOutSA.getFlowRate()));
            //記録ノードにドレン量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID7i, this.name,
AirFormat.df_2(D_Wrate));
         }

         if( CheckPrintModule.isPrintStateMy ){
            //記録ノードに室外機処理熱量（全熱）を設定
            if( this.Tc_Load > 0 ){
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT, this.name, AirFormat.df_2(-
this.Tc_Load ));//室外機処理全熱量合計#W#熱量
            }else{
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_qT, this.name,
AirFormat.df_2(this.Th_Load ));//室外機処理全熱量合計#W#熱量
            }
            //記録ノードに室内機処理熱量（顕熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID1c, this.name,
AirFormat.df_2(this.allSc_heat));//室内機冷却処理顕熱量合計#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID1h, this.name,
AirFormat.df_2(this.allSh_heat));//室内機加熱処理顕熱量合計#W#熱量
            //記録ノードに室内機処理熱量（全熱）を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID2c, this.name,
AirFormat.df_2(this.allTc_heat));//室内機冷却処理全熱量合計#W#熱量
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID2h, this.name,
AirFormat.df_2(this.allTh_heat));//室内機加熱処理全熱量合計#W#熱量

            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_cop, this.name,
AirFormat.df_4(this.cop));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_copC, this.name,
AirFormat.df_4(this.copC));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_copH, this.name,
AirFormat.df_4(this.copH));

            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_pidVal, this.name,
AirFormat.df_4(this.PIDrate.getValue()));

            //記録ノードに入口乾球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBoa, this.name, AirFormat.df_2(DBoa));
            //記録ノードに入口絶対湿度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_Xoa, this.name, AirFormat.df_5(XGoa));
            //記録ノードに入口湿球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_WBoa, this.name, AirFormat.df_2(WBoa));
         }


         if( CheckPrintModule.isPrintStateIn ){
/*          //記録ノードに加湿給水量を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_ID8i, this.name,
AirFormat.df_2(CW_Wrate));
*/          //記録ノードに還気乾球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBra, this.name, AirFormat.df_2(DBra));
            //記録ノードに還気絶対湿度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_Xra, this.name, AirFormat.df_5(XGra));
            //記録ノードに還気湿球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_WBra, this.name, AirFormat.df_2(WBra));
/*          //記録ノードに入口乾球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_DBin, this.name, AirFormat.df_2(DBin));
            //記録ノードに入口絶対湿度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_Xin, this.name, AirFormat.df_5(XGin));
            //記録ノードに入口湿球温度を設定
            super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_WBin, this.name,
AirFormat.df_2(WBin));*/
         }

         if( CheckPrintModule.isPrintAdjust ){
            //
            if( this.isAdjust2012 ){
               //記録ノードに室外機調整能力を設定
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qc_Adjust, this.name,
this.out_Qc_S );//室外機調整能力[W]
               super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_out_Qh_Adjust, this.name,
```

```
this.out_Qh_S );//室外機調整能力[W]
                //記録ノードに室外機調整能力を設定
                //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rcAdjust", this.name,
this.out_rcAdjust );//室外機調整率[-]
                //super.rm.setRecord(super.getConnectionNode(this.R_NODE), "out_rhAdjust", this.name,
this.out_rhAdjust );//室外機調整率[-]
                //記録ノードに調整台数を設定
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_in_daicAdjust, this.name,
AirFormat.df_2(this.in_daicAdjust) );
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_in_daihAdjust, this.name,
AirFormat.df_2(this.in_daihAdjust) );
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_avecdaiAdjust, this.name,
AirFormat.df_2(this.avecdaiAdjust) );
                super.rm.setRecord(super.getConnectionNode(this.R_NODE), this.RECORD_avehdaiAdjust, this.name,
AirFormat.df_2(this.avehdaiAdjust) );
            }
        }
    }
}

/**
 * ウォールスルーEHP計算/各室内機の集計後
 */

@Override
public void update() {
    if( this.isUseZoneAir ){
        this.zoneAirforSSinside._update( this.airOutSA );
    }
}

public Object viewInternal(TestCommand cmd) {

    ArrayList<Object> result = new ArrayList<Object>();
    result.add(super.sm);
    result.add(super.cm);
    result.add(super.rm);
    //外部定義
    result.add(this.name);
    result.add(this.m_kt);
    result.add(this.m_no);
    result.add(this.in_run_stop_Num);

    result.add(this.out_Qc_S);
    result.add(this.out_PPEc_S);
    result.add(this.out_Qh_S);
    result.add(this.out_PPEh_S);

    result.add(this.DBoa);
    result.add(this.XGoa);
    result.add(this.DBin);
    result.add(this.XGin);

    return result;
}



private double WB_cal(double ddb,double xx){
    double wwb,x1,de=1;
    int kk=-1,j=0;
    wwb=ddb;

    do{
        j++;
        x1=Psychrometrics.FNXtw(ddb,wwb);
        if(Math.abs(x1-xx)<0.000003) {
            //          System.out.println(" j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
            return wwb;
        }
        if(x1<xx) {
```

```
            if(kk==-1) de=de/2.0;
            wwb=wwb+de;
            kk=1;
        }
            if(x1>=xx) {
            if(kk==1) de=de/2.0;
            wwb=wwb-de;
            kk=-1;
        }

    }while(j<1000);
    //    System.out.println("*j="+j+" wwb="+wwb+" X1="+x1+" xx="+xx+" de="+de);
    return wwb;

}



private void cooling_cal_In(){
    //室側の冷房計算
    double heatCapacity=0.0;
    double IAout1,DBout1,XGout1;
    double S_heat,T_heat;
    double Rp;

        //吸込み状態を求める※in_OA_S = 0
        this.DBin = this.DBra;
        this.XGin = this.XGra;
        this.CO2ppmin = this.CO2ppmra;
        this.DBin = this.DBra;//20130109
        this.XGin = this.XGra;//20130109
        this.CO2ppmin = this.CO2ppmra;//20131031

    this.WBin = Psychrometrics.FNWbtx( this.DBin, this.XGin );

    // System.out.println("C/DBin="+DBin+" "+XGin+"  ");
    // System.out.println("C/DBoa="+DBoa+" "+XGoa+"  ");
    // System.out.println("C/DBra="+DBra+" "+XGra+"  ");
    // System.out.println("  ");
    // System.out.println("DBra="+DBra+" "+XGra+"  ");

    //上下限のチェック 20120821nino
    double dbOA = this.DBoa;
    double wbIn = this.WBin;

    double dCheck = this.pacDB.getformulaRangeMaxRaw( this.pacDB.getFRACKcw("RACKcw10")  );
    if( dbOA > dCheck ){//getFACKcta()
        //外気乾球温度＞上限の時停止
        this.message.append( "(C)外気DB＞上限"+dCheck+"→停止");
        //this.S_Load = 0.0;
        //this.T_Load = 0.0;
        this.Sc_Load = 0.0;
        this.Tc_Load = 0.0;
        this.Sh_Load = 0.0;
        this.Th_Load = 0.0;
        this.PPE_out = 0.0;
        return;
    }
    //System.out.println( "(C)外気DB上限"+dCheck );

    dCheck = this.pacDB.getformulaRangeMinRaw( this.pacDB.getFRACKcw("RACKcw10") );
    if( dbOA < dCheck ){//getFACKcta()
        //外気乾球温度＜下限の時　下限値の特性で運転
        this.message.append( "(C)外気DB＜下限"+dCheck+"→下限値で特性計算");
        dbOA = dCheck;//this.pacDB.getformulaRangeMin( this.pacDB.getFACKcta() );
    }
    //System.out.println( "(C)外気DB下限"+dCheck );

    dCheck = this.pacDB.getformulaRangeMaxCol( this.pacDB.getFRACKcw("RACKcw10") );//getFACKcti()
    if( wbIn > dCheck ){
        //室内湿球温度＞上限の時　上限値の特性で運転
        this.message.append( "(C)室WB＞上限"+dCheck+"→上限値で特性計算");
```

```
            wbIn = dCheck;//this.pacDB.getformulaRangeMax( this.pacDB.getFACKcti() );
        }
        //System.out.println( "(C)室WB上限"+dCheck );

        dCheck = this.pacDB.getformulaRangeMinCol( this.pacDB.getFRACKcw("RACKcw10") );//getFACKcti()
        if( wbIn < dCheck ){
            //室内湿球温度＜下限の時　下限値の特性で運転
            this.message.append( "(C)室WB＜下限"+dCheck+"→下限値で特性計算");
            wbIn = dCheck;//this.pacDB.getformulaRangeMin( this.pacDB.getFACKcti() );
        }
        //System.out.println( "(C)室WB下限"+dCheck );


        //PIDからの負荷率
        double rc = this.PIDrate.getValue();

        //heatCapacity = this.out_Qc_S;
        heatCapacity = this.out_Qc_Smax;

        if( heatCapacity < 0 ){
            heatCapacity = 0;
        }


        this.IAin   = Psychrometrics.FNH( this.DBin, this.XGin );//J/kg
        IAout1 = this.IAin - heatCapacity * rc / (this.in_VaC_S / 1000.0 * this.vavRate);// 負荷率をここで適用するように変
更20130702
        //ルームエアコンは　バイパスファクターBFで吹き出しポイントを求める
        double IAout100 = this.IAin - ( this.IAin - IAout1 ) / (1.0-this.BF);
        double DBout100 = Psychrometrics.FNDbrh( 100.0, IAout100 );//DBout1 = Psychrometrics.FNDbrh( 90.0, IAout1 );//乾球
温度
        double XGout100 = Psychrometrics.FNxtr( DBout100, 100.0);//XGout1 = Psychrometrics.FNxtr( DBout1, 90.0);//絶対湿度

        DBout1 = DBout100 + (this.DBin - DBout100) * this.BF;
        XGout1 = XGout100 + ( XGin - XGout100 ) * this.BF;

        //加湿はされない
        if( this.XGin < XGout1 ){
            XGout1 = this.XGin;
            DBout1 = Psychrometrics.FNDbxh( XGout1, IAout1 );

        // 最低出口温度を7℃とする。//180119_iida
        //}else if (DBout1 < 7. ) {
        }else if (DBout1 < this.t_airOut_Lower ) {//20181119nion
            DBout1 = 7.0;
            DBout100 = this.DBin - (this.DBin - DBout1) / (1.0 - this.BF);
            XGout100 = Psychrometrics.FNxtr(DBout100, 100.0);// 絶対湿度-更新
            IAout100 = Psychrometrics.FNH(DBout100, XGout100);// エンタルピーJ/kg-更新
            IAout1 = IAout100 + (this.IAin - IAout100) * this.BF;
            XGout1 = XGout100 + ( XGin - XGout100 ) * this.BF;
            // rcの更新(>0)
            rc = (this.IAin - IAout1) / heatCapacity * (this.in_VaC_S/1000.0 * this.vavRate);
            if(rc<0) rc=0;
        }


        // System.out.println("IAra="+IAra+" "+IAout1+"  "+MHP_in_Vc_S);
        this.message.append( "(C)BF="+this.BF);



        //
        if( rc > 1. ){
            rc = 1.;
            this.message.append( "(W)負荷率＞上限　上限で計算");
        }

        this.in_run_stop = this.pLRListC[ this.n_pLR ][0];

        //低負荷領域の計算仕分け:Rh
        if( rc < this.in_run_stop  / this.max_Qc_rate ){
            switch( this.num_calcTypeLowerRangeLoad){
```

```
            case 0://  0_発停運転
               Rp = 0;
               break;

            case 1://  1_下限入力値固定
            case 2://  2_下限COP値固定
            case 3://  3_下限入力値と中間切片
               Rp = this.in_run_stop / this.max_Qc_rate;
               break;

            default:
               Rp = 0;
            }
         }else{
            Rp = rc;
         }
         //


         S_heat = ( this.DBin - DBout1 ) *this.CP* this.in_VaC_S*this.vavRate;
//       S_heat = ( this.DBin - DBout1 ) * this.in_VaC_S*this.vavRate;
         if( S_heat > heatCapacity * rc ){
            S_heat = heatCapacity * rc;
            this.message.append( "(C)S_heat補正");
         }
         if( Math.abs( this.XGin - XGout1 ) < 0.00001 ){
//          T_heat = S_heat;
            T_heat = heatCapacity * rc;
            this.message.append( "(C)T_heat補正１");
         }else{
            T_heat = heatCapacity * rc;
            this.message.append( "(C)T_heat補正２");
         }

         this.T_C_heat = T_heat;

         //this.S_Load = S_heat;//負荷率を削除20130702
         //this.T_Load = T_heat;//負荷率を削除20130702
         this.Sc_Load = S_heat;
         this.Tc_Load = T_heat;
         this.Sh_Load = 0.0;
         this.Th_Load = 0.0;

         if( this.PIDrate.getValue() < this.in_run_stop / this.max_Qc_rate ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
               //this.S_Load = 0.0;
               //this.T_Load = 0.0;
               this.Sc_Load = 0.0;
               this.Tc_Load = 0.0;
               this.Sh_Load = 0.0;
               this.Th_Load = 0.0;
               break;

            case 1://  1_下限入力値固定
               break;

            case 2://  2_下限COP値固定
               break;

            case 3://  3_下限入力値と中間切片
               break;

            default:
            }
         }
         // System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate);

         //this.PPE_out = rCode0 *rCode1 * rCode2 * rCode3 * this.out_PPEc_S;
         this.PPE_out = this.out_PPEc_S * this.pacDB.getRACKcw( dbOA, wbIn, rc * this.max_Qc_rate,
this.pLRListC[ this.n_pLR ], this.pLRstrListC[ this.n_pLR ] );
```

```
   //低負荷領域の計算仕分け:PPE
   if( rc < this.in_run_stop / this.max_Qc_rate && Rp > 0 ){
      switch( this.num_calcTypeLowerRangeLoad){
      case 0://  0_発停運転
         this.PPE_out = 0.0;
         break;

      case 1://  1_下限入力値固定
         //何もしない
         break;

      case 2://  2_下限COP値固定
         this.PPE_out *= ( rc / Rp );
         break;

      case 3://  3_下限入力値と中間切片
         this.PPE_out *= ( 0.5 + rc / Rp / 2. );
         break;

      default:
      }
   }else{
      //何もしない
   }

   if( rc == 0 ){
      this.PPE_out = 10;
   }


}



static double dbOA;
static double wbIn;
static double maxCapacity;
static double opeCapacity;

static double rCode0;
static double rCode1;
static double rCode2;
static double rCode3;
static double IAout1;

static double rc;//負荷率
static double s_Load;//室内機処理負荷顕熱
static double t_Load;//室内機処理負荷全熱

static double alph;
static double beta;

static double ppe_out;


static double wbOA;
static double dbIn;

static double rh;


private void heating_cal_In(){
   //室側の暖房計算
   double heatCapacity=0.0;
   double S_heat;
   double Rp;

   this.DBin = this.DBra;// 20130109
   this.XGin = this.XGra;// 20130109
   this.CO2ppmin = this.CO2ppmra;// 20131031
```

```
this.WBin = Psychrometrics.FNWbtx(this.DBin, this.XGin);// 20130109

// System.out.println("H/DBin="+DBin+" "+XGin+"  ");
// System.out.println("H/DBoa="+DBoa+" "+XGoa+"  ");
// System.out.println("H/DBra="+DBra+" "+XGra+"  ");
// System.out.println("  ");

//上下限のチェック 20120821nino
double wbOA = this.WBoa;
double dbIn = this.DBin;

double dCheck = this.pacDB.getformulaRangeMaxRaw( this.pacDB.getFRACKcw( "RACKhw10" )  );
if( wbOA > dCheck ){//this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() )
   //外気湿球温度＞上限の時　上限値の特性で運転
   this.message.append( "(C)外気WB＞上限"+dCheck+"→上限値で特性計算");
   wbOA = dCheck;//this.pacDB.getformulaRangeMax( this.pacDB.getFACKhta() );
}
//System.out.println( "(C)外気WB上限"+dCheck );

dCheck = this.pacDB.getformulaRangeMinRaw( this.pacDB.getFRACKcw( "RACKhw10" )  );
if( wbOA < dCheck ){//this.pacDB.getformulaRangeMin( this.pacDB.getFACKhta() )
   //外気湿球温度＜下限の時　停止
   this.message.append( "(C)外気WB＜下限"+dCheck+"→停止");
   //this.S_Load = 0.0;
   //this.T_Load = 0.0;
   this.Sc_Load = 0.0;
   this.Tc_Load = 0.0;
   this.Sh_Load = 0.0;
   this.Th_Load = 0.0;
   this.PPE_out = 0;
   return;
}
//System.out.println( "(C)外気WB下限"+dCheck );

dCheck = this.pacDB.getformulaRangeMaxCol( this.pacDB.getFRACKcw( "RACKhw10" )  );
if( dbIn > dCheck ){//this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() )
   //室内乾球温度＞上限の時　上限値の特性で運転
   this.message.append( "(C)室DB＞上限"+dCheck+"→上限値で特性計算");
   dbIn = dCheck;//this.pacDB.getformulaRangeMax( this.pacDB.getFACKhti() );
}
//System.out.println( "(C)室DB上限"+dCheck );

dCheck = this.pacDB.getformulaRangeMinCol( this.pacDB.getFRACKcw( "RACKhw10" )  );
if( dbIn < dCheck ){//this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() )
   //室内乾球温度＜下限の時　下限値の特性で運転
   this.message.append( "(C)室DB＜下限"+dCheck+"→下限値で特性計算");
   dbIn = dCheck;//this.pacDB.getformulaRangeMin( this.pacDB.getFACKhti() );
}
//System.out.println( "(C)室DB下限"+dCheck );


double rh = this.PIDrate.getValue();//負荷率をここで適用することに変更20130702

//if( rh > this.max_Qh_rate ){
// rh = this.max_Qh_rate;
if( rh > 1. ){
   rh = 1.;
   this.message.append( "(W)負荷率＞上限　上限で計算");
}

this.in_run_stop = this.pLRListH[ this.n_pLR ][0];

//低負荷領域の計算仕分け:Rh
if( rh < this.in_run_stop / this.max_Qh_rate ){
   switch( this.num_calcTypeLowerRangeLoad){
   case 0://  0_発停運転
      Rp = 0;
      break;

   case 1://  1_下限入力値固定
   case 2://  2_下限COP値固定
```

```
            case 3://  3_下限入力値と中間切片
               Rp = this.in_run_stop / this.max_Qh_rate ;
               break;

            default:
               Rp = 0;
            }
         }else{
            Rp = rh;
         }

         //heatCapacity = this.out_Qh_S;
         heatCapacity = this.out_Qh_Smax;

         if( heatCapacity < 0 ){
            heatCapacity = 0;
         }

      // IAin   = Psychrometrics.FNH( this.DBin, this.XGin );
      // IAout1 = IAin + heatCapacity * rh / this.in_Va_S * 1000.0;//負荷率をここで適用することに変更20130702
      //  XGout1 = this.XGin;
      // DBout1 = Psychrometrics.FNDbxh( XGout1, IAout1 );
         S_heat = heatCapacity  * rh;

         //this.S_Load = S_heat;//負荷率削除2130702
         //this.T_Load = heatCapacity  * rh;//負荷率削除2130702
         this.Sc_Load = 0.0;
         this.Tc_Load = 0.0;
         this.Sh_Load = S_heat;
         this.Th_Load = heatCapacity  * rh;

         if( this.PIDrate.getValue() < this.in_run_stop / this.max_Qh_rate  ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
               //this.S_Load = 0.0;
               //this.T_Load = 0.0;
               this.Sc_Load = 0.0;
               this.Tc_Load = 0.0;
               this.Sh_Load = 0.0;
               this.Th_Load = 0.0;
               break;

            case 1://  1_下限入力値固定
               break;

            case 2://  2_下限COP値固定
               break;

            case 3://  3_下限入力値と中間切片
               break;

            default:
            }
         }

         // System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate+" DBout="+DBout1);

         this.PPE_out = this.out_PPEh_S * this.pacDB.getRACKcw( wbOA, dbIn, rh * this.max_Qh_rate,
this.pLRListH[ this.n_pLR ], this.pLRstrListH[ this.n_pLR ] );


         //低負荷領域の計算仕分け:PPE
         if( rh < this.in_run_stop / this.max_Qh_rate  && Rp > 0 ){
            switch( this.num_calcTypeLowerRangeLoad){
            case 0://  0_発停運転
               this.PPE_out = 0.0;
               break;

            case 1://  1_下限入力値固定
               //何もしない
               break;
```

```
            case 2://  2_下限COP値固定
               this.PPE_out *= ( rh / Rp );
               break;

            case 3://  3_下限入力値と中間切片
               this.PPE_out *= ( 0.5 + rh / Rp / 2. );
               break;

            default:
            }
         }else{
            //何もしない
         }

         if( rh == 0 ){
            this.PPE_out = 10;
         }


      }


      private void ventilation_cal(){
         //double heatCapacity=0.0,rCode1 = 0.0,rCode2 = 0.0,rCode3 = 0.0,rCode4 = 0.0,rCode5 = 0.0,rCode6 = 0.0,rCode7 =
   0.0,rCode8 = 0.0;
         //double IAra,IAout1,DBout1,XGout1,S_heat,T_heat;

            this.DBin = this.DBra;//20130109
            this.XGin = this.XGra;//20130109
            this.CO2ppmin = this.CO2ppmra;//20131031

         this.WBin = Psychrometrics.FNWbtx( this.DBin, this.XGin );
         //    System.out.println("C/DBin="+DBin+" "+XGin+"  ");
         //    System.out.println("C/DBoa="+DBoa+" "+XGoa+"  ");
         //    System.out.println("C/DBra="+DBra+" "+XGra+"  ");
         //    System.out.println("  ");

         //    System.out.println("DBra="+DBra+" "+XGra+"  ");

   //    this.LPrate = 1.0;

         this.T_C_heat = 0.0;

         this.Sc_Load = 0.0;
         this.Tc_Load = 0.0;
         this.Sh_Load = 0.0;
         this.Th_Load = 0.0;

         // System.out.println("S_Load="+S_Load+" T_Load="+T_Load+" PIDrate="+PIDrate.getValue());
      }


      private void cooling_supply(){
         /*
            double IAra,IAout;
            IAra=Psychrometrics.FNH(DBra,XGra);
            IAout=IAra-T_Load_RM/in_Va_S*1000.0;
            DBout=DBra-S_Load_RM/in_Va_S;//乾球温度
            XGout= Psychrometrics.FNXth(DBout, IAout);//絶対湿度
          */
         //******************090725-s
         double IAout,sRc;
         double SS_rm;

         //IAin = Psychrometrics.FNH( this.DBin, this.XGin );
         if( this.T_C_heat > 0.0 ){
            sRc = this.Tc_Load_RM / this.T_C_heat;
         }else{
            sRc = 0.0;
         }

      // if( this.numCalcSA == 1 ){//20141110BESTEST
```

```
   //    if( this.isFanOnOff ){
   //       SS_rm = this.Sc_Load_RM - this.in_PPEac_S * this.PIDrate.getValue();
   //       if( SS_rm > this.Tc_Load_RM - this.in_PPEac_S * this.PIDrate.getValue() ){
   //          SS_rm = this.Tc_Load_RM - this.in_PPEac_S * this.PIDrate.getValue();
   //       }
   //    }else{
   //       SS_rm = this.Sc_Load_RM - this.in_PPEac_S;
   //       if( SS_rm > this.Tc_Load_RM - this.in_PPEac_S ){
   //          SS_rm = this.Tc_Load_RM - this.in_PPEac_S;
   //       }
   //    }
   // }else{
         //IAout = IAin - this.Tc_Load_RM / this.GW * 1000.0;//20130109
         SS_rm = this.Sc_Load_RM + ( this.Tc_Load_RM - this.Sc_Load_RM ) * ( 1.0 - sRc ) * this.saijohatsu / 100.0;

         if( SS_rm > this.Tc_Load_RM ){
            SS_rm = this.Tc_Load_RM;
         }
   // }

      this.DBout = this.DBin - SS_rm / ( this.GW * 1.006 );//乾球温度//20130109
      this.XGout = this.XGin - ( this.Tc_Load_RM - this.Sc_Load_RM ) / 2501. / this.GW;
      if( this.XGout > this.XGin ){
         System.out.println( "     77777777777777 sa="+(this.XGout- this.XGin));
         this.XGout = this.XGin;
      }

   // IAout = IAin - this.Tc_Load_RM / this.in_VaC_S * 1000.0;
   // SS_rm = this.Sc_Load_RM + ( this.Tc_Load_RM - this.Sc_Load_RM ) * ( 1.0 - sRc ) * this.saijohatsu / 100.0;
   // if( SS_rm > this.Tc_Load_RM ){
   //    SS_rm = this.Tc_Load_RM;
   // }
   // this.DBout = this.DBin - SS_rm / this.in_VaC_S;//乾球温度
   // this.XGout = Psychrometrics.FNXth( this.DBout, IAout);//絶対湿度
      /*
         if(saijohatsu>0.0 && Psychrometrics.FNRhtx(DBin, XGin)<40.0){
            System.out.println("40以下"+(++u40)+"S_Load_RM="+S_Load_RM+" T_Load=_RM"+T_Load_RM+" sRc="+sRc+"
SS_rm="+SS_rm);
            System.out.println("   DBout0="+(DBra-S_Load_RM/in_Va_S)+"  DBout1="+(DBra-SS_rm/in_Va_S));
            System.out.println("   XXout0="+Psychrometrics.FNXth((DBra-S_Load_RM/in_Va_S), IAout)+"
XXout1="+Psychrometrics.FNXth((DBra-SS_rm/in_Va_S), IAout));
            System.out.println("   DBin  ="+DBin+"  XGin  ="+XGin+"  RHin  ="+Psychrometrics.FNRhtx(DBin, XGin));
            System.out.println("   DBoa  ="+DBoa+"  XGoa  ="+XGoa+"  RHoa  ="+Psychrometrics.FNRhtx(DBoa, XGoa));
            System.out.println("   DBra  ="+DBra+"  XGra  ="+XGra+"  RHra  ="+Psychrometrics.FNRhtx(DBra, XGra));
         }
       */
      //*****************090725-e
      //    ********************************************************************
      //    System.out.println("DBout="+DBout+" "+XGout+"  ");
   }



   private void heating_supply(){
      double IAout;

      this.RHra = Psychrometrics.FNRhtx( this.DBra, this.XGra );
      this.Sh_Load_RM = this.Sh_Load;
      this.Th_Load_RM = this.Th_Load;

      if( this.in_HUM_on <= this.RHra || this.in_HUM_mx < 0.00001 ){//RAが設定相対湿度以上　あるいは　加湿能力が０の時
         IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
         IAout = IAin +this.Th_Load_RM / (this.in_VaH_S*this.vavRate) * 1000.0;
         this.DBout = this.DBin + this.Sh_Load_RM / (this.GW*1.006);//乾球温度
         //this.XGout = Psychrometrics.FNXth( this.DBout, IAout );//絶対湿度
         //加湿しない
         this.XGout = this.XGin;
         this.CW_Wrate = 0.0;
         //    System.out.println("1 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
         //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
      } else {
         IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
```

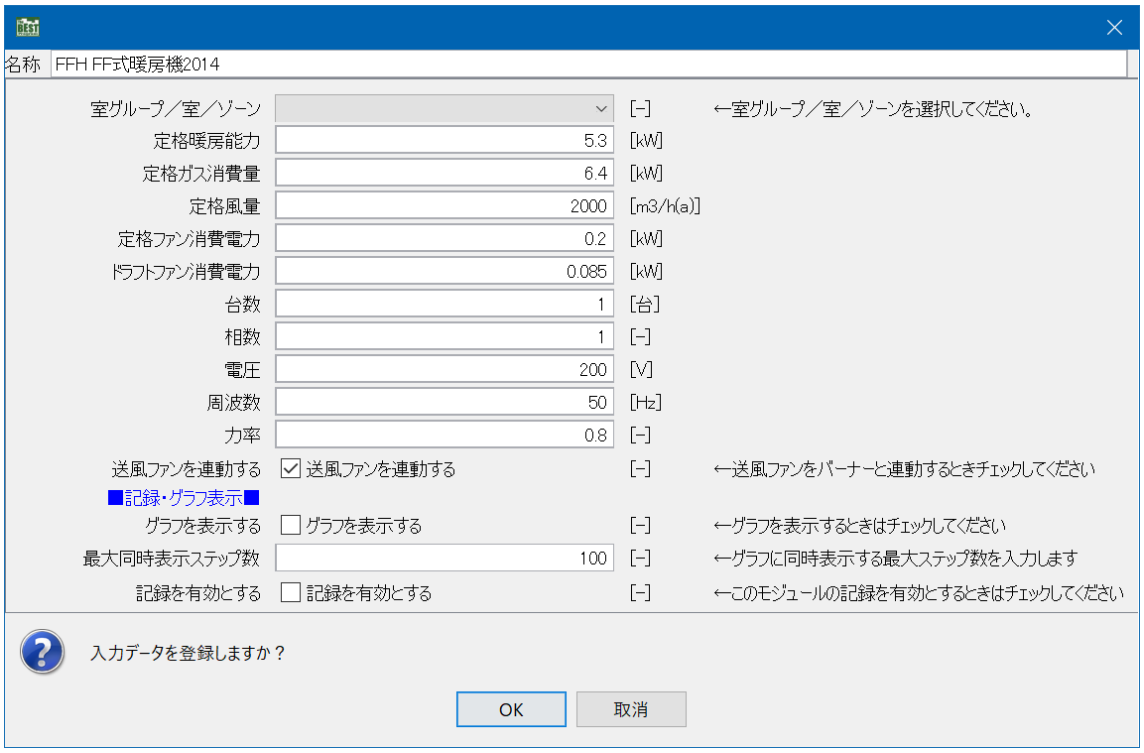
```
            IAout = IAin + this.Th_Load_RM / (this.in_VaH_S*this.vavRate)*1000.0;
            this.DBout = Psychrometrics.FNDbrh( this.in_HUM_rt, IAout);
            this.XGout = Psychrometrics.FNxtr( this.DBout, this.in_HUM_rt);
            this.CW_Wrate = ( this.XGout - this.XGin ) * this.in_VaH_S*this.vavRate;

            //定格加湿量をオーバーするときの処理
            if( this.CW_Wrate > this.in_HUM_mx ){
               this.XGout = this.in_HUM_mx / (this.in_VaH_S*this.vavRate)+ this.XGin;
               this.DBout =Psychrometrics.FNDbxh( this.XGout, IAout);
               this.CW_Wrate = this.in_HUM_mx;
            }
            if( this.XGout < this.XGin){
               IAin  = Psychrometrics.FNH( this.DBin, this.XGin );
               IAout = IAin + this.Th_Load_RM / (this.in_VaH_S*this.vavRate) * 1000.0;
               this.DBout = this.DBin + this.Sh_Load_RM / (this.GW*1.006);//乾球温度
               this.XGout = Psychrometrics.FNXth( this.DBout, IAout);//絶対湿度
               this.CW_Wrate = 0.0;
            }
            //    System.out.println("2 DBra="+DBra+" "+XGra+"  "+S_Load_RM+"  "+T_Load_RM);
            //    System.out.println("DBout="+DBout+" "+XGout+"  "+IAra+"  "+IAout);
         }
         //    this.watInCW.setFlowRate(CW_Wrate);
      }


      private void ventilation_supply(){

         this.Sh_Load_RM = 0.0;
         this.Th_Load_RM = 0.0;

         this.DBout = this.DBin;//乾球温度
         this.XGout = this.XGin;//絶対湿度
         this.CO2ppmout = this.CO2ppmin;//20131031
         this.CW_Wrate = 0.0;

      }

   }
```

「FFH FF 式暖房機 2014」（場所：設備 2015／個別分散 2015／）

モジュール名	FFH FF 式暖房機 2014
クラス	FuelFiredFurnaceModule2014

（1）入力画面

・スペック



（2）モジュールの概要

BESTEST 用にその仕様で開発した FuelFiredFurnace モジュールです

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	室グループ／室／ゾーン	String	zoneAirforSSinside	#RoomGroup	[-]	―	―		←室グループ／室／ゾーンを選択してください。
2	定格暖房能力	double	deQH	5.3	[kW]	―	0		
3	定格ガス消費量	double	dePG	6.4	[kW]	―	0		
4	定格風量	double	deFlowRate_SA	2000	[m3/h(a)]	―	1		
5	定格ファン消費電力	double	dePE_SA	0.2	[kW]	―	0		
6	ドラフトファン消費電力	double	dePE_DR	0.085	[kW]	―	0		
7	台数	double	NUM	1	[台]	―	0		
8	相数	int	phase	1	[-]	3	1		
9	電圧	double	voltage	200	[V]	―	0		
10	周波数	double	frequency	50	[Hz]	60	50		
11	力率	double	powerFactor	0.8	[-]	1	0		
12	送風ファンを連動する	boolean	isFanCycleWithBurner	TRUE	[-]	―	―		←送風ファンをバーナーと連動するときチェックしてください
13	■記録・グラフ表示■					―	―		
14	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
15	最大同時表示ステップ数	int	maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
16	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	外気入口	L0_airIn	airIn	-	状態	空気	入口	
5	排気出口	L0_airOut	airOut	-	状態	空気	出口	
6	排ガス出口	L0_airOutEX	airOutEX	-	状態	空気	出口	
7	操作量入口	L0_valInCtrl	valInCtrl	-	値	値	入口	
8	ガス入口	L0_gasIn	gasIn	-	状態	ガス	入口	
9	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	FFF_Message#-#-	メッセージ	—	メッセージ
2	FFF_入口空気温度#℃#温度	FFF_入口空気温度	℃	入口
3	FFF_入口空気絶対湿度#g/g'#絶対湿度	FFF_入口空気絶対湿度	g/g	入口
4	FFF_入口空気流量#g/s#質量流量	FFF_入口空気流量	g/s	入口
5	FFF_出口空気温度#℃#温度	FFF_出口空気温度	℃	出口
6	FFF_出口空気絶対湿度#g/g'#絶対湿度	FFF_出口空気絶対湿度	g/g	出口
7	FFF_出口空気流量#g/s#質量流量	FFF_出口空気流量	g/s	出口
8	FFF_排気温度#℃#温度	FFF_排気温度	℃	出口
9	FFF_排気流量#g/s#質量流量	FFF_排気流量	g/s	出口
10	FFF_PID操作量#-#-	FFF_PID操作量	—	入口
11	FFF_消費ガス#W#ガス 熱源	FFF_消費ガス	W	エネルギー消費
12	FFF_合計消費電力#W#電力 熱搬送 空調空気搬送	FFF_合計消費電力	W	エネルギー消費
13	FFF_循環ファン消費電力#W#電力 熱搬送 空調空気搬送	FFF_循環ファン消費電力	W	My
14	FFF_ドラフトファン消費電力#W#電力 熱源	FFF_ドラフトファン消費電力	W	My
15	FFF_処理熱量#W#処理熱量	FFF_処理熱量	W	負荷

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestGas;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.dk.test.TestCommand;
import java.util.Map;


/**
 * @author HIROSHI NINOMIYA 2014/11/11
 * FuelFiredFurnaceモジュールです
 * BESTEST仕様に対応で開発
 */
public class FuelFiredFurnaceModule2014 extends AbstractBestModule implements
      IBestMetaModule {

      private final String moduleName="(FuelFiredFurnaceModule2014)";
      private final String S_NODE_airIn    = "L0_airIn";    //入口空気
      private final String S_NODE_airOut   = "L0_airOut";      //出口空気
      private final String S_NODE_airOutEX = "L0_airOutEX";    //排気
      private final String S_NODE_eleIn    = "L0_eleIn";
      private final String S_NODE_gasIn    = "L0_gasIn";

      private final String S_NODE_valInCTRL = "L0_valInCtrl";//PID操作量

      private final String C_NODE_swcIn = "L1_swcIn";       //運転状態  on/off
      private final String C_NODE_modIn = "L1_modIn";       //運転モード 停止/冷房/暖房

      private final String R_NODE = "L2_recOut";            //記録

      private final String RECORD_message    = "FFF_Message#-#-" ;
      private final String RECORD_T_airIn    = "FFF_入口空気温度#℃#温度" ;
      private final String RECORD_X_airIn    = "FFF_入口空気絶対湿度#g/g'#絶対湿度" ;
      private final String RECORD_M_airIn    = "FFF_入口空気流量#g/s#質量流量" ;
      private final String RECORD_T_airOut   = "FFF_出口空気温度#℃#温度" ;
      private final String RECORD_X_airOut   = "FFF_出口空気絶対湿度#g/g'#絶対湿度" ;
      private final String RECORD_M_airOut   = "FFF_出口空気流量#g/s#質量流量" ;
      private final String RECORD_T_airOutEX = "FFF_排気温度#℃#温度" ;
      private final String RECORD_M_airOutEX = "FFF_排気流量#g/s#質量流量" ;
      private final String RECORD_V_valInCTRL= "FFF_PID操作量#-#-";
      private final String RECORD_PG         = "FFF_消費ガス#W#ガス 熱源";
      private final String RECORD_PE         = "FFF_合計消費電力#W#電力 熱搬送 空調空気搬送";
      private final String RECORD_PE_SAFan   = "FFF_循環ファン消費電力#W#電力 熱搬送 空調空気搬送";
      private final String RECORD_PE_DRFan   = "FFF_ドラフトファン消費電力#W#電力 熱源";
      private final String RECORD_Q          = "FFF_処理熱量#W#処理熱量";

//    外部定義項目
      private final String SPEC_name           = "名称";

      private final String SPEC_grzName  = "室グループ／室／ゾーン";

      private final String SPEC_deQH           = "定格暖房能力[W]" ;//"加熱能力[W]" ;
      private final String SPEC_deFlowRate_SA = "定格風量[g/s]";//"循環送風量[g/s]";
      private final String SPEC_dePG           = "定格暖房入力(ガス)[W]";//"消費ガス[W]" ;
      private final String SPEC_dePE_SA        = "定格ファン消費電力[W]" ;//"循環ファン消費電力[W]"
      private final String SPEC_dePE_DR        = "ドラフトファン消費電力[W]" ;
      private final String SPEC_NUM            = "台数[-]";
      private final String SPEC_Phase          = "相数[-]";
```

```
	private final String SPEC_Voltage        = "電圧[V]";
	private final String SPEC_Frequency      = "周波数[Hz]";
	private final String SPEC_PowerFactor    = "力率[-]";
	private final String SPEC_isFanCycleWithBurner = "送風ファンを連動する[-]";
	//
	private final String SPEC_isGVisible = "グラフを表示する";
	private final String SPEC_maxItemCount = "最大同時表示ステップ数";
	private final String SPEC_isRecord   = "記録を有効とする";

	private StringBuffer message= new StringBuffer();

	private BestAir   airIn    = null;
	private BestAir   airOut   = null;
	private BestAir   airOutEX = null;
	private BestValue valInCTRL= null;
	private BestGas   gasIn    = null;
	private BestElectricity eleIn = null;

	//仕様
	private String name=null;//名称
	private double deQH;//        = "加熱能力[W]" ;
	private double deFlowRate_SA;// = "循環送風量[g/s]";
	private double dePG;//        = "消費ガス[W]" ;
	private double dePE_SA;//     = "循環ファン消費電力[W]" ;
	private double dePE_DR;//     = "ドラフトファン消費電力[W]" ;
	private double PG;            //消費gas(W)
	private double PE;            //消費電力(W)
	private double pe_SA;            //消費電力(W)
	private double pe_DR;            //消費電力(W)

	private double NUM; //台数
	private int phase;               //相数[-]
	private double voltage;       //電圧[V]
	private double frequency;        //周波数[Hz]
	private double powerFactor;      //力率[-]
	private boolean isFanCycleWithBurner = false;// true="送風ファンを連動する[-]"
	//
	private boolean isGVisible = false;//このグラフを表示する=true
	private int  maxItemCount = 100;//最大同時表示ステップ数
	private boolean isRecord   = false;//記録を有効とする=true

//入力値
	private int modIn;
	private int swcIn;
	private int mState;  //停止、冷房、暖房フラグ
	private int kState;  //収束フラグ

//計算使用
	private double q = 0;//処理熱量[W]
	private double cop = 0;//

	//グラフ表示など
	private GraphCommonJFrame2015 gcFFFH = null;
	private GraphCommonData2015[] gcData = null;

	private ZoneAirforSSinsideModule201502 zoneAirforSSinside = null;
	private String grzName = null;
	private boolean isUseZoneAir = false;


   @Override
//外部定義項目取得
   public void setProfile(BestSpecs spec){

	if(spec == null) {
	   return;
	}
	Map<String,String> map=spec.getSpec();
	if(map == null) {
	   return;
	}
```

```
    //名称
    this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

    this.grzName = spec.getSpecValue( this.SPEC_grzName, "" );
    if( !this.grzName.equals( "" )){
       this.isUseZoneAir = true;
    }

    if( this.isUseZoneAir ){
       this.zoneAirforSSinside = new ZoneAirforSSinsideModule201502();
       this.zoneAirforSSinside._setProfile( this.name, this.grzName );
    }

    //private final String SPEC_deQH          = "加熱能力[W]" ;
    this.deQH = spec.getSpecValue( this.SPEC_deQH, 0. );

    //private final String SPEC_deFlowRate_SA = "循環送風量[g/s]";
    this.deFlowRate_SA = spec.getSpecValue( this.SPEC_deFlowRate_SA, 0. );

    //private final String SPEC_dePG          = "消費ガス[W]" ;
    this.dePG = spec.getSpecValue( this.SPEC_dePG, 0. );

    //

    //消費電力  SPEC_dePE_SA      = "循環ファン消費電力[W]" ;
    this.dePE_SA = spec.getSpecValue( this.SPEC_dePE_SA, 0. );

    //消費電力  SPEC_dePE         = "ドラフトファン消費電力[W]" ;
    this.dePE_DR = spec.getSpecValue( this.SPEC_dePE_DR, 0. );

    //台数
    this.NUM = spec.getSpecValue( this.SPEC_NUM, 1. );

    this.deQH *= this.NUM;
    this.deFlowRate_SA *= this.NUM;
    this.dePG *= this.NUM;
    this.dePE_SA *= this.NUM;
    this.dePE_DR *= this.NUM;

    //相数
    this.phase = spec.getSpecValue( this.SPEC_Phase, 1 );

    //電圧
    this.voltage = spec.getSpecValue( this.SPEC_Voltage, 200. );

    //周波数
    this.frequency = spec.getSpecValue( this.SPEC_Frequency, 50. );

    //力率
    this.powerFactor = spec.getSpecValue( this.SPEC_PowerFactor, 0.8 );

    //SPEC_isFanCycleWithBurner = "送風ファンを連動する[-]"
    this.isFanCycleWithBurner = spec.getSpecValue( this.SPEC_isFanCycleWithBurner, true );

    //isGVisibleを取得
    this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

    // 最大同時表示ステップ数
    this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

    //isRecordを取得
    this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

}

public void initialize(IBestStateMessage stateNodes,
      IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
      IBestDomainSchedule schedule) {
    //状態ノードを取得
    super.sm = stateNodes;

    //制御ノードを取得
```

```
        super.cm = commandNodes;

        //記録ノードを取得
        super.rm = recordNodes;

        //接続ノード　出口

        //airIn nino
        this.airIn = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airIn );
        this.airIn.setMaxFlowRate( this.deFlowRate_SA );//20150423

        //valInCTRL
        this.valInCTRL = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInCTRL );

        //airOut
        this.airOut = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOut );
        this.airOut.setMaxFlowRate( this.deFlowRate_SA );//20150423
        if( this.isUseZoneAir ){
            this.zoneAirforSSinside.checkNumberAirChange( this.deFlowRate_SA, "deFlowRate_SA", this.moduleName,
"initialize()", this.name, this.isRecord, this.message);
            this.zoneAirforSSinside._initialize( );
        }

        //airOutEX
        this.airOutEX = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOutEX );
        this.airOutEX.setMaxFlowRate( 0. );//20150423

        //gasIn
        this.gasIn = BestGas.bindnode( super.transferMapState, super.sm, this.S_NODE_gasIn );

        //eleIn
        this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

        this.eleIn.setPhase( this.phase );
        this.eleIn.setVoltage( this.voltage );
        this.eleIn.setFrequency( this.frequency );


        //グラフ表示の準備
        if( this.isGVisible ){
            this.gcData = new GraphCommonData2015[11];

            this.gcData[0] = new GraphCommonData2015( "T_airIn[℃]", this.maxItemCount, -20, 50, 0 );
            this.gcData[1] = new GraphCommonData2015( "T_airOut[℃]", this.maxItemCount, -20, 50, 0 );
            this.gcData[2] = new GraphCommonData2015( "T_airOutEX[℃]", this.maxItemCount, -20, 50, 0 );
            this.gcData[3] = new GraphCommonData2015( "M_airOut[g/s]", this.maxItemCount, 0, 10000, 1 );
            this.gcData[4] = new GraphCommonData2015( "Q_heat[W]", this.maxItemCount, 0, 100000, 2 );
            this.gcData[5] = new GraphCommonData2015( "W_gasIn[W]", this.maxItemCount, 0, 100000, 2 );
            this.gcData[6] = new GraphCommonData2015( "W_eleIn[W]", this.maxItemCount, 0, 100000, 2 );
            this.gcData[7] = new GraphCommonData2015( "W_eleFanC[W]", this.maxItemCount, 0, 100000, 2 );
            this.gcData[8] = new GraphCommonData2015( "W_eleFanD[W]", this.maxItemCount, 0, 100000, 2 );
            this.gcData[9] = new GraphCommonData2015( "COPgas[-]", this.maxItemCount, 0, 2, 3 );
            this.gcData[10] = new GraphCommonData2015( "QRate[-]", this.maxItemCount, 0, 2, 3 );

            String[] yName = { "温度[℃]", "風量[g/s]", "熱量,電力[W]", "COP・負荷率[-]"};

            gcFFFH = new GraphCommonJFrame2015( this.name, this.gcData, yName );
        }
    }

    private int calc_Stop(){
        this.PE = 0;
        this.airOut.setFlowRate( 0 );
        this.airOutEX.setFlowRate( 0 );
        this.q = 0;
        this.cop = 0;

        return 0;
    }

    public void outputs() {
        if(super.sm == null || super.cm == null ){
```

```
            return;
        }

        this.PG = 0;
        this.PE = 0;
        this.pe_SA = 0;
        this.pe_DR = 0;

        //ノード接続毎回読込み
        this.airIn = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airIn));
        if( this.isUseZoneAir ){
            this.zoneAirforSSinside._outputSetRA( this.airIn );
        }

        this.swcIn=super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcIn) );
        this.modIn=super.cm.getCommand(super.getConnectionNode(this.C_NODE_modIn));

        this.airOut.copyAllvalState( this.airIn );//20131029

        if( this.isRecord ){
            this.message.append( "(C)swcIn="+this.swcIn+"/modeIn="+this.modIn );
        }

        if( Airswc.isOFF( this.swcIn )){
            this.mState = Airswc.OFF;

        }else if( Airmod.isHEAT( this.modIn )){
            this.mState = Airmod.HEAT;

        }else{
            this.mState = Airswc.OFF;
        }

        switch(mState){
        case Airswc.OFF:
            //停止
            if( this.isRecord ){
                message.append("(C)停止");
            }

            kState = calc_Stop();

            break;

        case Airmod.HEAT:
            //暖房
            if( this.isRecord ){
                message.append("(C)暖房運転");
            }
            BestAir.checkOpeData( this.airIn, "airIn", this.moduleName, "outputs()", this.name, this.isRecord,
this.message );

            double var;//負荷率
            var = this.valInCTRL.getValue();
            //ドラフトファン消費電力    圧縮機と連動
            this.pe_DR = this.dePE_DR * var;

            //処理負荷[W]
            this.q = this.deQH * var;

            //消費ガス
            this.PG = this.dePG * ( 0.0080472574
                  + 0.87564457 * var
                  + 0.29249943 * var * var
                  - 0.17624156 * var * var * var);

            if( this.isFanCycleWithBurner ){
            }else{
                var = 1;
            }
            this.pe_SA = this.dePE_SA * var;
            this.PE = this.pe_DR + this.pe_SA;
```

```
            //吹き出し状態
            if( var == 0 ){
               this.airOut.setTempDB( this.airIn.getTempDB() );
            }else{
               this.airOut.setTempDB( this.airIn.getTempDB() + ( this.q + this.pe_SA ) / this.deFlowRate_SA / var / 1.006 );
            }
            this.airOut.setFlowRate( this.deFlowRate_SA * var );

         // System.out.println(" airIn_T="+this.airIn.getTempDB()+" airOut.t="+airOut.getTempDB() +"  dT="+(( this.q +
this.dePE_SA * var ) / this.deFlowRate_SA / 1.006) + " q="+this.q+ " FlowRate="+this.deFlowRate_SA);

            //COP
            if( this.PE == 0 ){
               this.cop = 0;
            }else{
               this.cop = Math.abs( this.q / this.PE );
            }

            break;

         default:
            System.out.println(this.moduleName + ">>Error<< onOff*modeが範囲外");
         }

         this.gasIn.setWatt( this.PG );

         this.eleIn.setActivePower( this.PE );
         this.eleIn.setReactivePower( this.PE * Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 ) );

         //ノードに設定
         //出力側ノード
         super.sm.setState( super.getConnectionNode( this.S_NODE_airOut ), this.airOut );
         super.sm.setState( super.getConnectionNode( this.S_NODE_airOutEX ), this.airOutEX );
         super.sm.setState( super.getConnectionNode( this.S_NODE_gasIn ), this.gasIn );
         super.sm.setState( super.getConnectionNode( this.S_NODE_eleIn ), this.eleIn );

         //記録ノードに設定
         //グラフデータ追加
         if( this.isGVisible ){

            double[] gData = new double[11];

            gData[0] = this.airIn.getTempDB();
            gData[1] = this.airOut.getTempDB();
            gData[2] = this.airOutEX.getTempDB();
            gData[3] = this.airOut.getFlowRate();
            gData[4] = this.q;
            gData[5] = this.gasIn.getWatt();
            gData[6] = this.eleIn.getActivePower();
            gData[7] = this.pe_SA;
            gData[8] = this.pe_DR;
            gData[9] = this.cop;
            gData[10] = this.valInCTRL.getValue();

            this.gcFFFH.addData( BestTimeManager.getDateWeatherTime(), gData );
         }

         //記録ノード
         if( this.isRecord  && super.rm != null ){
            this.record();
         }

         message.setLength(0);
      }


      /**
       * 記録
       */
      private void record(){
         if( CheckPrintModule.isPrintMessage ){
```

```
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_message, this.name, message.toString());
        }

        if( CheckPrintModule.isPrintLoad ){
            //Q
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_Q, this.name, AirFormat.df_2(this.q) );
        }

        if( CheckPrintModule.isPrintEnergy ){
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_PG, this.name, AirFormat.df_2( this.gasIn.getWatt()));
            //PE
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_PE, this.name, AirFormat.df_2( this.eleIn.getActivePower()));
        }

        if( CheckPrintModule.isPrintStateIn ){
            //入口空気
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_T_airIn, this.name, AirFormat.df_2( this.airIn.getTempDB() ));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_X_airIn, this.name, AirFormat.df_5( this.airIn.getHumi()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_M_airIn, this.name, AirFormat.df_2( this.airIn.getFlowRate()));
            //入口
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_V_valInCTRL, this.name, AirFormat.df_2( this.valInCTRL.getValue()));
        }

        if( CheckPrintModule.isPrintStateOut ){
            //出口空気
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_T_airOut, this.name, AirFormat.df_2( this.airOut.getTempDB()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_X_airOut, this.name, AirFormat.df_5( this.airOut.getHumi()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_M_airOut, this.name, AirFormat.df_2( this.airOut.getFlowRate()));
            //出口冷温水
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_T_airOutEX, this.name, AirFormat.df_2( this.airOutEX.getTempDB()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_M_airOutEX, this.name, AirFormat.df_2( this.airOutEX.getFlowRate()));
        }

        if( CheckPrintModule.isPrintStateMy ){
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_PE_SAFan, this.name, AirFormat.df_2( this.pe_SA ));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_PE_DRFan, this.name, AirFormat.df_2( this.pe_DR ));

        }

    }


    public void update() {
        if( this.isUseZoneAir ){
            this.zoneAirforSSinside._update( this.airOut);
        }

    }


    public Object viewInternal(TestCommand cmd){
        java.util.List<Object> result = new java.util.ArrayList<Object>();

        result.add(super.sm);
        result.add(super.cm);
        result.add(super.rm);
        //外部定義項目
```

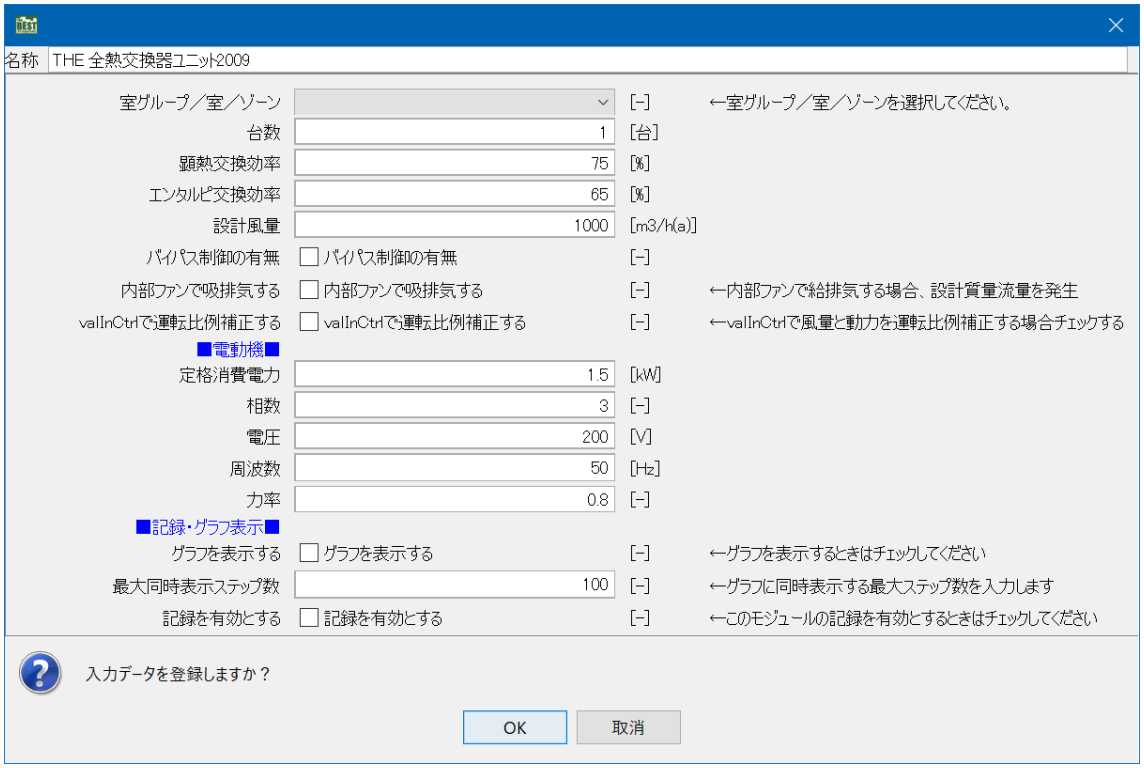
```
        result.add(this.PE);
        result.add(this.NUM);
        result.add(this.phase);
        result.add(this.voltage);
        result.add(this.frequency);
        result.add(this.powerFactor);
        result.add(this.kState);


        if(airIn != null && airOut != null ){
           result.add(this.airIn.getTempDB());       //入口温度
           result.add(this.airIn.getHumi());         //入口湿度
           result.add(this.airIn.getFlowRate());
           result.add(this.airOut.getTempDB());      //出口温度
           result.add(this.airOut.getHumi());        //出口湿度
           result.add(this.airOut.getFlowRate());
        }else{
           result.add(new Double(0.0));
           result.add(new Double(0.0));
           result.add(new Double(0.0));
           result.add(new Double(0.0));
           result.add(new Double(0.0));
           result.add(new Double(0.0));
        }

        return result;
    }

}
```

「THE 全熱交換器ユニット 2009」（場所：設備 2015／個別分散 2015／）

モジュール名	THE 全熱交換器ユニット 2009
クラス	TotalHeatExchangerSimpleModule20090505

（1）入力画面

・スペック



（2）モジュールの概要

全熱交換器ユニットモジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

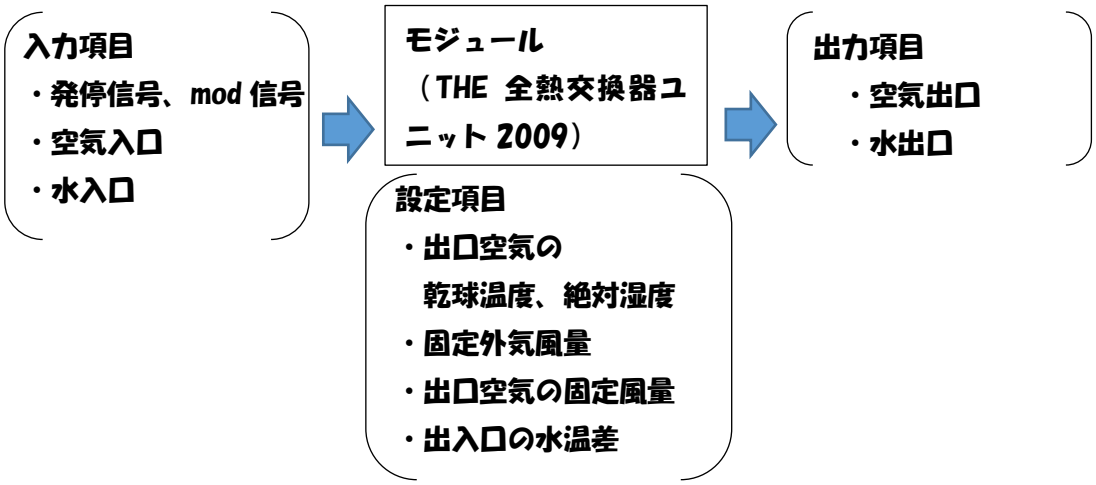


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	室グループ／室／ゾーン	String	zoneAirforSSinside	#RoomGroup	[-]	―	―		←室グループ／室／ゾーンを選択します
2	台数	double	dai	1	[台]	―	0		
3	顕熱交換効率	double	deffiS	75	[%]	100	0		
4	エンタルピ交換効率	double	deffiE	65	[%]	100	0		
5	設計風量	double	defmFlowRate	1000	[m3/h(a)]	―	0		
6	バイパス制御の有無	boolean	isBypass	FALSE	[-]	―	―		
7	内部ファンで吸排気する	boolean	isOpeMyFan	FALSE	[-]	―	―		←内部ファンで給排気する場合、設計質量流量を発生
8	valInCtrl で運転比例補正	boolean	isOpeValInCtrl	FALSE	[-]	―	―		←運転風量を valInCtrl の値で比例補正する場合チェックします
9	■電動機■					―	―		
10	定格消費電力	double	dPEini	1.5	[kW]	―	0		
11	相数	int	phase	3	[-]	3	1		
12	電圧	double	voltage	200	[V]	―	0		
13	周波数	double	frequency	50	[Hz]	60	50		
14	力率	double	powerFactor	0.8	[-]	1	0		
15	■記録・グラフ表示■					―	―		
16	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックします
17	最大同時表示ステップ数	int	maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
18	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録をするときはチェックします

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	外気導入信号入口	L1_swcInOA	swcInOA	-	制御	OnOff 信号	入口	
5	空気入口 0	L0_airIn0	airIn0	-	状態	空気	入口	
6	空気入口 1	L0_airIn1	airIn1	-	状態	空気	入口	
7	空気出口 0	L0_airOut0	airOut0	-	状態	空気	出口	
8	空気出口 1	L0_airOut1	airOut1	-	状態	空気	出口	
9	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	
10	操作量入口	L0_valInCtrl	valInCtrl	-	値	制御	入口	運転風量と動力をこの値で比例補正するときに使用する

（6）記録項目

No.	項目	出力内容	単位	分類
1	全熱交換器_Message#-#-	メッセージ	—	メッセージ
2	全熱交換器_空気0入口温度#℃#温度	全熱交換器_空気0入口温度#℃#温度	℃	入口
3	全熱交換器_空気0入口絶対湿度#g/g'#絶対湿度	全熱交換器_空気0入口絶対湿度#g/g'#絶対湿度	g/g	入口
4	全熱交換器_空気0入口風量#g/s#質量流量	全熱交換器_空気0入口風量#g/s#質量流量	g/s	入口
5	全熱交換器_空気1入口温度#℃#温度	全熱交換器_空気1入口温度#℃#温度	℃	入口
6	全熱交換器_空気1入口絶対湿度#g/g'#絶対湿度	全熱交換器_空気1入口絶対湿度#g/g'#絶対湿度	g/g	入口
7	全熱交換器_空気1入口風量#g/s#質量流量	全熱交換器_空気1入口風量#g/s#質量流量	g/s	入口
8	全熱交換器_空気0出口温度#℃#温度	全熱交換器_空気0出口温度#℃#温度	℃	出口
9	全熱交換器_空気0出口絶対湿度#g/g'#絶対湿度	全熱交換器_空気0出口絶対湿度#g/g'#絶対湿度	g/g	出口
10	全熱交換器_空気0出口風量#g/s#質量流量	全熱交換器_空気0出口風量#g/s#質量流量	g/s	出口
11	全熱交換器_空気1出口温度#℃#温度	全熱交換器_空気1出口温度#℃#温度	℃	出口
12	全熱交換器_空気1出口絶対湿度#g/g'#絶対湿度	全熱交換器_空気1出口絶対湿度#g/g'#絶対湿度	g/g	出口
13	全熱交換器_空気1出口風量#g/s#質量流量	全熱交換器_空気1出口風量#g/s#質量流量	g/s	出口
14	全熱交換器_消費電力#W#電力 熱搬送 空調空気搬送	全熱交換器_消費電力#W#電力 熱搬送 空調空気搬送	W	エネルギー消費
15	全熱交換器_空気0処理顕熱量#W#熱量	全熱交換器_空気0処理顕熱量#W#熱量	W	負荷
16	全熱交換器_空気0処理潜熱量#W#熱量	全熱交換器_空気0処理潜熱量#W#熱量	W	負荷
17	全熱交換器_空気0処理全熱量#W#熱量	全熱交換器_空気0処理全熱量#W#熱量	W	My
18	全熱交換器_空気0/定格風量比#-#-	全熱交換器_空気0/定格風量比#-#-	—	My
19	全熱交換器_空気1/定格風量比#-#-	全熱交換器_空気1/定格風量比#-#-	—	My
20	全熱交換器_空気0/1風量比#-#-	全熱交換器_空気0/1風量比#-#-	—	My
21	全熱交換器_空気1/0風量比#-#-	全熱交換器_空気1/0風量比#-#-	—	My
22	全熱交換器_顕熱交換効率#-#-	全熱交換器_顕熱交換効率#-#-	—	My
23	全熱交換器_潜熱交換効率#-#-	全熱交換器_潜熱交換効率#-#-	—	My
24	全熱交換器_全熱交換効率#-#-	全熱交換器_全熱交換効率#-#-	—	My

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.essentials.utility.Psychrometrics;

/**
 * @author HIROSHI NINOMIYA /20090505
 * 全熱交換機です
 *
 * 20101206nino/消費電力の出力と記録を追加
 * 20131122 台数追加
 * 20170926
 */
public class TotalHeatExchangerSimpleModule20090505 extends AbstractBestModule implements
      IBestMetaModule {

   private final String moduleName = "(TotalHeatExchangerSimpleModule20090505) ";

   //接続ノード
   //出入り口
   private final String S_NODE_airIn0  = "L0_airIn0";
   private final String S_NODE_airIn1  = "L0_airIn1";
   private final String S_NODE_airOut0 = "L0_airOut0";
   private final String S_NODE_airOut1 = "L0_airOut1";
   private final String S_NODE_eleIn   = "L0_eleIn";

   //20190903住宅版のスケジュール連動への対応
   private final String S_NODE_valInCtrl  = "L0_valInCtrl"; //ファン運転能力比

   //制御ノード
   private final String C_NODE_swcIn  = "L1_swcIn";   //運転状態：on/off
   private final String C_NODE_swcInOA= "L1_swcInOA"; //20150410外気　on/off
   private final String C_NODE_modIn  = "L1_modIn";   //空調モード：停止/冷房/暖房

   //記録ノード
   private final String R_NODE = "L2_recOut";

   private final String RECORD_message = "全熱交換器_Message#-#-";
   private final String RECORD_T_aIn0  = "全熱交換器_空気0入口温度#℃#温度";
   private final String RECORD_X_aIn0  = "全熱交換器_空気0入口絶対湿度#g/g'#絶対湿度";
   private final String RECORD_M_aIn0  = "全熱交換器_空気0入口風量#g/s#質量流量";
   private final String RECORD_T_aIn1  = "全熱交換器_空気1入口温度#℃#温度";
   private final String RECORD_X_aIn1  = "全熱交換器_空気1入口絶対湿度#g/g'#絶対湿度";
   private final String RECORD_M_aIn1  = "全熱交換器_空気1入口風量#g/s#質量流量";
   private final String RECORD_T_aOut0 = "全熱交換器_空気0出口温度#℃#温度";
   private final String RECORD_X_aOut0 = "全熱交換器_空気0出口絶対湿度#g/g'#絶対湿度";
   private final String RECORD_M_aOut0 = "全熱交換器_空気0出口風量#g/s#質量流量";
   private final String RECORD_T_aOut1 = "全熱交換器_空気1出口温度#℃#温度";
   private final String RECORD_X_aOut1 = "全熱交換器_空気1出口絶対湿度#g/g'#絶対湿度";
   private final String RECORD_M_aOut1 = "全熱交換器_空気1出口風量#g/s#質量流量";

   private final String RECORD_PE      = "全熱交換器_消費電力#W#電力 熱搬送 空調空気搬送";

   private final String RECORD_qS_air  = "全熱交換器_空気0処理顕熱量#W#熱量";//加熱が正値
```

```
private final String RECORD_qL_air  = "全熱交換器_空気0処理潜熱量#W#熱量";//加湿が正値
private final String RECORD_qT_air  = "全熱交換器_空気0処理全熱量#W#熱量";

private final String RECORD_mRatio_airIn0defm  = "全熱交換器_空気0/定格風量比#-#-";
private final String RECORD_mRatio_airIn1defm  = "全熱交換器_空気1/定格風量比#-#-";
private final String RECORD_mRatio_airIn01  = "全熱交換器_空気0/1風量比#-#-";
private final String RECORD_mRatio_airIn10  = "全熱交換器_空気1/0風量比#-#-";

private final String RECORD_qSRatio  = "全熱交換器_顕熱交換効率#-#-";
private final String RECORD_qLRatio  = "全熱交換器_潜熱交換効率#-#-";
private final String RECORD_qTRatio  = "全熱交換器_全熱交換効率#-#-";

//仕様
private final String SPEC_name = "名称";

private final String SPEC_grzName  = "室グループ／室／ゾーン";
//
private final String SPEC_deffiS       = "顕熱交換効率[-]";
private final String SPEC_deffiE       = "エンタルピ交換効率[-]";
private final String SPEC_defmFlowRate = "設計質量流量[g/s]";
private final String SPEC_isBypass     = "バイパス制御の有無[-]";
//
private final String SPEC_isOpeMyFan   = "内部ファンで給排気する[-]";//20121002

private final String SPEC_isOpeValInCtrl = "valInCtrlで運転比例補正";
//
private final String SPEC_dai         = "台数[-]";
//
private final String SPEC_dPEini       = "定格消費電力[W]";
private final String SPEC_Phase        = "相数[-]";
private final String SPEC_Voltage      = "電圧[V]";
private final String SPEC_Frequency    = "周波数[Hz]";
private final String SPEC_PowerFactor  = "力率[-]";
//
private final String SPEC_isGVisible   = "グラフを表示する";//このグラフを表示する
private final String SPEC_maxItemCount = "最大同時表示ステップ数";//最大同時表示ステップ数
private final String SPEC_isRecord     = "記録を有効とする";//このモジュールの記録を有効とする

//仕様など
private String name = null;   //名称
private StringBuffer message = new StringBuffer();
//
private double deffiS;     //顕熱交換効率[-]
private double deffiE;     //エンタルピ交換効率[-]
private double defmFlowRate;  //設計質量流量[g/s]
private boolean isBypass;//バイパス制御の有無[-]
//
private boolean isOpeMyFan = false;//内部ファンで吸排気する

protected boolean isOpeValInCtrl = false;

private double dai         ;//= "台数";

private double dPEini;     //定格消費電力[W]
private int phase;         //相数[-]
private double voltage;    //電圧[V]
private double frequency;     //周波数[Hz]
private double powerFactor;   //力率[-]
//
private boolean isGVisible = false;//このグラフを表示する=true
private int  maxItemCount = 100;//最大同時表示ステップ数
private boolean isRecord   = false;//記録を有効とする=true

//接続熱媒など
private BestAir airIn0 = null;
private BestAir airIn1 = null;
private BestAir airOut0 = null;
private BestAir airOut1 = null;
private BestElectricity eleIn = null;

private BestValue valInCtrl;
```

```
//制御信号など
private int modIn;            //空調モード
private int modMy;
private int swcIn;            //運転状態
private int swcInOA;//20150410

//
private double m_airIn0;//質量流量[g/s]
private double m_airIn1;//質量流量[g/s]
private double t_airIn0;//温度[℃]
private double t_airIn1;//温度[℃]
//private double x_airIn0;//絶対湿度[g/g]
//private double x_airIn1;//絶対湿度[g/g]
private double e_airIn0;//エンタルピ[J/g]
private double e_airIn1;//エンタルピ[J/g]
//private double m_airOut0;//質量流量[g/s]
//private double m_airOut1;//質量流量[g/s]
private double t_airOut0;//温度[℃]
private double t_airOut1;//温度[℃]
private double x_airOut0;//絶対湿度[g/g]
private double x_airOut1;//絶対湿度[g/g]
private double e_airOut0;//エンタルピ[J/g]
private double e_airOut1;//エンタルピ[J/g]
private double f_PE;//電力消費量[W]

private double qS_air;//処理顕熱量[W]
private double qL_air;//処理潜熱量[W]
private double qT_air;//処理全熱量[W]

private double qSRatio;//  = "全熱交換器_顕熱交換効率#-#-";
private double qLRatio;//  = "全熱交換器_潜熱交換効率#-#-";
private double qTRatio;//  = "全熱交換器_全熱交換効率#-#-";
private final double CP = 1.006;//空気比熱[J/gK]

//グラフ表示など
private GraphJFrameTotalHeatExchanger20090505 gOAChamber = null;
private GraphCommonJFrame2015 gGCJF = null;
private GraphCommonData2015[] gData = null;

private ZoneAirforSSinsideModule201502 zoneAirforSSinside = null;
private String grzName = null;
private boolean isUseZoneAir = false;


@Override
public void setProfile(BestSpecs spec) {

   if( spec == null ){
      return;
   }

   Map<String, String> map = spec.getSpec();
   if( map == null ){
      return;
   }

   //名称を取得
   this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

   this.grzName = spec.getSpecValue( this.SPEC_grzName, "" );
   if( !this.grzName.equals( "" )){
      this.isUseZoneAir = true;
   }

   if( this.isUseZoneAir ){
      this.zoneAirforSSinside = new ZoneAirforSSinsideModule201502();
      this.zoneAirforSSinside._setProfile( this.name, this.grzName );
   }

   // 台数
   this.dai = spec.getSpecValue( this.SPEC_dai, 1. );
```

```
        //顕熱交換効率[-]を取得
        this.deffiS = spec.getSpecValue( this.SPEC_deffiS, 0.75 );

        //エンタルピ交換効率[-]を取得
        this.deffiE = spec.getSpecValue( this.SPEC_deffiE, 0.75 );

        //設計質量流量[g/s]を取得
        this.defmFlowRate = spec.getSpecValue( this.SPEC_defmFlowRate, 0. ) * this.dai;

        //バイパス制御の有無[-]を取得
        this.isBypass = spec.getSpecValue( this.SPEC_isBypass, false );
        //
        this.isOpeValInCtrl = spec.getSpecValue( this.SPEC_isOpeValInCtrl, false );

        //定格消費電力を取得
        this.dPEini = spec.getSpecValue( this.SPEC_dPEini, 0. ) * this.dai;

        //相数を取得
        this.phase = spec.getSpecValue( this.SPEC_Phase, 3 );

        //電圧を取得
        this.voltage = spec.getSpecValue( this.SPEC_Voltage, 200. );

        //周波数を取得
        this.frequency = spec.getSpecValue( this.SPEC_Frequency, 50. );

        //力率を取得
        this.powerFactor = spec.getSpecValue( this.SPEC_PowerFactor, 0.8 );

        //isOpeMyFanを取得
        this.isOpeMyFan = spec.getSpecValue( this.SPEC_isOpeMyFan, false );

        //isGVisibleを取得
        this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

        // 最大同時表示ステップ数
        this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

    }

    public void initialize(IBestStateMessage stateNodes,
          IBestControlMessage commandNodes,
          IBestRecordMessage recordNodes,
          IBestDomainSchedule schedule) {
        //状態ノードを取得
        super.sm = stateNodes;

        //制御ノードを取得
        super.cm = commandNodes;

        //記録ノードを取得
        super.rm = recordNodes;

        //接続ノード出口
        //airOut0, airOut1
        this.airOut0
        = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOut0 );
        this.airOut1
        = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOut1 );
        this.airOut0.setMaxFlowRate( this.defmFlowRate );//20150423
        this.airOut1.setMaxFlowRate( this.defmFlowRate );//20150423

        if( this.isUseZoneAir ){
            this.zoneAirforSSinside.checkNumberAirChange( this.defmFlowRate, "defmFlowRate", this.moduleName,
"initialize()", this.name, this.isRecord, this.message);
            this.zoneAirforSSinside._initialize();
        }

        //接続ノード入口チェック
```

```
    //airIn0, airIn1
    this.airIn0
    = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airIn0 );
    this.airIn1
    = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airIn1 );
    this.airIn0.setMaxFlowRate( this.defmFlowRate );//20150423
    this.airIn1.setMaxFlowRate( this.defmFlowRate );//20150423

    //20190903住宅版のスケジュール連動への対応
    this.valInCtrl = BestValue.bindnode( super.transferMapState,  super.sm, this.S_NODE_valInCtrl);

    //eleIn
    this.eleIn
    = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

    //グラフ表示の準備
    if( this.isGVisible ){
       gOAChamber = new GraphJFrameTotalHeatExchanger20090505( this.name, this.maxItemCount,  0, 0 );
    }
    //グラフ表示の準備
    if( this.isGVisible ){

       this.gData = new GraphCommonData2015[10];

       int i = 0;

       this.gData[i++] = new GraphCommonData2015( "DB_airIn0[℃]", this.maxItemCount, -20, 50, 0 );
       this.gData[i++] = new GraphCommonData2015( "DB_airIn1[℃]", this.maxItemCount, -20, 50, 0 );
       this.gData[i++] = new GraphCommonData2015( "DB_airOut0[℃]", this.maxItemCount, -20, 50, 0 );
       this.gData[i++] = new GraphCommonData2015( "DB_airOut0[℃]", this.maxItemCount, -20, 50, 0 );

       this.gData[i++] = new GraphCommonData2015( "X_airIn0[g/g]", this.maxItemCount, 0, 0.1, 1 );
       this.gData[i++] = new GraphCommonData2015( "X_airIn1[g/g]", this.maxItemCount, 0, 0.1, 1 );
       this.gData[i++] = new GraphCommonData2015( "X_airOut0[g/g]", this.maxItemCount, 0, 0.1, 1 );
       this.gData[i++] = new GraphCommonData2015( "X_airOut0[g/g]", this.maxItemCount, 0, 0.1, 1 );

       this.gData[i++] = new GraphCommonData2015( "M_airIn0[g/s]", this.maxItemCount, 0, 5000, 0 );
       this.gData[i++] = new GraphCommonData2015( "M_airIn1[g/s]", this.maxItemCount, 0, 5000, 0 );

       String[] yName = { "温度[℃]","絶対湿度[g/g]","流量[g/s]" };

       gGCJF = new GraphCommonJFrame2015( this.name+"_THEX", this.gData, yName );

    }
}


public void outputs() {
    if(super.sm == null || super.cm == null ){
       System.out.println( "OAChamber sm cm == null");
       return;
    }

    //今は毎回読み込む
    this.swcIn   = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ));
    this.swcInOA = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcInOA ));
    this.modIn   = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn ));

    //this.message.append( "(C)swcIn="+this.swcIn+"/modeIn="+this.modIn );

    //
    this.airIn0
    = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airIn0));
    if( this.isUseZoneAir ){
       this.zoneAirforSSinside._outputSetRA( this.airIn1 );
    }else{
       this.airIn1
       = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airIn1));
    }

    this.qS_air = 0;
    this.qL_air = 0;
```

```
        this.qT_air = 0;

        this.qSRatio = 0;
        this.qLRatio = 0;
        this.qTRatio = 0;


        if( Airswc.isOFF( this.swcIn ) || Airswc.isOFF( this.swcInOA ) ) {
           //停止時
           if( this.isRecord ){
              this.message.append( "(C)停止");
           }
           //内部ファンで吸排気するとき  airIn0とairIn1の風量を設定する  //20150409
           if( this.isOpeMyFan ){
              this.airIn0.setFlowRate( 0. );
              this.airIn1.setFlowRate( 0. );
           }
           //入口と出口の状態は同じとする
           this.airOut0.copyAllvalState( this.airIn0 );
           this.airOut1.copyAllvalState( this.airIn1 );

           //消費電力は0
           this.f_PE = 0;

        }else{
           //運転時;
           if( this.isRecord ){
              this.message.append( "(C)運転");
           }
           BestAir.checkOpeData( this.airIn0, "airIn0", this.moduleName, "outputs()", this.name, this.isRecord,
this.message );
           BestAir.checkOpeData( this.airIn1, "airIn1", this.moduleName, "outputs()", this.name, this.isRecord,
this.message );

           //バイパス制御のチェック
           if( this.isBypass ){
              this.modMy = Airmod.offVENTILATE( this.modMy );

              if( Airmod.isCOOL( this.modIn )){
                 if( this.airIn0.getEnthalpy() < this.airIn1.getEnthalpy() ){
                    this.modMy = Airmod.onVENTILATE( this.modMy );
                 }
              }else if( Airmod.isHEAT( this.modIn )){
                 if( this.airIn0.getEnthalpy() > this.airIn1.getEnthalpy() ){
                    this.modMy = Airmod.onVENTILATE( this.modMy );
                 }
              }else{

              }
           }

           double hosei = 1;//20190903

           //内部ファンで吸排気するとき  airIn0とairIn1の風量を設定する  //20121002nino
           if( this.isOpeMyFan ){

              //valInCtrlで運転補正するのは内部ファンで吸排気するときに限る
              if( this.isOpeValInCtrl ) {//20190903
                 hosei = this.valInCtrl.getValue();
              }

              this.airIn0.setFlowRate( this.defmFlowRate * hosei );
              this.airIn1.setFlowRate( this.defmFlowRate * hosei );
           }

           if( this.airIn0.getFlowRate() == 0 || this.airIn1.getFlowRate() == 0 ) {//外部ファンで吸排気しかつ、風量が０の時
              //停止時の計算
              //交換は行なわれない
              if( this.isRecord ){
                 this.message.append( "(C)風量=0orバイパス換気");
              }
              this.airOut0.copyAllvalState( this.airIn0 );
```

```
    this.airOut1.copyAllvalState( this.airIn1 );

}else if( Airmod.isVENTILATE( this.modIn ) ){//バイパス換気のとき
    //上位からのバイパス制御時の計算
    //交換は行なわれない
    if( this.isRecord ){
        this.message.append( "(C)上位バイパス換気指示");
    }
    this.airOut0.copyAllvalState( this.airIn0 );
    this.airOut1.copyAllvalState( this.airIn1 );

}else if( Airmod.isVENTILATE( this.modMy ) ){//バイパス換気のとき
    //上位からのバイパス制御時の計算
    //交換は行なわれない
    if( this.isRecord ){
        this.message.append( "(C)自己バイパス換気制御");
    }
    this.airOut0.copyAllvalState( this.airIn0 );
    this.airOut1.copyAllvalState( this.airIn1 );

}else{
    //全熱交換時の計算
    if( this.isRecord ){
        this.message.append( "(C)全熱交換運転");
    }
    //入条件
    this.m_airIn0 = this.airIn0.getFlowRate();
    this.m_airIn1 = this.airIn1.getFlowRate();
    this.t_airIn0 = this.airIn0.getTempDB();
    this.t_airIn1 = this.airIn1.getTempDB();
    this.e_airIn0 = this.airIn0.getEnthalpy();
    this.e_airIn1 = this.airIn1.getEnthalpy();

    //基準効率の算定
    double eff0;
    double eff1;
    double m01Ratio;
    double m10Ratio;
    double m0Ratio;
    double m1Ratio;
    double x0;
    double x1;
    double velocityS = 2.5;//基準風速[m/s]

    m01Ratio = this.m_airIn0 / this.m_airIn1;
    m10Ratio = this.m_airIn1 / this.m_airIn0;
    m0Ratio  = this.m_airIn0 / this.defmFlowRate;
    m1Ratio  = this.m_airIn1 / this.defmFlowRate;
    x0 = velocityS * m0Ratio;
    x1 = velocityS * m1Ratio;

    //特性式の上下限のチェック　と　補正
    if( x0 < 1.5 ){
        x0 = 1.5;
    }
    if( x0 > 5 ){
        x0 = 5;
    }
    if( m01Ratio < 0.7 ){
        m01Ratio = 0.7;
    }
    if( m01Ratio > 1.5 ){
        m01Ratio = 1.5;
    }
    if( x1 < 1.5 ){
        x1 = 1.5;
    }
    if( x1 > 5 ){
        x1 = 5;
    }
    if( m10Ratio < 0.7 ){
        m10Ratio = 0.7;
```

```
}
if( m1ORatio > 1.5 ){
   m1ORatio = 1.5;
}

eff0 = 0.006 * x0 * x0 - 0.069 * x0 + 1.1292;//風速による効率
eff0 = 0.0512 * m01Ratio * m01Ratio - 0.5338 * m01Ratio + 1.4826;//m01Ratioを考慮
eff1 = 0.006 * x1 * x1 - 0.069 * x1 + 1.1292;//風速による効率
eff1 = 0.0512 * m1ORatio * m1ORatio - 0.5338 * m1ORatio + 1.4826;//m01Ratioを考慮

//出口温度[℃]
this.t_airOut0
= this.t_airIn0 - (this.t_airIn0 - this.t_airIn1 ) * this.deffiS * eff0;
this.t_airOut1
= this.t_airIn1 - (this.t_airIn1 - this.t_airIn0 ) * this.deffiS * eff1;

//出口エンタルピJ/g
this.e_airOut0
= this.e_airIn0 - (this.e_airIn0 - this.e_airIn1 ) * this.deffiE * eff0;
this.e_airOut1
= this.e_airIn1 - (this.e_airIn1 - this.e_airIn0 ) * this.deffiE * eff1;

//System.out.println( "e0 = " + this.e_airIn0 + " e1 = " + this.e_airIn1 );

//出口の絶対湿度
this.x_airOut0 = Psychrometrics.FNXth( this.t_airOut0, this.e_airOut0 * 1000. );
this.x_airOut1 = Psychrometrics.FNXth( this.t_airOut1, this.e_airOut1 * 1000. );

//2つの入口絶対湿度よりも低い若しくは高い絶対湿度になってしまう不具合に対応（応急処置）
boolean TorF0 = false;//そのままでよいか
boolean TorF1 = false;
if (this.airIn0.getHumi() > this.airIn1.getHumi()) {// 外気絶対湿度g/g'が高い
   if (this.x_airOut0 > this.airIn0.getHumi()) {
      this.x_airOut0 = this.airIn0.getHumi();
   } else if (this.x_airOut0 < this.airIn1.getHumi()) {
      this.x_airOut0 = this.airIn1.getHumi();
   } else {
      TorF0 = true;
   }
   if (this.x_airOut1 > this.airIn0.getHumi()) {
      this.x_airOut1 = this.airIn0.getHumi();
   } else if (this.x_airOut1 < this.airIn1.getHumi()) {
      this.x_airOut1 = this.airIn1.getHumi();
   } else {
      TorF1 = true;
   }
} else if (this.airIn0.getHumi() < this.airIn1.getHumi()) {// 外気絶対湿度g/g'が低い
   if (this.x_airOut0 < this.airIn0.getHumi()) {
      this.x_airOut0 = this.airIn0.getHumi();
   } else if (this.x_airOut0 > this.airIn1.getHumi()) {
      this.x_airOut0 = this.airIn1.getHumi();
   }else {
      TorF0 = true;
   }
   if (this.x_airOut1 < this.airIn0.getHumi()) {
      this.x_airOut1 = this.airIn0.getHumi();
   } else if (this.x_airOut1 > this.airIn1.getHumi()) {
      this.x_airOut1 = this.airIn1.getHumi();
   }else {
      TorF1 = true;
   }
}
if(TorF0 == false) {
   //this.e_airIn0 = Psychrometrics.FNH(this.t_airOut0, this.x_airOut0)/1000;
   this.e_airOut0 = Psychrometrics.FNH(this.t_airOut0, this.x_airOut0)/1000;
}
if(TorF1 == false) {
   //this.e_airIn1 = Psychrometrics.FNH(this.t_airOut1, this.x_airOut1)/1000;
   this.e_airOut1 = Psychrometrics.FNH(this.t_airOut1, this.x_airOut1)/1000;
}
```

```
            //出口の状態セット
            this.airOut0.setTempDB( this.t_airOut0 );
            this.airOut0.setHumi( this.x_airOut0 );
            this.airOut0.setFlowRate( this.m_airIn0 );
            this.airOut0.setCO2ppm( this.airIn0.getCO2ppm() );//20131029
            //
            this.airOut1.setTempDB( this.t_airOut1 );
            this.airOut1.setHumi( this.x_airOut1 );
            this.airOut1.setFlowRate( this.m_airIn1 );
            this.airOut1.setCO2ppm( this.airIn1.getCO2ppm() );//20131029

            //処理熱量
            this.qS_air = ( this.airOut0.getTempDB() - this.airIn0.getTempDB() ) * this.CP* this.airIn0.getFlowRate();
            this.qT_air = ( this.e_airOut0 - this.e_airIn0 ) * this.airIn0.getFlowRate();
            this.qL_air = this.qT_air - this.qS_air;

            if( this.t_airIn1 == this.t_airIn0 ) {
               this.qSRatio = 0;
            }else {
               this.qSRatio = this.qS_air / ( ( this.t_airIn1 - this.t_airIn0 )  * this.CP * this.defmFlowRate );
            }

            if( this.e_airIn1 == this.e_airIn0 ) {
               this.qTRatio = 0;
            }else {
               this.qTRatio = this.qT_air / ( ( this.e_airIn1 - this.e_airIn0 ) * this.defmFlowRate );

            }
            if( this.airIn1.getHumi() - this.airIn0.getHumi() == 0) {
               this.qLRatio = 0;
            }else {
               this.qLRatio = ( this.airOut0.getHumi() - this.airIn0.getHumi() ) * this.airIn0.getFlowRate()
                     / ( ( this.airIn1.getHumi() - this.airIn0.getHumi() ) * this.defmFlowRate );
            }

         }

         //消費電力は0
         this.f_PE = this.dPEini * hosei;
      }

      //電力
      this.eleIn.setActivePower( this.f_PE );
      this.eleIn.setReactivePower(
            this.f_PE * Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 ) );
      this.eleIn.setPhase( this.phase );
      this.eleIn.setVoltage( this.voltage );
      this.eleIn.setFrequency( this.frequency );

      //ノードに設定
      //出力側ノード
      super.sm.setState( super.getConnectionNode(this.S_NODE_airOut0), this.airOut0 );
      super.sm.setState( super.getConnectionNode(this.S_NODE_airOut1), this.airOut1 );
      super.sm.setState(super.getConnectionNode( this.S_NODE_eleIn ), this.eleIn);

      //グラフデータ追加
      if( this.isGVisible ){
         gOAChamber.addData( BestTimeManager.getDateWeatherTime(), this.airIn0, this.airIn1, this.airOut0,
this.airOut1 );

         double[] gData = new double[10];
         int i = 0;

      /*
         this.gData = new GraphCommonData2015[10];

         int i = 0;

         this.gData[i++] = new GraphCommonData2015( "DB_airIn0[℃]", this.maxItemCount, -20, 50, 0 );
         this.gData[i++] = new GraphCommonData2015( "DB_airIn1[℃]", this.maxItemCount, -20, 50, 0 );
         this.gData[i++] = new GraphCommonData2015( "DB_airOut0[℃]", this.maxItemCount, -20, 50, 0 );
         this.gData[i++] = new GraphCommonData2015( "DB_airOut0[℃]", this.maxItemCount, -20, 50, 0 );
```

```
        this.gData[i++] = new GraphCommonData2015( "X_airIn0[g/g]", this.maxItemCount, 0, 0.1, 1 );
        this.gData[i++] = new GraphCommonData2015( "X_airIn1[g/g]", this.maxItemCount, 0, 0.1, 1 );
        this.gData[i++] = new GraphCommonData2015( "X_airOut0[g/g]", this.maxItemCount, 0, 0.1, 1 );
        this.gData[i++] = new GraphCommonData2015( "X_airOut0[g/g]", this.maxItemCount, 0, 0.1, 1 );

        this.gData[i++] = new GraphCommonData2015( "M_airIn0[g/s]", this.maxItemCount, 0, 5000, 0 );
        this.gData[i++] = new GraphCommonData2015( "M_airIn1[g/s]", this.maxItemCount, 0, 5000, 0 );

        String[] yName = { "温度[℃]","絶対湿度[g/g]","流量[g/s]" };

    */

        gData[i++] = this.airIn0.getTempDB();
        gData[i++] = this.airIn1.getTempDB();
        gData[i++] = this.airOut0.getTempDB();
        gData[i++] = this.airOut1.getTempDB();

        gData[i++] = this.airIn0.getHumi();
        gData[i++] = this.airIn1.getHumi();
        gData[i++] = this.airOut0.getHumi();
        gData[i++] = this.airOut1.getHumi();

        gData[i++] = this.airIn0.getFlowRate();
        gData[i++] = this.airIn1.getFlowRate();

        this.gGCJF.addData( BestTimeManager.getDateWeatherTime(), gData );
    }

    //記録ノード
    if( this.isRecord  && super.rm != null ){
        this.record();
    }

    message.setLength(0);
}


/**
 * 記録
 */
private void record(){

    if( CheckPrintModule.isPrintMessage ){
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                this.RECORD_message, this.name, message.toString() );
    }

    if( CheckPrintModule.isPrintEnergy ){
        //消費電力
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                this.RECORD_PE, this.name, AirFormat.df_2( this.f_PE));
    }

    if( CheckPrintModule.isPrintLoad ){
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                this.RECORD_qS_air, this.name, AirFormat.df_2( this.qS_air ));
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                this.RECORD_qL_air, this.name, AirFormat.df_2( this.qL_air ));
    }

    if( CheckPrintModule.isPrintStateOut ){
        //出口0　（給気）
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                this.RECORD_T_aOut0, this.name, AirFormat.df_2( this.airOut0.getTempDB()));
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                this.RECORD_X_aOut0, this.name, AirFormat.df_5( this.airOut0.getHumi()));
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                this.RECORD_M_aOut0, this.name, AirFormat.df_2( this.airOut0.getFlowRate()));
        //出口1　（排気）
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                this.RECORD_T_aOut1, this.name, AirFormat.df_2( this.airOut1.getTempDB()));
```

```
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_X_aOut1, this.name, AirFormat.df_5( this.airOut1.getHumi()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_M_aOut1, this.name, AirFormat.df_2( this.airOut1.getFlowRate()));
        }

        if( CheckPrintModule.isPrintStateMy ){
            //
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_qT_air, this.name, AirFormat.df_2( this.qT_air ));

            //private final String RECORD_mRatio_airIn0defm  = "全熱交換器_空気0/定格風量比#-#-";
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_mRatio_airIn0defm, this.name, AirFormat.df_2( this.airIn0.getFlowRate() /
this.defmFlowRate ));
            //private final String RECORD_mRatio_airIn1defm  = "全熱交換器_空気1/定格風量比#-#-";
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_mRatio_airIn1defm, this.name, AirFormat.df_2( this.airIn1.getFlowRate() /
this.defmFlowRate ));
            //private final String RECORD_mRatio_airIn01  = "全熱交換器_空気0/1風量比#-#-";
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_mRatio_airIn01, this.name, AirFormat.df_2( this.airIn0.getFlowRate() /
this.airIn1.getFlowRate() ));
            //private final String RECORD_mRatio_airIn10  = "全熱交換器_空気1/0風量比#-#-";
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_mRatio_airIn10, this.name, AirFormat.df_2( this.airIn1.getFlowRate() /
this.airIn0.getFlowRate() ));

            //private final String RECORD_qSRatio  = "全熱交換器_顕熱交換効率#-#-";
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_qSRatio, this.name, AirFormat.df_2( this.qSRatio ));
            //private final String RECORD_qLRatio  = "全熱交換器_潜熱交換効率#-#-";
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_qLRatio, this.name, AirFormat.df_2( this.qLRatio ));
            //private final String RECORD_qTRatio  = "全熱交換器_全熱交換効率#-#-";
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_qTRatio, this.name, AirFormat.df_2( this.qTRatio ));
        }

        if( CheckPrintModule.isPrintStateIn ){
            //入口0
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_T_aIn0, this.name, AirFormat.df_2( this.airIn0.getTempDB()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_X_aIn0, this.name, AirFormat.df_5( this.airIn0.getHumi()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_M_aIn0, this.name, AirFormat.df_2( this.airIn0.getFlowRate()));
            //入口1  (還気)
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_T_aIn1, this.name, AirFormat.df_2( this.airIn1.getTempDB()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_X_aIn1, this.name, AirFormat.df_5( this.airIn1.getHumi()));
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_M_aIn1, this.name, AirFormat.df_2( this.airIn1.getFlowRate()));
        }
    }


    public void update() {

        if( this.isUseZoneAir ){
            //this.zoneAirforSSinside._update( this.airOut1 );
            this.zoneAirforSSinside._update( this.airOut0 );
        }
    }


    public Object viewInternal(TestCommand cmd) {
        ArrayList<Object> result = new ArrayList<Object>();
        result.add(super.sm);
        result.add(super.cm);
        result.add(super.rm);
```

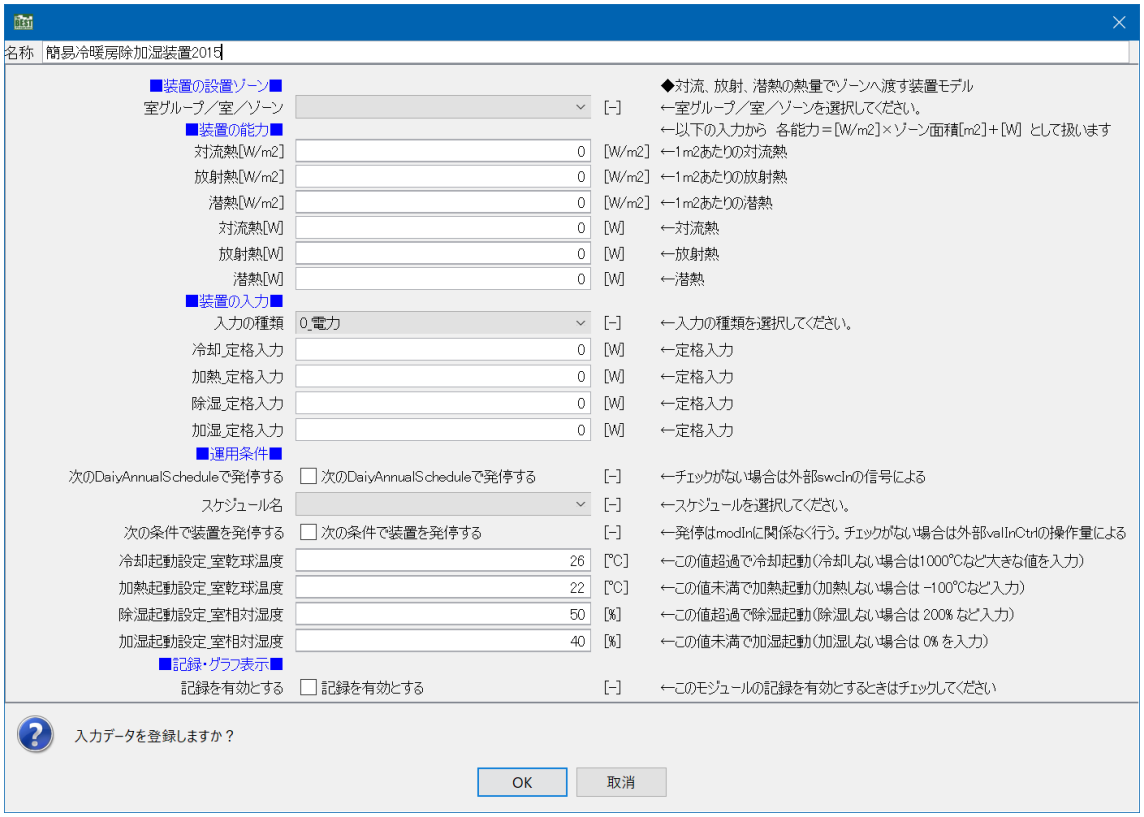
```
        //外部定義項目

        //記録ノードに設定する値
        if(airIn0 != null) {
            result.add(new Double(this.airIn0.getTempDB()));
            result.add(new Double(this.airIn0.getHumi()));
            result.add(new Double(this.airIn0.getFlowRate()));
        }else{
            result.add(new Double(0.0));
            result.add(new Double(0.0));
            result.add(new Double(0.0));
        }

        result.add(super.transferMapCommand);
        result.add(super.transferMapRecord);
        result.add(super.transferMapState);
        return result;
    }
}
```

「簡易冷暖房除加湿装置 2015」（場所：設備 2 015／個別分散 2015／）

モジュール名	簡易冷暖房除加湿装置 2015
クラス	ZoneSystemHeatGainCRLModule2015

（1）入力画面

・スペック



（2）モジュールの概要

冷却、除湿、加熱、加湿を行う簡易モジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

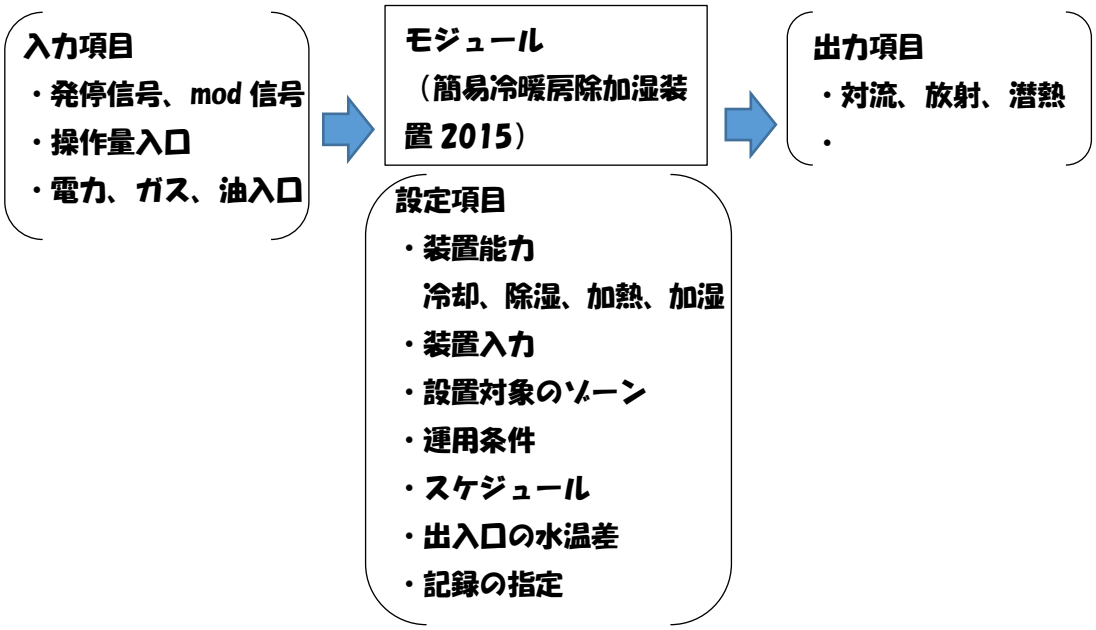


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	■装置の設置ゾーン■					―	―		◆対流、放射、潜熱の熱量でゾーンへ渡す装置モデル
2	室グループ／室／ゾーン	String	coreSpace	#RoomGroup	[-]	―	―		←室グループ／室／ゾーンを選択してください。
3	■装置の能力■					―	―		←以下の入力から　各能力＝[W/m2]×ゾーン面積[m2]＋[W]　として扱います
4	対流熱[W/m2]	double	heatCm2_S	0	[W/m2]	―	0		←1m2 あたりの対流熱
5	放射熱[W/m2]	double	heatRm2_S	0	[W/m2]	―	0		←1m2 あたりの放射熱
6	潜熱[W/m2]	double	heatLm2_S	0	[W/m2]	―	0		←1m2 あたりの潜熱
7	対流熱[W]	double	heatC_S	0	[W]	―	0		←対流熱
8	放射熱[W]	double	heatR_S	0	[W]	―	0		←放射熱
9	潜熱[W]	double	heatL_S	0	[W]	―	0		←潜熱
10	■装置の入力■					―	―		
11	入力の種類	String	typeEne	0_電力、1_ガス、2_油	[-]	―	―		←入力の種類を選択してください。
12	冷却_定格入力	double	deInp_C	0	[W]	―	0		←定格入力
13	加熱_定格入力	double	deInp_H	0	[W]	―	0		←定格入力
14	除湿_定格入力	double	deInp_RC	0	[W]	―	0		←定格入力
15	加湿_定格入力	double	deInp_RH	0	[W]	―	0		←定格入力
16	■運用条件■					―	―		
17	次の DaiyAnnualSchedule で発停する	boolean	isUseThisSche	FALSE	[-]	―	―		←チェックがない場合は外部 swcIn の信号による
18	スケジュール名	String	heaterSchedule	#Userm、時刻変動スケジ	[-]	―	―		←スケジュールを選択してください。

				ュール					
19	次の条件で装置を発停する	boolean	isUseThisCtrl	FALSE	[-]	—	—		←発停は modIn に関係なく行う。チェックがない場合は外部 valInCtrl の操作量による
20	冷却起動設定_室乾球温度	double	setTempC	26	[℃]	—	0		←この値超過で冷却起動（冷却しない場合は 1000℃など大きな値を入力）
21	加熱起動設定_室乾球温度	double	setTempH	22	[℃]	—	0		←この値未満で加熱起動（加熱しない場合は -100℃など入力）
22	除湿起動設定_室相対湿度	double	setHumiRC	50	[%]	—	0		←この値超過で除湿起動（除湿しない場合は 200% など入力）
23	加湿起動設定_室相対湿度	double	setHumiRH	40	[%]	—	0		←この値未満で加湿起動（加湿しない場合は 0% を入力）
24	■記録・グラフ表示■					—	—		
25	記録を有効とする	boolean	isRecord	FALSE	[-]	—	—		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	モード信号入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	対流量モード信号入口	L1_modInPID_HC	modInPID_H	-	制御	制御モード	入口	
5	放射用モード信号入口	L1_modInPID_HR	modInPID_HR	-	制御	制御モード	入口	
6	潜熱用モード信号入口	L1_modInPID_HL	modInPID_HL	-	制御	制御モード	入口	
7	Env 出口	L0_envOut	envOut	-	状態	室環境	出口	
8	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	
9	ガス入口	L0_gasIn	gasIn	-	状態	ガス	入口	
10		L0_gasIn	gasIn	-	状態	ガス	入口	
11	対流用操作量入口	L0_valInCtrl_HC	valInCtrl_HC	-	値	値	入口	
12	放射用操作量入口	L0_valInCtrl_HR	valInCtrl_HR	-	値	値	入口	
13	潜熱用操作量入口	L0_valInCtrl_HL	valInCtrl_HL	-	値	値	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	ZSHGain_Message#-#-	メッセージ	—	メッセージ
2	ZSHGain_処理対流熱量#W#熱量	処理した対流熱量//加熱が負値	W	My
3	ZSHGain_処理放射熱量#W#熱量	処理した放射熱量//加熱が負値	W	My
4	ZSHGain_処理顕熱量#W#熱量	処理した顕熱量//加熱が負値	W	負荷
5	ZSHGain_処理潜熱量#W#熱量	処理した潜熱量//加湿が負値	W	負荷
6	ZSHGain_処理対流熱量密度#W/m2#熱量	処理した対流熱量密度//加熱が負値	W/m2	My
7	ZSHGain_処理放射熱量密度#W/m2#熱量	処理した放射熱量密度//加熱が負値	W/m2	My
8	ZSHGain_処理潜熱熱量密度#W/m2#熱量	処理した潜熱量密度//加湿が負値	W/m2	My
9	ZSHGain_室 DB#℃#温度	室の乾球温度	℃	My
10	ZSHGain_室 X#g/g#湿度	室の絶対湿度	g/g	My
11	ZSHGain_室 RH#℃#相対湿度	室の相対湿度	%	My
12	ZSHGain_室 OT#℃#温度	室の作用温度	℃	My
13	ZSHGain_電力消費量#W#電力消費電	電力消費量	W	エネルギー
14	ZSHGain_ガス消費量#W#ガス消費量	ガス消費量	W	エネルギー
15	ZSHGain_油消費量#W#油消費量	油消費量	W	エネルギー

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestGas;
import jp.or.ibec.best.DO.BestOil;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.building.load.SystemHeatGain;
import jp.or.ibec.best.domain.building.manager.ScheduleManager;
import jp.or.ibec.best.domain.building.schedule.DailyAnnualSchedule;
import jp.or.ibec.best.domain.building.spaces.ISpace;
import jp.or.ibec.best.domain.building.spaces.ZoneEnv;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;


/**
 * 室内ヒーター：SystemHeatGainを使用するモジュールの例
 * 201304--201308
 *
 * @author HIROSHI NINOMIYA
 *
 */
public class ZoneSystemHeatGainCRLModule2015 extends AbstractBestModule implements
      IBestMetaModule {

   //
   private final String moduleName = "(ZoneSystemHeatGainCRLModule2015) ";
   //外部定義項目
   private final String SPEC_name = "名称";

   private final String SPEC_grzName = "室グループ／室／ゾーン";
// private final String SPEC_msName = "MultiSpaceName";
// private final String SPEC_zoneName = "ZoneName";

   private final String SPEC_setTempC  = "室温度冷却[℃]";
   private final String SPEC_setTempH  = "室温度加熱[℃]";
   private final String SPEC_setHumiRC = "室湿度除湿[-]";
   private final String SPEC_setHumiRH = "室湿度加湿[-]";

   private final String SPEC_heatCm2_S = "対流熱[W/m2]";
   private final String SPEC_heatRm2_S = "放射熱[W/m2]";
   private final String SPEC_heatLm2_S = "潜熱[W/m2]";
   private final String SPEC_heatC_S   = "対流熱[W]";
   private final String SPEC_heatR_S   = "放射熱[W]";
   private final String SPEC_heatL_S   = "潜熱[W]";

   private final String SPEC_typeEne  = "入力種類[-]";
   private final String SPEC_deInp_C  = "入力冷却[W]";
   private final String SPEC_deInp_H  = "入力加熱[W]";
   private final String SPEC_deInp_RC = "入力除湿[W]";
   private final String SPEC_deInp_RH = "入力加湿[W]";


   private final String SPEC_isUseThisCtrl = "isUseThisCtrl";//このOnOff制御をする
   private final String SPEC_isUseThisSche = "isUseThisSche";//このスケジュールを使用する
   private final String SPEC_shceduleName = "scheduleName";//ヒータスケジュール名
```

```
private final String SPEC_isGVisible   = "グラフを表示する";//このグラフを表示する
private final String SPEC_maxItemCount = "最大同時表示ステップ数";//最大同時表示ステップ数
private final String SPEC_isRecord     = "記録を有効とする";//このモジュールの記録を有効とする

private final String R_NODE = "L2_recOut";

private final String RECORD_message  = "ZSHGain_Message#-#-";
private final String RECORD_heatCsum = "ZSHGain_処理対流熱量#W#熱量";//加熱が負値
private final String RECORD_heatRsum = "ZSHGain_処理放射熱量#W#熱量";//加熱が負値
private final String RECORD_heatSsum = "ZSHGain_処理顕熱量#W#熱量";//加熱が負値
private final String RECORD_heatLsum = "ZSHGain_処理潜熱量#W#熱量";  //加湿が負値
private final String RECORD_heatC    = "ZSHGain_処理対流熱量密度#W/m2#熱量";//加熱が負値
private final String RECORD_heatR    = "ZSHGain_処理放射熱量密度#W/m2#熱量";//加熱が負値
private final String RECORD_heatL    = "ZSHGain_処理潜熱熱量密度#W/m2#熱量";  //加湿が負値

private final String RECORD_ta       = "ZSHGain_室DB#℃#温度";
private final String RECORD_x        = "ZSHGain_室X#g/g#湿度";
private final String RECORD_RH       = "ZSHGain_室RH#℃#相対湿度";
private final String RECORD_OT       = "ZSHGain_室OT#℃#温度";

private final String RECORD_opeEle   = "ZSHGain_電力消費量#W#電力消費電";
private final String RECORD_opeGas   = "ZSHGain_ガス消費量#W#ガス消費量";
private final String RECORD_opeOil   = "ZSHGain_油消費量#W#油消費量";

private final String C_NODE_swcIn      = "L1_swcIn";
private final String C_NODE_modIn      = "L1_modIn";
private final String C_NODE_modInPID_HC = "L1_modInPID_HC";
private final String C_NODE_modInPID_HR = "L1_modInPID_HR";
private final String C_NODE_modInPID_HL = "L1_modInPID_HL";

private final String S_NODE_envOut = "L0_envOut";
private final String S_NODE_eleIn  = "L0_eleIn";
private final String S_NODE_gasIn  = "L0_gasIn";
private final String S_NODE_oilIn  = "L0_oilIn";

private final String S_NODE_valInCtrl_HC = "L0_valInCtrl_HC";
private final String S_NODE_valInCtrl_HR = "L0_valInCtrl_HR";
private final String S_NODE_valInCtrl_HL = "L0_valInCtrl_HL";


private String name;//ヒータ名
private DailyAnnualSchedule heaterSchedule;//ヒータスケジュールインスタンス

private SystemHeatGain systemHeatGain; //建物側インスタンス
private double floorArea;//床面積[㎡]


private double setTempC;// = "室温冷房[℃]";
private double setTempH;// = "室温暖房[℃]";
private double setHumiRC;// = "室湿度冷房[-]";
private double setHumiRH;// = "室湿度暖房[-]";

private double heatCm2_S;// = "対流熱[W/m2]";
private double heatRm2_S;// = "放射[W/m2]";
private double heatLm2_S;// = "潜熱[W/m2]";
private double heatC_S;// = "対流熱[W]";
private double heatR_S;// = "放射[W]";
private double heatL_S;// = "潜熱[W]";
private double heatCsum_S;// = "対流熱[W]";
private double heatRsum_S;// = "放射[W]";
private double heatLsum_S;// = "潜熱[W]";

private boolean isUseThisCtrl = false;
private boolean isUseThisSche = false;
//
private double deInp_C;//  = "消費電力冷却[W]";
private double deInp_H;//  = "消費電力加熱[W]";
private double deInp_RC;// = "消費電力除湿[W]";
private double deInp_RH;// = "消費電力加湿[W]";

private String typeEne = null;
private int numTypeEne = 0;
```

```
private boolean isGVisible = false;//このグラフを表示する=true
private int  maxItemCount = 100;//最大同時表示ステップ数
private boolean isRecord = false;//記録を有効とする=true

private ZoneEnv   envOut = null;
private BestElectricity eleIn = null;
private BestGas gasIn = null;
private BestOil oilIn = null;

private BestValue valInCtrl_HC = null;
private BestValue valInCtrl_HR = null;
private BestValue valInCtrl_HL = null;

//制御信号など
private int swcIn;                //運転モード
private int modIn;
private int modInPID_HC;
private int modInPID_HR;
private int modInPID_HL;


private StringBuffer message = new StringBuffer();

private ISpace coreSpace   ;//建物側インスタンス


double heatCsum=0.;//対流熱[W/㎡]
double heatRsum=0.;//放射熱[W/㎡]
double heatLsum=0.;//潜熱[W/㎡]

double opeInp = 0;//消費電力[W]


@Override
public void setProfile(BestSpecs spec) {
   if(spec==null) return;
   Map<String, String> map=spec.getSpec();
   if(map==null) return;

   //名称を取得
   this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

   //室グループ、ゾーン名の取得
   String grzName = null;
   grzName = spec.getSpecValue( this.SPEC_grzName, "" );

   this.coreSpace = ZoneGRZName.getCoreSpace( grzName, this.name );

   //SystemHeatGainインスタンスを建物側に登録
   this.systemHeatGain = new SystemHeatGain();
   this.systemHeatGain.setProfile(
         ZoneGRZName.getMultiSpaceName( grzName, this.name ),
         ZoneGRZName.getZoneName( grzName, this.name ) );

   //室床面積[㎡]
   this.floorArea = this.systemHeatGain.getZone().getValue(ISpace.FLOORAREA);


   //ヒータスケジュールの設定
   String schName=(String)map.get(this.SPEC_shceduleName);
   this.heaterSchedule=null;
   //System.out.println( " schName = "+schName);
   if(schName!=null && !schName.equals("")){
      this.heaterSchedule=ScheduleManager.getDailyScheduleManager().
         getDailyAnnualSchedule(schName);
      ///System.out.println( " ******************************************************");
   }

   int print=1;
   if(print==0) return;
   //System.out.print("(SampleZoneSystemHeatGainModule) name:"+name);
```

```
//if(schName!=null)System.out.print(" heaterSch:"+schName);
//System.out.println();


//isUseThisCtrlを取得
this.isUseThisCtrl = spec.getSpecValue( this.SPEC_isUseThisCtrl, false );

this.isUseThisSche = spec.getSpecValue( this.SPEC_isUseThisSche, false );

//isGVisibleを取得
this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

// 最大同時表示ステップ数
this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

//isRecordを取得
this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );


//SPEC_setTempC = "室温冷房[℃]";
this.setTempC = spec.getSpecValue( this.SPEC_setTempC, 26.0 );

//SPEC_setTempH = "室温暖房[℃]";
this.setTempH = spec.getSpecValue( this.SPEC_setTempH, 22.0 );


//SPEC_setHumiRC = "室湿度冷房[-]";
this.setHumiRC = spec.getSpecValue( this.SPEC_setHumiRC, 0.5 );

//SPEC_setHumiRH = "室湿度暖房[-]";
this.setHumiRH = spec.getSpecValue( this.SPEC_setHumiRH, 0.4 );


//SPEC_heatCm2_S = "対流熱[W/m2]";
this.heatCm2_S = spec.getSpecValue( this.SPEC_heatCm2_S, 0. );

//SPEC_heatRm2_S = "放射熱[W/m2]";
this.heatRm2_S = spec.getSpecValue( this.SPEC_heatRm2_S, 0. );

//SPEC_heatLm2_S = "潜熱[W/m2]";
this.heatLm2_S = spec.getSpecValue( this.SPEC_heatLm2_S, 0. );


//SPEC_heatC_S = "対流熱[W]";
this.heatC_S = spec.getSpecValue( this.SPEC_heatC_S, 0. );

//SPEC_heatR_S = "放射熱[W]";
this.heatR_S = spec.getSpecValue( this.SPEC_heatR_S, 0. );

//SPEC_heatL_S = "潜熱[W]";
this.heatL_S = spec.getSpecValue( this.SPEC_heatL_S, 0. );


this.heatCsum_S = this.heatCm2_S * this.floorArea + this.heatC_S;
this.heatRsum_S = this.heatRm2_S * this.floorArea + this.heatR_S;
this.heatLsum_S = this.heatLm2_S * this.floorArea + this.heatL_S;

//private final String SPEC_dePE_C  = "消費電力冷却[W]";
this.deInp_C = spec.getSpecValue( this.SPEC_deInp_C, 0. );

//private final String SPEC_dePE_H  = "消費電力加熱[W]";
this.deInp_H = spec.getSpecValue( this.SPEC_deInp_H, 0. );

//private final String SPEC_dePE_RC = "消費電力除湿[W]";
this.deInp_RC = spec.getSpecValue( this.SPEC_deInp_RC, 0. );

//private final String SPEC_dePE_RH = "消費電力加湿[W]";
this.deInp_RH = spec.getSpecValue( this.SPEC_deInp_RH, 0. );

this.typeEne = spec.getSpecValue( this.SPEC_typeEne, "" );
if( this.typeEne.equals( "0_電力" ) ){
   this.numTypeEne = 0;
```

```
    }else if( this.typeEne.equals( "1_ガス" ) ){
       this.numTypeEne = 1;
    }else if( this.typeEne.equals( "2_油" ) ) {
       this.numTypeEne = 2;
    }else{
       this.numTypeEne = 0;
    }

}

public void initialize(IBestStateMessage stateNodes,
      IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
      IBestDomainSchedule schedule) {
    super.sm = stateNodes   ;     //状態ノードを取得
    super.cm = commandNodes ;     //制御ノードを取得
    super.rm = recordNodes  ;     //記録ノードを取得
    //envOut
    if( super.sm.getState( super.getConnectionNode( this.S_NODE_envOut)) != null ){
       this.envOut
       = (ZoneEnv) super.sm.getState( super.getConnectionNode( this.S_NODE_envOut));
    }else{
       this.envOut = new ZoneEnv();
       super.sm.setState( super.getConnectionNode( this.S_NODE_envOut ), this.envOut );
    }

    //eleInLighting
    this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

    //gasIn
    this.gasIn = BestGas.bindnode( super.transferMapState, super.sm, this.S_NODE_gasIn );

    //oilIn
    this.oilIn = BestOil.bindnode( super.transferMapState, super.sm, this.S_NODE_oilIn );

    //valInCtrl_HC
    this.valInCtrl_HC = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInCtrl_HC );

    //valInCtrl_HR
    this.valInCtrl_HR = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInCtrl_HR );

    //valInCtrl_HL
    this.valInCtrl_HL = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInCtrl_HL );

}



public void outputs() {


    this.swcIn = super.cm.getCommand(super.getConnectionNode( this.C_NODE_swcIn ));
    this.modIn = super.cm.getCommand(super.getConnectionNode( this.C_NODE_modIn ));
    this.modInPID_HC = super.cm.getCommand(super.getConnectionNode( this.C_NODE_modInPID_HC ));
    this.modInPID_HR = super.cm.getCommand(super.getConnectionNode( this.C_NODE_modInPID_HR ));
    this.modInPID_HL = super.cm.getCommand(super.getConnectionNode( this.C_NODE_modInPID_HL ));

    this.valInCtrl_HC = (BestValue) super.sm.getState( super.getConnectionNode( this.S_NODE_valInCtrl_HC ));
    this.valInCtrl_HR = (BestValue) super.sm.getState( super.getConnectionNode( this.S_NODE_valInCtrl_HR ));
    this.valInCtrl_HL = (BestValue) super.sm.getState( super.getConnectionNode( this.S_NODE_valInCtrl_HL ));

    //ヒータ運転スケジュール（1：運転、0：停止）
    double sch=1.;

    boolean isSWCon = false;

    if( this.isUseThisSche ){
       if( this.heaterSchedule != null ){
          sch = this.heaterSchedule.getSchedule();
          if( sch > 0 ){
             isSWCon = true;
             //System.out.println("[1]");
          }else{
```

```
            isSWCon = false;
            //System.out.println("[2]");
        }
    }else{
        isSWCon = false;
        //System.out.println("[3]");
    }

}else{
    if( Airswc.isON( this.swcIn ) ){
        isSWCon = true;
        //System.out.println("[4]");
    }else{
        isSWCon = false;
        //System.out.println("[5]");
    }
}

double rh=this.systemHeatGain.getZoneEnv().getRh();

this.heatCsum = 0;
this.heatRsum = 0;
this.heatLsum = 0;

this.opeInp = 0;

if( !isSWCon ){
    //すべて0のまま

}else if( this.isUseThisCtrl ){
    //this.rateCtrl = 1.;


    //室温が22℃より低いとき加熱
    if( sch==1. && this.systemHeatGain.getZoneEnv().getTa() < this.setTempH ){
        this.heatCsum = this.heatCsum_S;
        this.heatRsum = this.heatRsum_S;
        this.opeInp += this.deInp_H;
    }
    //室温が26℃より高いとき冷却
    if( sch==1. && this.systemHeatGain.getZoneEnv().getTa() > this.setTempC ){
        this.heatCsum = - this.heatCsum_S;
        this.heatRsum = - this.heatRsum_S;
        this.opeInp += this.deInp_C;
    }
    //室温が40%より低いとき加湿
    if( sch==1. && rh<this.setHumiRH*100. ){
        this.heatLsum = this.heatLsum_S;
        this.opeInp += this.deInp_RH;
    }
    //室温が40%より高いとき除湿
    if( sch==1. && rh>this.setHumiRC*100. ){
        this.heatLsum = - this.heatLsum_S;
        this.opeInp += this.deInp_RC;
    }

}else{

    //
    double heatCRsum = 0.;
    if (Airmod.isCOOL(this.modIn) && Airmod.isHEAT(this.modIn)) {

        if (Airmod.isCOOL(this.modInPID_HC)) {
            this.heatCsum = -this.heatCsum_S * this.valInCtrl_HC.getValue();
            this.opeInp += this.deInp_C * this.heatCsum * this.valInCtrl_HC.getValue();
        } else if (Airmod.isHEAT(this.modInPID_HC)) {
            this.heatCsum = this.heatCsum_S * this.valInCtrl_HC.getValue();
            this.opeInp += this.deInp_H * this.heatCsum * this.valInCtrl_HC.getValue();
        } else {
            this.heatCsum = 0;
        }
        if (Airmod.isCOOL(this.modInPID_HR)) {
```

```
            this.heatRsum = -this.heatRsum_S * this.valInCtrl_HR.getValue();
            this.opeInp += this.deInp_C * this.heatRsum * this.valInCtrl_HR.getValue();
        } else if (Airmod.isHEAT(this.modInPID_HR)) {
            this.heatRsum = this.heatRsum_S * this.valInCtrl_HR.getValue();
            this.opeInp += this.deInp_H * this.heatRsum * this.valInCtrl_HR.getValue();
        } else {
            this.heatRsum = 0;
        }
        heatCRsum = this.heatCsum + this.heatRsum;
        if (heatCRsum == 0.) {
            this.opeInp = 0;
        } else {
            this.opeInp = this.opeInp /heatCRsum;
        }

    } else if (Airmod.isCOOL(this.modIn)) {
        this.heatCsum = -this.heatCsum_S * this.valInCtrl_HC.getValue();
        this.heatRsum = -this.heatRsum_S * this.valInCtrl_HR.getValue();
        heatCRsum = this.heatCsum + this.heatRsum;
        if (heatCRsum == 0.) {
            this.opeInp = 0;
        } else {
        this.opeInp += this.deInp_C * this.heatCsum / heatCRsum * this.valInCtrl_HC.getValue();
        this.opeInp += this.deInp_C * this.heatRsum / heatCRsum * this.valInCtrl_HR.getValue();
        }
    } else if (Airmod.isHEAT(this.modIn)) {
        this.heatCsum = this.heatCsum_S * this.valInCtrl_HC.getValue();
        this.heatRsum = this.heatRsum_S * this.valInCtrl_HR.getValue();
        heatCRsum = this.heatCsum + this.heatRsum;
        if (heatCRsum == 0.) {
            this.opeInp = 0;
        } else {
            this.opeInp += this.deInp_H * this.heatCsum / heatCRsum * this.valInCtrl_HC.getValue();
            this.opeInp += this.deInp_H * this.heatRsum / heatCRsum * this.valInCtrl_HR.getValue();
        }
    } else {
        this.heatCsum = 0;
        this.heatRsum = 0;

    }

    //
    if( Airmod.isHUMIDIFY( this.modIn ) &&  Airmod.isDEHUMIDIFY( this.modIn )){

        if( Airmod.isDEHUMIDIFY( this.modInPID_HL )){
            this.heatLsum = - this.heatLsum_S * this.valInCtrl_HL.getValue();
            this.opeInp += this.deInp_RC * this.valInCtrl_HL.getValue();

        }else if( Airmod.isHUMIDIFY( this.modInPID_HL )){
            this.heatLsum = this.heatLsum_S * this.valInCtrl_HL.getValue();
            this.opeInp += this.deInp_RH * this.valInCtrl_HL.getValue();

        }else{
            this.heatLsum = 0;
        }

    }else if( Airmod.isDEHUMIDIFY( this.modInPID_HL )){
        this.heatLsum = - this.heatLsum_S * this.valInCtrl_HL.getValue();
        this.opeInp += this.deInp_RC * this.valInCtrl_HL.getValue();

    }else if( Airmod.isHUMIDIFY( this.modInPID_HL )){
        this.heatLsum = this.heatLsum_S * this.valInCtrl_HL.getValue();
        this.opeInp += this.deInp_RH * this.valInCtrl_HL.getValue();

    }else{
        this.heatLsum = 0;

    }

}
```

```
//    System.out.println( "heatC="+heatC+" heatR="+heatR+" heatL="+heatL+" floorArea="+floorArea+" sch="+sch);

      if( this.numTypeEne == 0 ){
         this.eleIn.setActivePower( this.opeInp );

      }else if( this.numTypeEne == 1 ){
         this.gasIn.setWatt( this.opeInp );

      }else if( this.numTypeEne == 2 ){
         this.oilIn.setWatt( this.opeInp );

      }else{
         this.eleIn.setActivePower( this.opeInp );

      }

      //建物側へ熱量[W]を渡す
      this.systemHeatGain.integrateHeatGain(
            this.heatCsum,
            this.heatRsum,
            this.heatLsum,
            BestTimeManager.getCurrentInterval());

      this.envOut.setEnv( this.systemHeatGain.getZoneEnv().getTa(),
            this.systemHeatGain.getZoneEnv().getXa(),
            this.systemHeatGain.getZoneEnv().getPMV(),
            this.systemHeatGain.getZoneEnv().getOT());

      if( this.isRecord  && super.rm != null  ){
         this.record();
      }

      this.message.setLength(0);

   }


   private void record(){
      if( CheckPrintModule.isPrintMessage ){
         super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
               this.RECORD_message, this.name, this.message.toString() );
      }

      if( CheckPrintModule.isPrintEnergy ){
         if( this.numTypeEne == 0 ){
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_opeEle, this.name, AirFormat.df_1( this.eleIn.getActivePower()) );
         }else if( this.numTypeEne == 1 ){
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_opeGas, this.name, AirFormat.df_1( this.gasIn.getWatt()) );
         }else if( this.numTypeEne == 2 ){
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_opeOil, this.name, AirFormat.df_1( this.oilIn.getWatt()) );
         }
      }

      if( CheckPrintModule.isPrintLoad ){
         super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_heatSsum, this.name, AirFormat.df_1( this.heatRsum + this.heatCsum ));
         super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_heatLsum, this.name, AirFormat.df_1( this.heatLsum ));

      }

      //if( CheckPrintModule.isPrintStateOut ){
         //出口
      //}

      if( CheckPrintModule.isPrintStateMy ){
         //記録ノードに設定
         super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_heatCsum, this.name, AirFormat.df_1( this.heatCsum ));
```

```
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_heatRsum, this.name, AirFormat.df_1( this.heatRsum ));

            if( this.floorArea != 0 ){
               super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                     this.RECORD_heatC, this.name, AirFormat.df_1( this.heatCsum / this.floorArea ));
               super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                     this.RECORD_heatR, this.name, AirFormat.df_1( this.heatRsum / this.floorArea ));
               super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                     this.RECORD_heatL, this.name, AirFormat.df_1( this.heatLsum / this.floorArea ));
            }
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_ta, this.name, AirFormat.df_2( this.systemHeatGain.getZoneEnv().getTa()) );
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_x, this.name, AirFormat.df_2( this.systemHeatGain.getZoneEnv().getXa()) );
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_RH, this.name, AirFormat.df_2( this.systemHeatGain.getZoneEnv().getRh()) );
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_OT, this.name, AirFormat.df_2( this.systemHeatGain.getZoneEnv().getOT()) );
         }

         //if( CheckPrintModule.isPrintStateIn ){
            //入口
         //}
      }


      public void update() {
      }


      public Object viewInternal(TestCommand cmd) {
         List<Object> result = new ArrayList<Object>();

         result.add( this.heatCsum );
         result.add( this.heatRsum );
         result.add( this.heatLsum );

         result.add( this.swcIn );

         result.add( this.isGVisible );
         result.add( this.maxItemCount );
         result.add( this.coreSpace );
         result.add( this.eleIn.getActivePower() );
         result.add( this.gasIn.getWatt() );
         result.add( this.oilIn.getWatt() );


         return result;
      }
   }
```

「簡易中央監視」（場所：設備 2015／制御機器 2015）

モジュール名	簡易中央監視 2009
クラス	ControlMEPAModule20090101

（1）入力画面

・スペック

（2）モジュールの概要

本モジュールは、各機器の運転状態（運転・停止）及び運転モード（冷房、暖房、冷暖房）を決定・出力する。本モジュールのスペック入力で指定された運転時間や冷暖房期間に関する情報から、現在の時刻の運転状態と運転モードを設備項目別に出力する。これらの出力を各機器に接続することにより、機器の運転状態や運転モードを制御することができる。

なお、空調機制御モジュール、熱源制御モジュールによってこのモジュールの代替が可能である。

（3）モジュールの入出力

スペック情報として指定する機器の運転に関する開始時刻・終了時刻や冷暖房期間から、現在の運転状態（0：停止、1：運転）及び運転モード（1：冷房、2：暖房、3：冷暖房）を設備項目（主制御、空調、電気、衛生、建築、予備 1、予備 2）ごとに出力する。

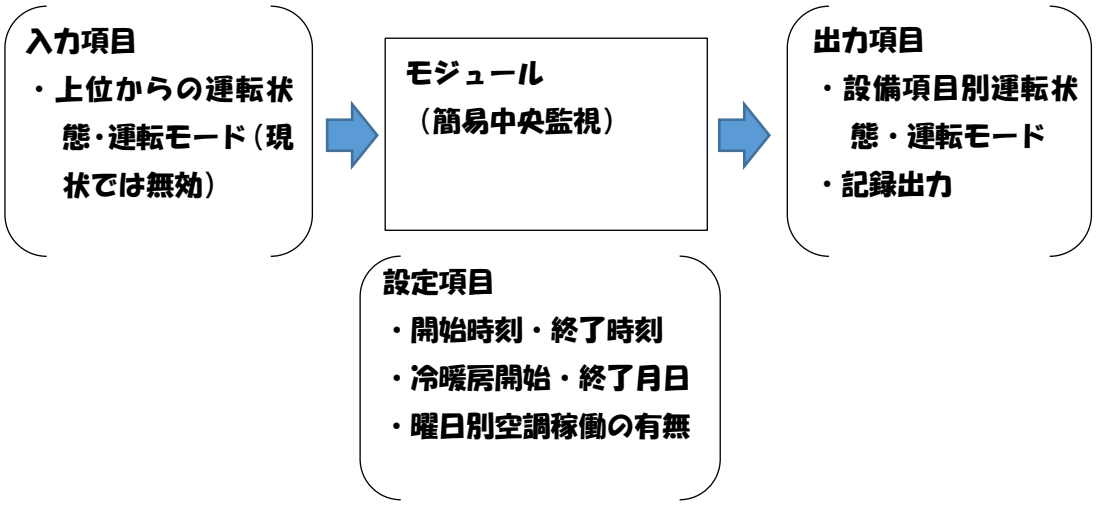


図 1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
①	■運転スケジュール■								
1	主制御　開始時刻ー終了時刻	String	MainOpe_START_END	0:00-24:00	[時:分]-[時:分]			◎	時刻と分を半角の[:]で、開始と終了を半角の[-]で区切る
2	周辺制御　開始時刻ー終了時刻	String		0:00-24:00	[時:分]-[時:分]				時刻と分を半角の[:]で、開始と終了を半角の[-]で区切る。現状では引用されない
3	冷房　開始月日ー終了月日	String	COOL_START_END	5/1-11/30	[月/日]-[月/日]			◎	月と日を半角の[:]で、開始と終了を半角の[-]で区切る
4	暖房　開始月日ー終了月日	String	HEAT_START_END	12/1-4/30	[月/日]-[月/日]			◎	月と日を半角の[:]で、開始と終了を半角の[-]で区切る
5	Main 制御 mode	int		1	[-]				主制御 mode。現状では無効
6	M 制御 mode	int		1	[-]				空調制御 mode。現状では無効
7	E 制御 mode	int		1	[-]				電気制御 mode。現状では無効
8	P 制御 mode	int		1	[-]				衛生制御 mode。現状では無効
9	A 制御 mode	int		1	[-]				建築制御 mode。現状では無効
10	R1 制御 mode	int		1	[-]				予備 1 制御 mode。現状では無効
11	R2 制御 mode	int		1	[-]				予備 2 制御 mode。現状では無効
12	空調 swc 日曜日	boolean	SWCon_AC_Sun	FALSE	[-]			◎	空調運転する場合 TRUE
13	空調 swc 月曜日	boolean	SWCon_AC_Mon	TRUE	[-]			◎	空調運転する場合 TRUE
14	空調 swc 火曜日	boolean	SWCon_AC_Tue	TRUE	[-]			◎	空調運転する場合 TRUE
15	空調 swc 水曜日	boolean	SWCon_AC_Wed	TRUE	[-]			◎	空調運転する場合 TRUE
16	空調 swc 木曜日	boolean	SWCon_AC_Thu	TRUE	[-]			◎	空調運転する場合 TRUE
17	空調 swc 金曜日	boolean	SWCon_AC_Fri	TRUE	[-]			◎	空調運転する場合 TRUE
18	空調 swc 土曜日	boolean	SWCon_AC_Sat	FALSE	[-]			◎	空調運転する場合 TRUE
19	空調 swc 祝日	boolean	SWCon_AC_Hol	FALSE	[-]			◎	空調運転する場合 TRUE
20	空調 swc 特別日	boolean	SWCon_AC_Spc	FALSE	[-]			◎	空調運転する場合 TRUE

②	■記録・グラフ表示■								
21	グラフを表示する	boolean	isGVisible	FALSE(チェック無し)	[-]	ー	ー	○	グラフを表示するときはチェック
22	最大同時表示ステップ数	Int	maxItemCount	100	[-]	ー	ー	○	グラフに同時表示する最大ステップ数を入力
23	記録を有効とする	boolean	isRecord	FALSE(チェック無し)	[-]	ー	ー	○	このモジュールの記録を有効とするときはチェック

（5）シーケンス接続（図1における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	運転状態	L1_swcIn	SWCin		制御	On/Off 信号	入口	群管理などからの上位入力を想定したもの。現状では無効
2	運転モード	L1_modIn	MODEin		制御	制御モード	入口	群管理などからの上位入力を想定したもの。現状では無効
3	運転状態(主出力)	L1_swcOutMain	SWC_{out_Main}		制御	On/Off 信号	出口	0:停止、1:運転
4	運転状態(空調)	L1_swcOutM	SWC_{out_M}		制御	On/Off 信号	出口	0:停止、1:運転
5	運転状態(電気)	L1_swcOutE	SWC_{out_E}		制御	On/Off 信号	出口	0:停止、1:運転
6	運転状態(衛生)	L1_swcOutP	SWC_{out_P}		制御	On/Off 信号	出口	0:停止、1:運転
7	運転状態(建築)	L1_swcOutA	SWC_{out_A}		制御	On/Off 信号	出口	0:停止、1:運転
8	運転状態(予備 1)	L1_swcOutR1	SWC_{out_R1}		制御	On/Off 信号	出口	0:停止、1:運転
9	運転状態(予備 2)	L1_swcOutR2	SWC_{out_R2}		制御	On/Off 信号	出口	0:停止、1:運転
10	運転モード(主出力)	L1_modOutMain	$MODE_{out\_Main}$		制御	制御モード	出口	1:冷房、2:暖房、3:冷暖房
11	運転モード(空調)	L1_modOutM	$MODE_{out\_M}$		制御	制御モード	出口	1:冷房、2:暖房、3:冷暖房
12	運転モード(電気)	L1_modOutE	$MODE_{out\_E}$		制御	制御モード	出口	1:冷房、2:暖房、3:冷暖房
13	運転モード(衛生)	L1_modOutP	$MODE_{out\_P}$		制御	制御モード	出口	1:冷房、2:暖房、3:冷暖房
14	運転モード(建築)	L1_modOutA	$MODE_{out\_A}$		制御	制御モード	出口	1:冷房、2:暖房、3:冷暖房
15	運転モード(予備1)	L1_modOutR1	$MODE_{out\_R1}$		制御	制御モード	出口	1:冷房、2:暖房、3:冷暖房
16	運転モード(予備2)	L1_modOutR2	$MODE_{out\_R2}$		制御	制御モード	出口	1:冷房、2:暖房、3:冷暖房
17	記録	L2_recOut	REC		記録	メモリ	出口	空調記録モジュールへ運転状態等を出力する。

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
2	上位からの運転状態信号（L1_swcIn）	運転状態	−	入口
3	上位からの運転モード信号（L1_modIn）	運転モード	−	入口
4	運転状態（主出力）（L1_swcOutMain）	運転状態	−	出口
6	運転モード（主出力）（L1_modOutMain）	運転モード	−	出口

（7）計算フロー・計算内容

1) 運転状態（主出力）、運転モード（主出力）の算出

スペック入力のうち、「主制御　開始時刻-終了時刻」に相当する文字列”MainOpe_START_END”から開始時刻と終了時刻を読み取り、計算時刻が開始終了時刻と終了時刻の間の場合には、SWCout_Main=1（運転）、その他の場合には SWCout_Main=0（停止）とする（開始時刻＝終了時刻の場合は停止とする）。

また、スペック入力のうち、「冷房　開始月日－終了月日」、「暖房　開始月日－終了月日」に相当する文字列”COOL_START_END”、”HEAT_START_END”から、冷房及び暖房の開始月日と終了月日を読み取り、計算日が開始月日と終了月日の間にあるかどうかで、冷房期か否か、及び暖房期か否かを判定する。冷房期にのみ該当する場合には、MODEout_Main=1（冷房）、暖房期にのみ該当する場合には MODEout_Main=2（暖房）、冷房期かつ暖房期の場合には MODEout_Main=3（冷暖房）とする（冷房、暖房ともに開始月日＝終了月日の場合は MODEout_Main=0 となる）。

2) 各設備項目の運転状態、運転モードの算出（空調以外）

設備項目（電気、衛生、建築、予備 1、予備 2）ごとに運転状態・運転モードを算出する枠組みになっているが、現状では、いずれの設備項目についても上記 1)の主出力と同一の運転状態・運転モードとする。

SWCout_E = SWCout_P = SWCout_A = SWCout_R1 = SWCout_R2 = SWCout_Main

MODEout_E = MODEout_P = MODEout_A = MODEout_R1 = MODEout_R2
= MODEout_Main

3) 空調運転状態、空調運転モードの算出

空調運転状態については、上記 1)の主出力の運転状態の他、曜日種別を考慮して算出する。スペック入力のうち SWCon_AC_Sun～SWCon_AC_Spc を参照し、計算日の曜日種別が運転日として指定されている場合には、

SWCout_M = SWCout_Main

運転日でない場合には、

SWCout_M = 0（停止）

とする。

空調運転モードについては、

MODEout_M = MODEout_Main

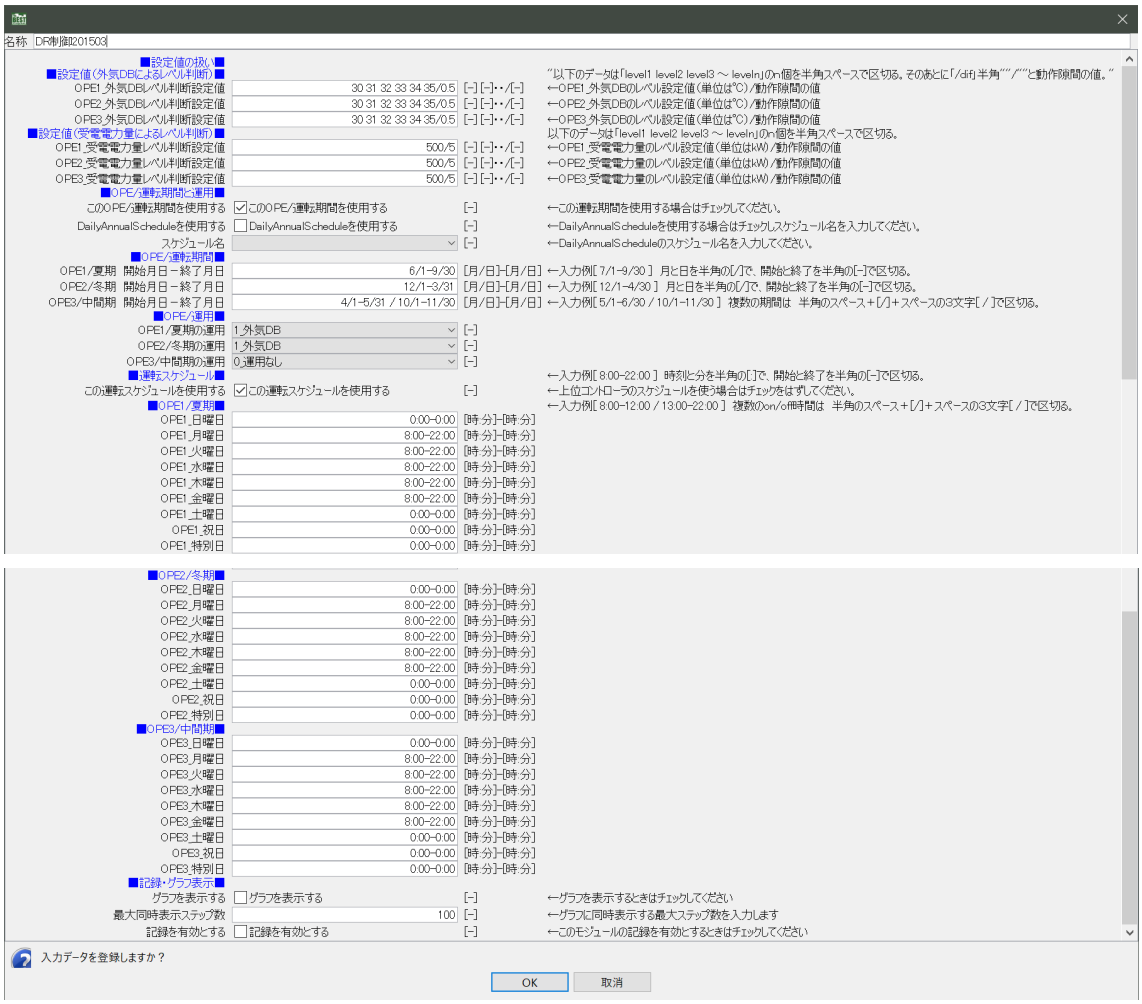
とする。

「DR 制御 201503」（場所：設備 2015／制御機器 2015）

モジュール名	DR 制御
クラス	ControlDRModule201503

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、DR 制御（デマンドレスポンス制御）の親機として、DR 制御のレベル

設定および DR 制御信号の発令・終了を DR 制御の子機に対して行う。

DR 制御の発令は、ユーザーが入力する外気温度 DB あるいは受電電力量をもとにしたレベル判断設定値を超えた場合に、それぞれのレベルを発信する。その設定値は、OPE1、OPE2、OPE3 に分けてセットできる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

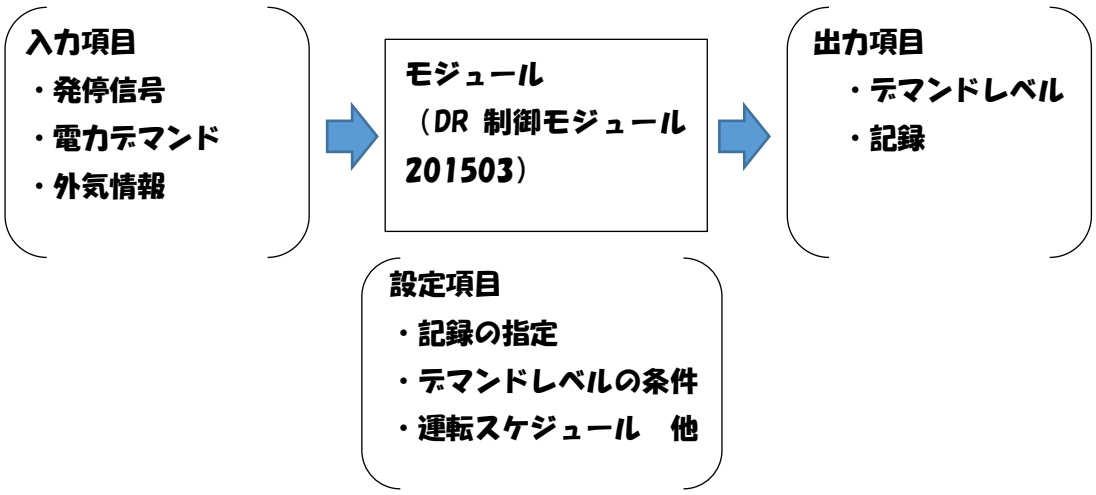


図１　モジュールの設定項目・入力項目・出力項目

気象データの流れについて次のような使い方がある

①　境界条件のデータファイルから気象データを取り込む場合

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name			—	—		
1	■設定値（外気 DB によるレベル判断）■	–	–	–	–				以下のデータは「level1 level2 level3 ～ leveln」の n 個を半角スペースで区切る。そのあとに「/dif」半角"/"と動作隙間の値。
2	OPE1_外気 DB レベル判断設定値	String	SPEC_OPE1_DBOALevelList	30 31 32 33 34 35/0.5	[-] [-]・・/[-]	—	—		←OPE1_外気 DB のレベル設定値（単位は℃）/動作隙間の値
3	OPE2_外気 DB レベル判断設定値	String	SPEC_OPE2_DBOALevelList	30 31 32 33 34 35/0.5	[-] [-]・・/[-]	—	—		←OPE2_外気 DB のレベル設定値（単位は℃）/動作隙間の値
4	OPE3_外気 DB レベル判断設定値	String	SPEC_OPE3_DBOALevelList	30 31 32 33 34 35/0.5	[-] [-]・・/[-]	—	—		←OPE3_外気 DB のレベル設定値（単位は℃）/動作隙間の値
5	■設定値（受電電力量によるレベル判断）■								以下のデータは「level1 level2 level3 ～ leveln」の n 個を半角スペースで区切る。
6	OPE1_受電電力量レベル判断設定値	String	SPEC_OPE1_RELELevelList	500/5	[-] [-]・・/[-]	—	—		←OPE1_受電電力量のレベル設定値（単位は kW）/動作隙間の値
7	OPE2_受電電力量レベル判断設定値	String	SPEC_OPE2_RELELevelList	500/5	[-] [-]・・/[-]	—	—		←OPE2_受電電力量のレベル設定値（単位は kW）/動作隙間の値
8	OPE3_受電電力量レベル判断設定値	String	SPEC_OPE3_RELELevelList	500/5	[-] [-]・・/[-]	—	—		←OPE3_受電電力量のレベル設定値（単位は kW）/動作隙間の値
9	■OPE/運転期間と運用■								
10	この OPE/運転期間を使用する	Boolean	isUseThisMode	TRUE	[-]	—	—		←この運転期間を使用する場合はチェックしてください。

11	DailyAnnualSchedule を使用する	Boolean	isUseDailyAnnualSchedule	FALSE	[-]	ー	ー		←DailyAnnualSchedule を使用する場合はチェックしスケジュール名を入力してください。
12	スケジュール名	String	schName	#Userm、時刻変動スケジュール、年間スケジュール	[-]	ー	ー		←DailyAnnualSchedule のスケジュール名を入力してください。
13	■OPE/運転期間■								
14	OPE1/夏期　開始月日－終了月日	String	airOpeSch / SPEC_OPE1_START_END	6/1-9/30	[月/日]-[月/日]	ー	ー		←入力例[7/1-9/30]　月と日を半角の[/]で、開始と終了を半角の[-]で区切る。
15	OPE2/冬期　開始月日－終了月日	String	airOpeSch / SPEC_OPE2_START_END	12/1-3/31	[月/日]-[月/日]	ー	ー		←入力例[12/1-4/30]　月と日を半角の[/]で、開始と終了を半角の[-]で区切る。
16	OPE3/中間期　開始月日－終了月日	String	airOpeSch / SPEC_OPE3_START_END	4/1-5/31 / 10/1-11/30	[月/日]-[月/日]	ー	ー		←入力例[5/1-6/30 / 10/1-11/30]　複数の期間は　半角のスペース＋[/]＋スペースの３文字[/]で区切る。
17	■OPE/運用■								
18	OPE1/夏期の運用	String	airOpeSch / SPEC_OPE1_mod	0_運用なし、1_外気DB、2_受電電力量、3_外気DB＋受電電力量	[-]	ー	ー		
19	OPE2/冬期の運用	String	airOpeSch / SPEC_OPE2_mod	0_運用なし、1_外気DB、2_受電電力量、3_外気DB＋受電電力量	[-]	ー	ー		
20	OPE3/中間期の運用	String	airOpeSch / SPEC_OPE3_mod	0_運用なし、1_外気DB、2_受電電力量、3_外気DB＋受電電力量	[-]	ー	ー		

21	■運転スケジュール■					—	—		←入力例[8:00-22:00] 時刻と分を半角の[:]で、開始と終了を半角の[-]で区切る。
22	この運転スケジュールを使用する	Boolean	isUseThisOpe	TRUE	[-]	—	—		←上位コントローラのスケジュールを使う場合はチェックをはずしてください。
23	■OPE1/夏期■								←入力例[8:00-12:00 / 13:00-22:00] 複数の on/off 時間は半角のスペース＋[/]＋スペースの 3 文字[/]で区切る。
24	OPE1_日曜日	String	airOpeSch / SPEC_ACOpe1_SUN	0:00-0:00	[時:分]-[時:分]	—	—		
25	OPE1_月曜日	String	airOpeSch / SPEC_ACOpe1_MON	8:00-22:00	[時:分]-[時:分]	—	—		
26	OPE1_火曜日	String	airOpeSch / SPEC_ACOpe1_TUE	8:00-22:00	[時:分]-[時:分]	—	—		
27	OPE1_水曜日	String	airOpeSch / SPEC_ACOpe1_WED	8:00-22:00	[時:分]-[時:分]	—	—		
28	OPE1_木曜日	String	airOpeSch / SPEC_ACOpe1_THU	8:00-22:00	[時:分]-[時:分]	—	—		
29	OPE1_金曜日	String	airOpeSch / SPEC_ACOpe1_FRI	8:00-22:00	[時:分]-[時:分]	—	—		
30	OPE1_土曜日	String	airOpeSch / SPEC_ACOpe1_SAT	0:00-0:00	[時:分]-[時:分]	—	—		
31	OPE1_祝日	String	airOpeSch / SPEC_ACOpe1_HOL	0:00-0:00	[時:分]-[時:分]	—	—		
32	OPE1_特別日	String	airOpeSch / SPEC_ACOpe1_SPC	0:00-0:00	[時:分]-[時:分]	—	—		
33	■OPE2/冬期■								
34	OPE2_日曜日	String	airOpeSch / SPEC_ACOpe2_SUN	0:00-0:00	[時:分]-[時:分]	—	—		
35	OPE2_月曜日	String	airOpeSch / SPEC_ACOpe2_MON	8:00-22:00	[時:分]-[時:分]	—	—		
36	OPE2_火曜日	String	airOpeSch / SPEC_ACOpe2_TUE	8:00-22:00	[時:分]-[時:分]	—	—		

37	OPE2_水曜日	String	airOpeSch / SPEC_ACOpe2_WED	8:00-22:00	[時:分]-[時:分]	—	—		
38	OPE2_木曜日	String	airOpeSch / SPEC_ACOpe2_THU	8:00-22:00	[時:分]-[時:分]	—	—		
39	OPE2_金曜日	String	airOpeSch / SPEC_ACOpe2_FRI	8:00-22:00	[時:分]-[時:分]	—	—		
40	OPE2_土曜日	String	airOpeSch / SPEC_ACOpe2_SAT	0:00-0:00	[時:分]-[時:分]	—	—		
41	OPE2_祝日	String	airOpeSch / SPEC_ACOpe2_HOL	0:00-0:00	[時:分]-[時:分]	—	—		
42	OPE2_特別日	String	airOpeSch / SPEC_ACOpe2_SPC	0:00-0:00	[時:分]-[時:分]	—	—		
43	■OPE3/中間期■								
44	OPE3_日曜日	String	airOpeSch / SPEC_ACOpe3_SUN	0:00-0:00	[時:分]-[時:分]	—	—		
45	OPE3_月曜日	String	airOpeSch / SPEC_ACOpe3_MON	8:00-22:00	[時:分]-[時:分]	—	—		
46	OPE3_火曜日	String	airOpeSch / SPEC_ACOpe3_TUE	8:00-22:00	[時:分]-[時:分]	—	—		
47	OPE3_水曜日	String	airOpeSch / SPEC_ACOpe3_WED	8:00-22:00	[時:分]-[時:分]	—	—		
48	OPE3_木曜日	String	airOpeSch / SPEC_ACOpe3_THU	8:00-22:00	[時:分]-[時:分]	—	—		
49	OPE3_金曜日	String	airOpeSch / SPEC_ACOpe3_FRI	8:00-22:00	[時:分]-[時:分]	—	—		
50	OPE3_土曜日	String	airOpeSch / SPEC_ACOpe3_SAT	0:00-0:00	[時:分]-[時:分]	—	—		
51	OPE3_祝日	String	airOpeSch / SPEC_ACOpe3_HOL	0:00-0:00	[時:分]-[時:分]	—	—		
52	OPE3_特別日	String	airOpeSch / SPEC_ACOpe3_SPC	0:00-0:00	[時:分]-[時:分]	—	—		
53	■記録・グラフ表示■								
54	グラフを表示する	Boolean	isGVisible	FALSE	[-]	—	—		←グラフを表示するときはチェックする
55	最大同時表示ステップ数	int	maxItemCount	100	[-]	—	—		←グラフに同時表示する最大ステップ数

56	記録を有効とする	Boolean	isRecord	FALSE	[-]	―	―		←記録を有効とするときはチェックする

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する
2	発停	L1_swcIn	SWC	-	制御	On/Off 信号	入口	上位からの発停信号
3	発停	L1_swcOut	SWC	-	制御	On/Off 信号	出口	DR 信号を含めた発停信号を出力する
4	雨	L0_eleObs	ELE	-	状態	水	観察	電力デマンドを観察する
5	日射	L0_airObsOA	AIR、OA	-	状態	空気	観察	外気を観察する

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
2	swcIn	発停信号	−	入口
3	swcOut	発停信号	−	出口
4	観察外気乾球温度	温度	℃	入口
5	観察電力デマンド	電力	W	入口

（7）計算フロー・計算内容・入力方法

・DR の発動・終了は次の項目を設定できる。

① 設定値

DR 制御の発令は、ユーザーが入力する外気温度 DB あるいは受電電力量をもとにしたレベル判断設定値を超えた場合に、それぞれのレベルを発信する。

設定値は、OPE1、OPE2、OPE3 に分けてセットできる。

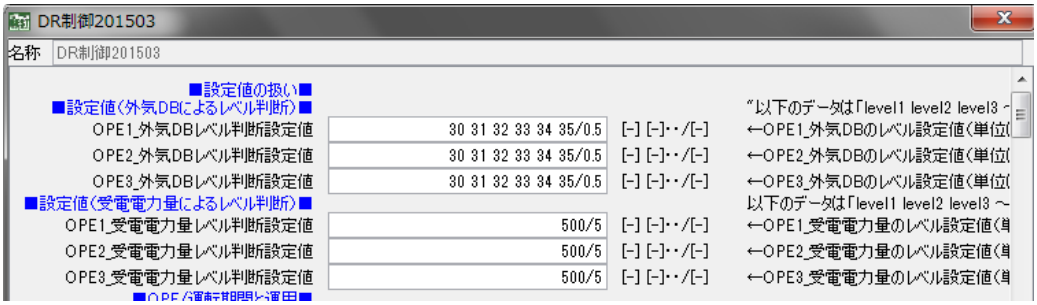
外気 DB のレベル判断設定値と受電電力量のレベル判断設定値を、半角スペース文字で区切った数値リストとしてセットする。

下図の入力例では、OPE1 から OPE3 が同じ設定となっており、外気 DB のレベル 1 が 30℃、レベル 2 が 31℃、34℃がレベル 5 である。その次の 35℃はレベル 6 ということになるが、制御に有効なレベルは 1 ～ 5 までの 5 段階である。34℃を超えた場合はすべてレベル 5 として扱う。

リストの最後はレベル移行時の動作隙間の値を半角”/”文字で区切って入力する。

動作隙間は全レベルに共通とし、下図の入力例では 0.5℃である。

図 1-1　DR 制御の設定値（外気 DB・受電電力量レベル判断設定値）



② 運転期間、運用

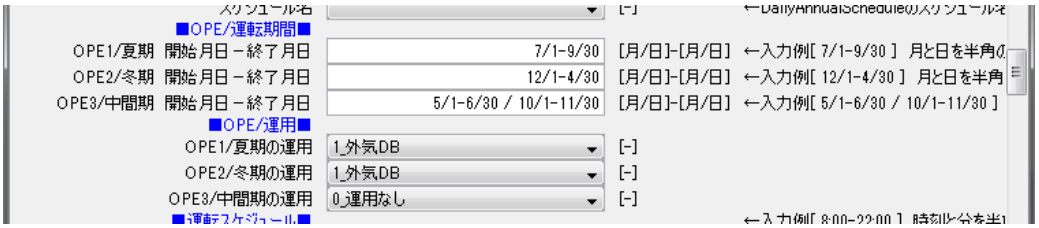
OPE1、OPE2、OPE3　の 3 期間を DR 発動対象期間として設定できる。

それぞれ　開始月日と終了月日をユーザーが入力する。

期間が重なった場合は、OPE1→OPE2→OPE3　の順で設定された運用を適用する。

OPE1 から OPE3 の運用を「0_運用なし、1_外気 DB、2_受電電力量、3_外気 DB ＋受電電力量」の 4 種類から選択する。

図 1-2　DR 制御　OPE 別の期間と運用の設定



③ 曜日別の時間帯

OPE1 から OPE3 のそれぞれの曜日別に、発動対象とする時間帯を設定できる。

１日の時間帯は、複数に分かれていても対応可能である。下図の OPE2 のように ”/”（半角のスペース＋/＋半角のスペースの 3 文字）で区切って入力する。

図 1-3　DR 制御曜日別の時間帯

（８）DR 制御の子機側（対応機器）の設定方法

８.1.　発電機の対応・・・レベルに応じて発電機の出力を変える

発電機の DR 制御は、発電機用の台数制御モジュールで対応する。この発電機用台数制御モジュールは、発電あるいは排熱需要に対して複数台の発電機の適切な運転台数を指示するものであり、DR レベルに応じて発電出力を変えるという運転制御に対応できる。

DR 制御を実施する場合、制御方式は「出力一定制御」に限定する。

発電機台数制御の入力画面に追加した DR 制御項目について説明する。

・OPE＊_DR 制御を実施する（チェックボックス）

OPE1～OPE3 の期間別に DE 制御を実施するかしないかを設定できる。

実施する場合はその期間のチェックボックスをチェックする。実施しない場合はチェックをはずす。

・OPE＊_DR 制御時の設定発電出力リスト

DR レベル別に発電出力を設定しておき、DR 発動時にはそのレベルの発電出力一定で運転するように運転台数を調整する。

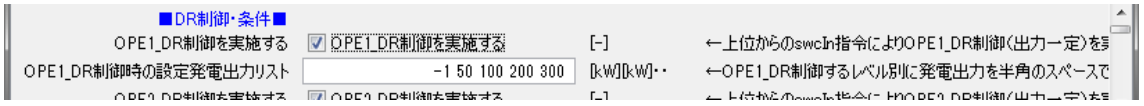
発電出力リストは、レベル１、レベル２・・の順番で半角のスペース文字で区切って入力する。

下図の例では、DR レベル 1 の時は 0kW、レベル 3 は 100kW、レベル 5 は 300kW の発電を行うこととなる。

図 8.1-1　発電機の DR 制御入力例 1

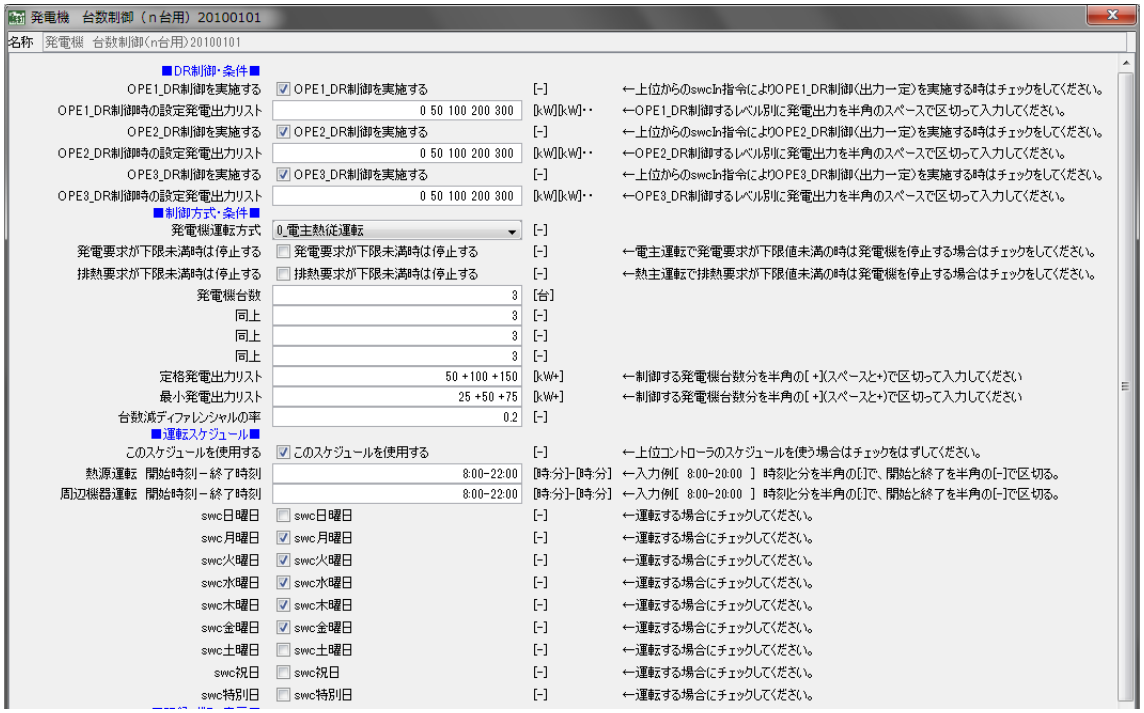
DR レベル 1 では通常制御（発電機運転方式に指定された制御方式）をしたい場合は、そのレベルの値を負の値としておく。

図 8.1-2　発電機の DR 制御入力例 2



運転台数調整からあとの手順は本来の発電機台数制御モジュールの動作となる。

・発電機台数制御（n 台用）モジュールの入力画面

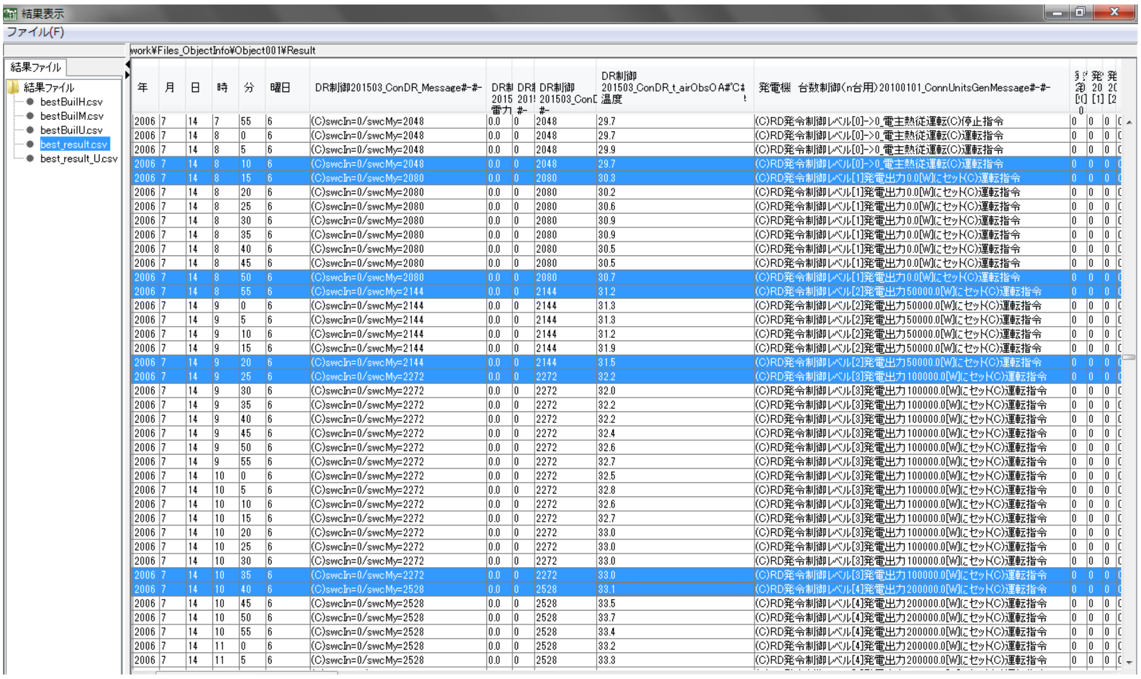


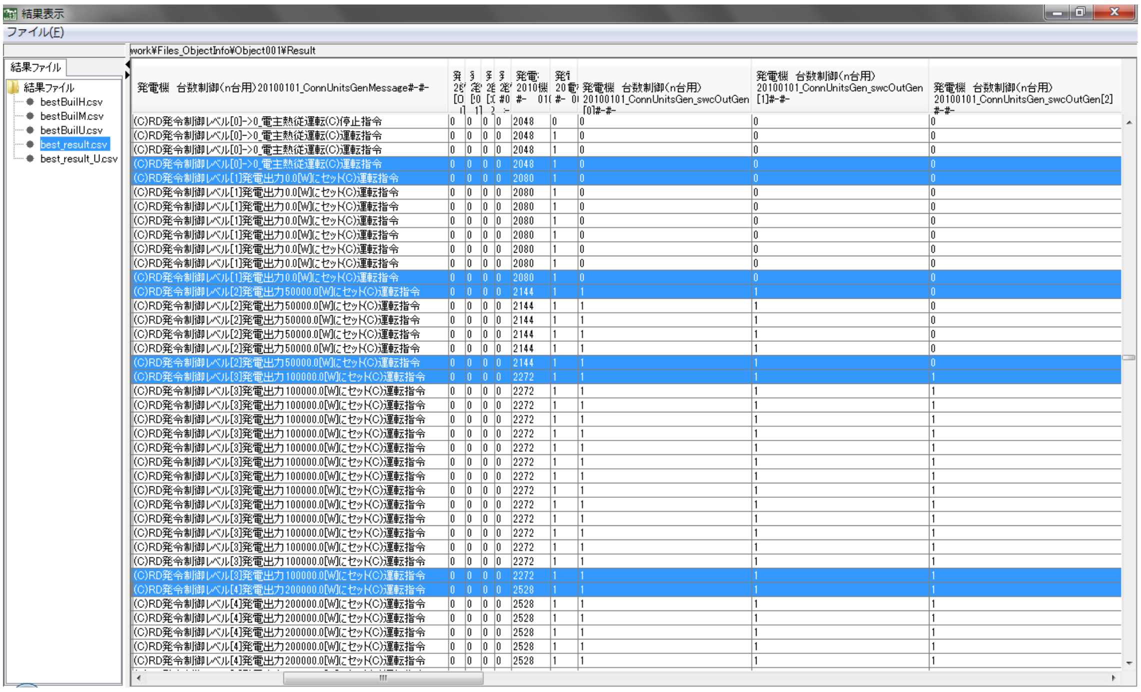
☞ここの運転スケジュールは DR 制御ではなく通常の台数制御に適用されるものである

・発電機台数制御（n 台用）モジュールのシーケンス接続画面　　3 台の例

・発電機 DR 制御の動作例（計算実行例）

入力条件は、これまでに説明した DR 制御（親）と発電機台数制御（DR 制御（子））の図 8.1-1 発電機の DR 制御入力例 1 で示したものである。

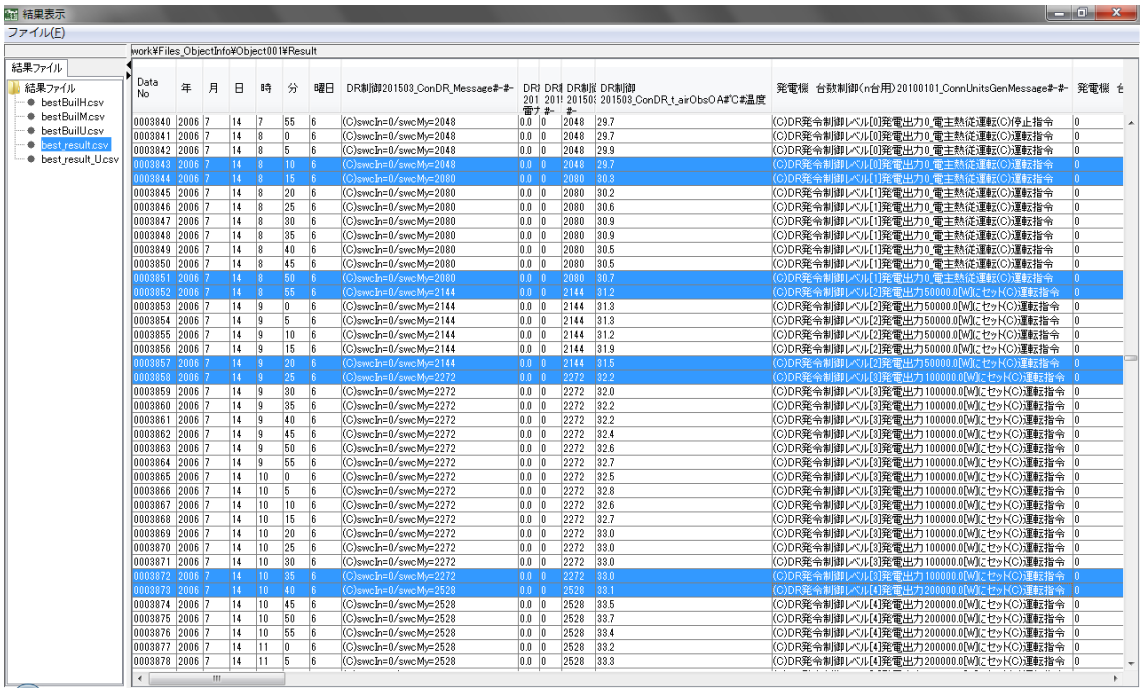




7 月 14 日の午前の制御結果の例で、設定した外気 DB によるレベル設定判断値により、レベル[0]（DR 発動なし）→レベル[1]0kW →レベル[2]50kW →レベル[3]100kW →レベル[4]200kW　と発動し、それに応じて運転台数を制御している様子がわかる。

＊）発電機の台数制御は、下限発電容量でまず運転台数が決まる。

次は、図 8.1-2 発電機の DR 制御入力例 2 で示したもので DR レベル 1 を負の値とし、通常制御とした場合である。7/14 の 8 時 15 分から 8 時 50 分までレベル 1 が発令されているが通常制御の電主熱従運転となる。



8.2.　ゾーンの対応・・・レベルに応じて空調時の室温目標値を変える

ゾーンの DR 制御は空調時の室温目標値を変えることで対応し、空調機用の制御モジュールの DR 制御機能を使用する。

空調機用制御モジュールは、PID 制御目標値（年間固定値あるいは 12 個の月別目標値）を BestValue 媒体クラスとして送信する接続ノード（例えば L0_valOutSP1_ConOpe）を備えている。（→図 8.2-1、図 8.2-2）

DR 制御の室温目標値をこれらの接続ノードから送ることで、ゾーン側の各種空調方式の室温目標値を変更することが可能である。

図 8.2-1　空調機制御モジュールの入力画面・・・SP1、SP2、SP3_設定値

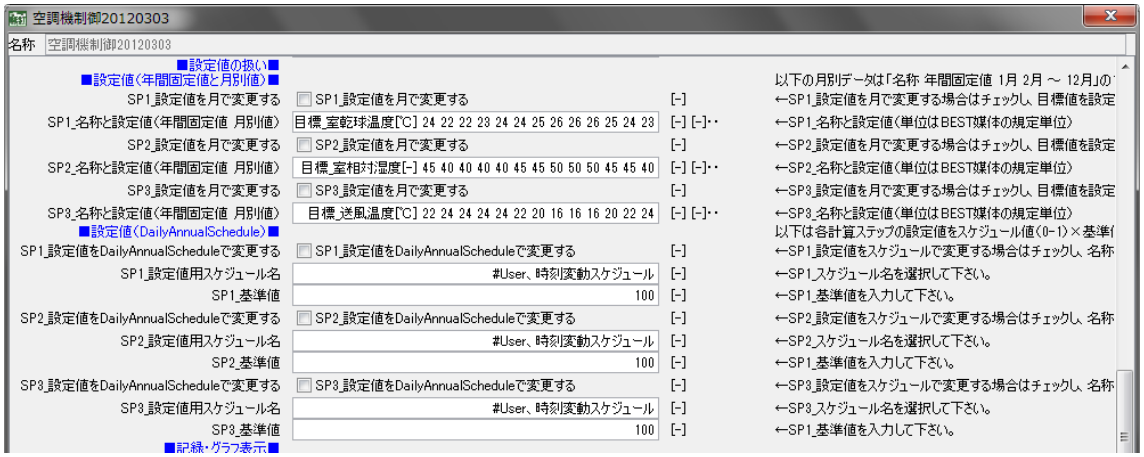
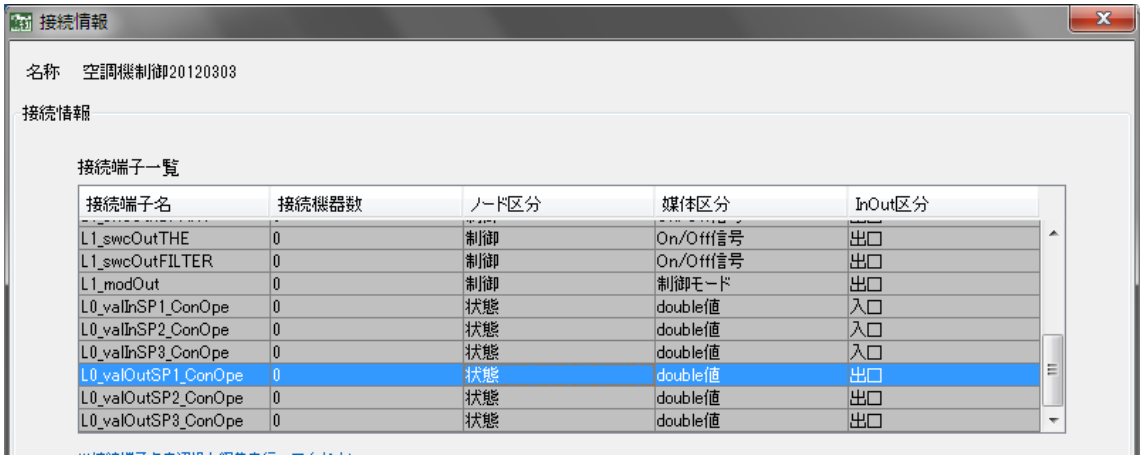


図 8.2-2　空調機制御モジュールのシーケンス接続画面



☞　L0_valOutSP1_ConOpe 接続ノードからは、通常制御時は「SP1_名称と設定値（年間固定値 月別値)」で入力された値が送信され、DR 制御時は「OPE1_DR 制御時の設定温度リスト」の値が DR レベルに応じて送信される。

同様に L0_valOutSP2_ConOpe 接続ノードからは「SP2_名称と設定値（年間固定値 月別値)」または「OPE2_DR 制御時の設定温度リスト」、L0_valOutSP3_ConOpe 接続ノードからは「SP3_名称と設定値（年間固定値 月別値)」または「OPE3_DR 制御時の設定温度リスト」の値が送信される。

L0_valOutSP＊_ConOpe 接続ノードの接続先は、対象とするゾーン側空調機器の運転容量を操作する PID 制御モジュールの「L0_valInConOpe」接続ノードに接続しておく。

主なゾーン側の空調方式については次のような制御の流れとなる。

① BM、PAC 室内機・・・室内機容量制御用 PID 制御モジュールへ室温目標値を送信
② FF 式暖房機・・・室内機容量制御用 PID 制御モジュールへ室温目標値を送信
③ FCU・・・FCU 冷温水コイル二方弁用 PID 制御モジュールへ RA 目標値を送信
④ VAV ユニット・・・ユニット風量制御用 PID 制御モジュールへ室温目標値を送信
⑤ 放射空調・・・放射パネル二方弁用 PID 制御モジュールへ制御目標値を送信

空調機制御の入力画面の DR 制御関連の項目について説明する。

・OPE＊_DR 制御を実施する（チェックボックス）

OPE1～OPE3 の期間別に DE 制御を実施するかしないかを設定できる。

実施する場合はその期間のチェックボックスをチェックする。実施しない場合はチェックをはずす。

・OPE＊_DR 制御時の設定温度リスト

DR レベル別に温度を設定しておき、DR 発動時にはそのレベルの設定温度を指定されたノードから送信する。

温度リストは、レベル１、レベル２・・の順番で半角のスペース文字で区切って入力する。

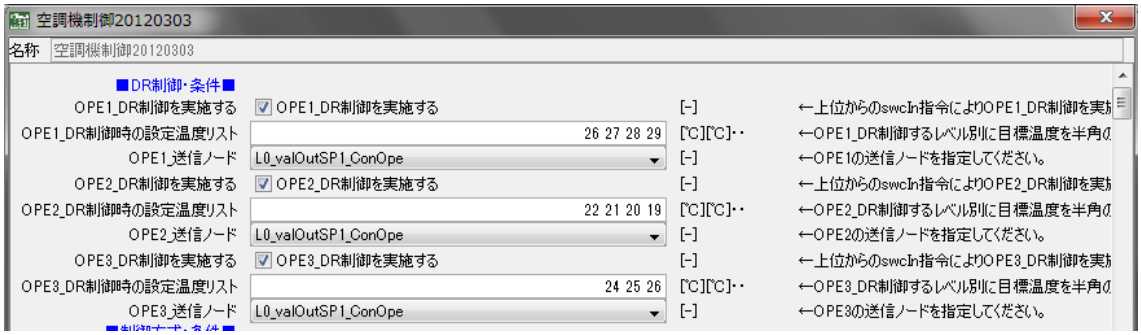
下図の例の OPE1 では、DR レベル 1 の時は 26℃、レベル 3 は 28℃、レベル 4 は 29℃の設定温度を送信することとなる。

・OPE＊_送信ノード

DR 制御時に設定値を送信に使用する接続ノードを指定する。

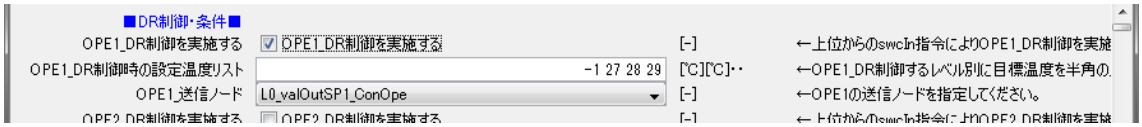
下図の例の OPE1 では､L0_valOutSP1_ConOpe 接続ノードから送信することになる。

図 8.2-3　空調機制御の DR 制御入力例 1



DR レベル 1 では通常制御の送信をしたい場合は、そのレベルの値を負の値としておく。

図 8.2-4　空調機制御の DR 制御入力例 2

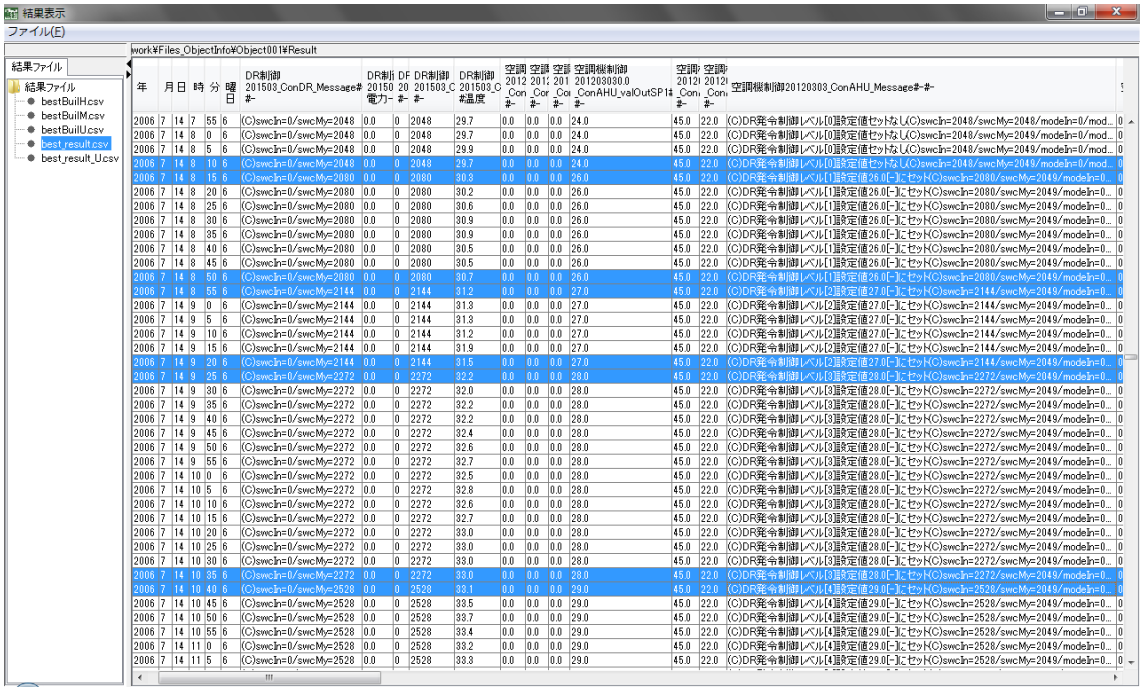


・空調機制御モジュールの入力画面

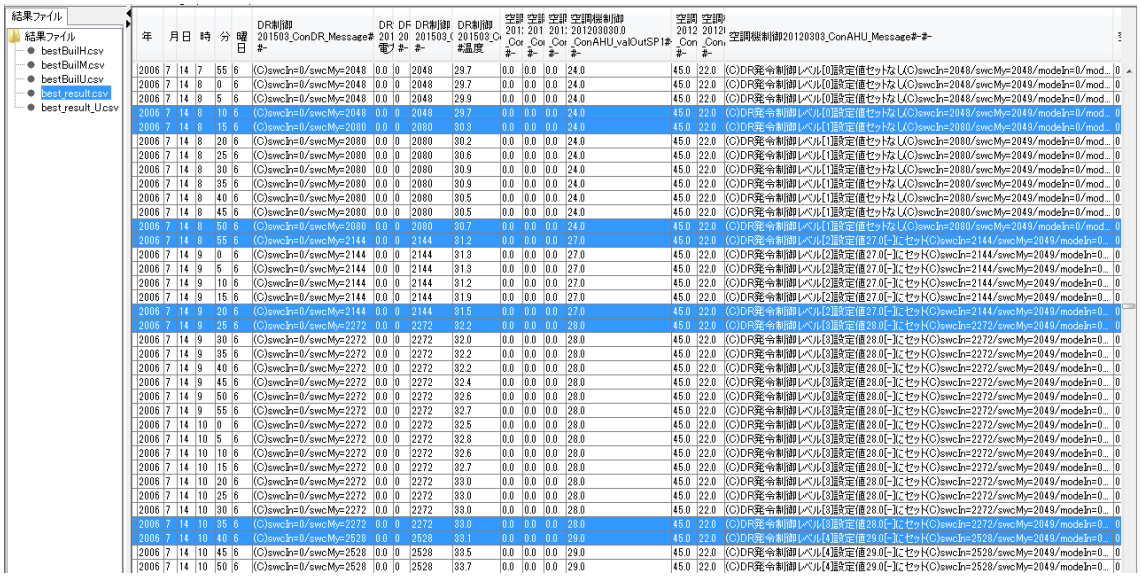
・空調機 DR 制御の動作の例（計算実行例）

入力条件は、ここまでに説明した DR 制御（親）と空調機制御（DR 制御（子））の図 8.2-3 空調機の DR 制御入力例 1 で示したものである。

7 月 14 日の午前の制御結果の例で、設定した外気 DB によるレベル設定判断値により、レベル[0]（DR 発動なし）→レベル[1]26℃ →レベル[2]27℃ →レベル[3]28℃ →レベル[4]29℃　と送信している様子がわかる。



次は、図 8.2-4 空調機の DR 制御入力例 2 で示したもので DR レベル 1 を負の値とし、通常制御とした場合である。7/14 の 8 時 15 分から 8 時 50 分までレベル 1 が発令されているが通常制御時の 24℃が送信される。



8.3. 空調機の対応・・・レベルに応じて送風温度目標値を変える、間欠送風する

空調機の DR 制御は、レベルに応じて送風温度等を変える方法と、間欠送風することで対応する。

それぞれ次のような制御の流れとなる。

① CAV 方式空調機・・・冷温水コイル二方弁用 PID 制御モジュールへ RA 目標値を送信
② VAV 方式空調機・・・冷温水コイル二方弁用 PID 制御モジュールへ SA 目標値を送信
③ 間欠送風（今後対応の予定）

送風温度等を変える方法に対しては 8.2 ゾーンの対応と同じく空調機用の制御モジュールの DR 制御機能が利用できる。

8.4. OA チャンバーの対応・・・レベルに応じて CO2 濃度制御目標値を変える

OA チャンバーの DR 制御は、レベルに応じて導入外気量を CO2 濃度制御目標値を変える方法と、間欠送風することで対応する。

次のような制御の流れとなる。

① 外気量制御・・・CO2 濃度制御用 PID 制御モジュールへ RA の CO2 濃度目標値を送信
② 間欠送風（今後対応の予定）

8.5.　熱源の対応・・・レベルに応じて出口温度の目標値を変える、発停運転をする

熱源の DR 制御は出口水温の目標値を変えるあるいは発停運転することで対応し、熱源用の制御モジュールの DR 制御機能を使用する。

熱源制御モジュールは、熱源出口水温目標値（年間固定値あるいは 12 個の月別目標値）を BestValue 媒体クラスとして送信する接続ノード（L0_valOutSP_T_watOutCH_HS）を備えている。(→図 8.5-1、図 8.5-2)

DR 制御の熱源出口温度目標値をこの接続ノードから送ることで、各種熱源の出口温度目標値を変更することが可能である。

図 8.5-1　熱源制御モジュールの入力画面・・・出口水温、入口水温など

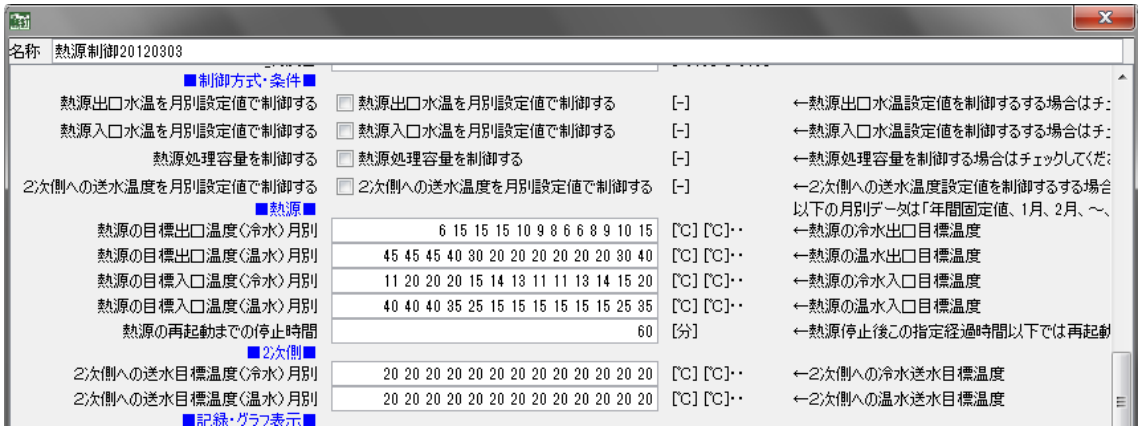
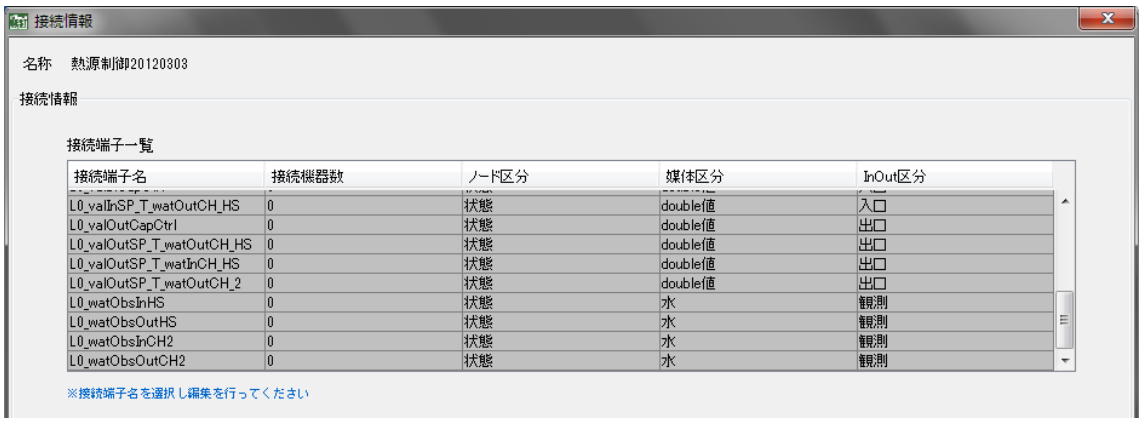


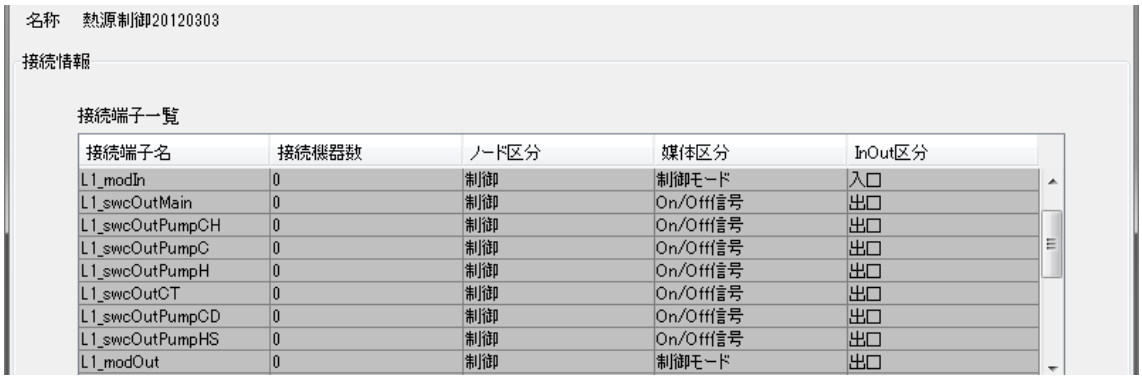
図 8.5-2　熱源制御モジュールのシーケンス接続画面　設定温度の送信ノード_valOutSP_



☞　L0_valOutSP_T_watOutCH_HS 接続ノードからは、通常制御時は「熱源の目標出口温度（冷水）月別」と「熱源の目標出口温度（温水）月別」で入力された値が送信され、DR 制御時は「OPE＊_DR 制御時の設定熱源出口温度リスト」の値が DR レベルに応じて送信される。

発停運転による DR 対応の場合は、DR レベルに応じて設定されている OnOff 条件により swcOut***ノードの信号を強制的に上書きして送信する。

図 8.5-3　熱源制御モジュールのシーケンス接続画面　発停の送信ノード_swcOut＊＊＊



接続端子名	接続機器数	ノード区分	媒体区分	InOut区分
L1_modIn	0	制御	制御モード	入口
L1_swcOutMain	0	制御	On/Off信号	出口
L1_swcOutPumpCH	0	制御	On/Off信号	出口
L1_swcOutPumpC	0	制御	On/Off信号	出口
L1_swcOutPumpH	0	制御	On/Off信号	出口
L1_swcOutCT	0	制御	On/Off信号	出口
L1_swcOutPumpCD	0	制御	On/Off信号	出口
L1_swcOutPumpHS	0	制御	On/Off信号	出口
L1_modOut	0	制御	制御モード	出口

☞　DR 制御で「停止」の場合 L1_swcOutMain から L1_swcOutPumpHS まですべて停止で上書きする。DR 制御で「運転」の場合は、L1_swcOutMain、L1_swcOutPumpCH 、L1_swcOutPumpHS は modIn（冷暖）の状態に関係なく運転で上書きし、modIn が冷房の時は L1_swcOutPumpC 、L1_swcOutCT 、L1_swcOutPumpCD が運転、modIng が暖房の時は L1_swcOutPumpH を運転で上書きして送信する。

空調機制御の入力画面の DR 制御機能の項目について説明する。

・OPE＊_DR 制御を実施する（チェックボックス）

OPE1～OPE3 の期間別に DE 制御を実施するかしないかを設定できる。

実施する場合はその期間のチェックボックスをチェックする。実施しない場合はチェックをはずす。

・OPE＊_DR 制御時の設定熱源出口温度リスト

DR レベル別に温度を設定しておき、DR 発動時にはそのレベルの設定熱源出口温度を L0_valOutSP_T_watOutCH_HS 接続ノードから送信する。

温度リストは、レベル１、レベル２・・の順番で半角のスペース文字で区切って入力する。

下図の例の OPE1 では、DR レベル 1 の時は 7℃、レベル 3 は 9℃、レベル 4 は 10℃の設定温度を送信することとなる。

・OPE＊_DR 制御時の設定二次側送水温度リスト

DR レベル別に温度を設定しておき、DR 発動時にはそのレベルの設定二次側送水温度を L0_valOutSP_T_watOutCH_2 接続ノードから送信する。

温度リストは、レベル 1、レベル 2・・の順番で半角のスペース文字で区切って入力する。

下図の例の OPE1 では、DR レベル 1 の時は 7℃、レベル 3 は 9℃、レベル 4 は 10℃の設定温度を送信することとなる。

・OPE＊_DR 制御時の設定発停リスト

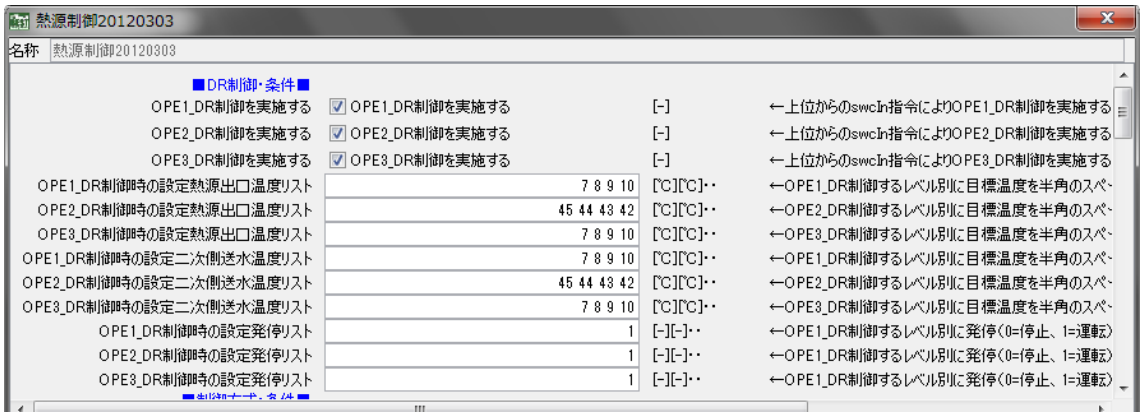
DR レベル別に発停条件を設定しておき、DR 発動時にはそのレベルの設定発停信号を L1_swcOut＊＊＊接続ノードから送信する。

発停リストは、レベル 1、レベル 2・・の順番で半角のスペース文字で区切って入力する。

その値は、運転=1、停止=0 とする。

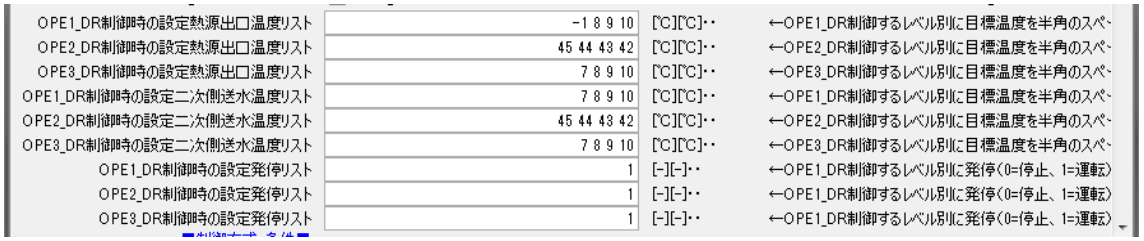
下図の例の OPE1 では、DR レベル 1 で運転（DR レベルに関係なく運転）を送信することとなる。

図 8.5-3　熱源制御の DR 制御入力例 1



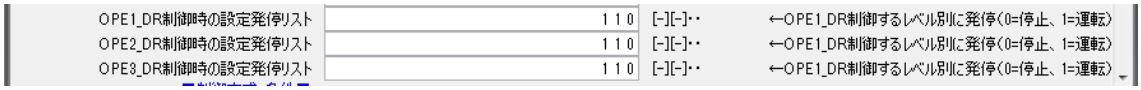
DR レベル 1 では通常制御の熱源出口水温送信をしたい場合は、そのレベルの値を負の値としておく。

図 8.5-4　熱源制御の DR 制御入力例 2



下図は熱源の発停を DR レベルにより設定したもので、レベル 3 以上で停止させるものである。

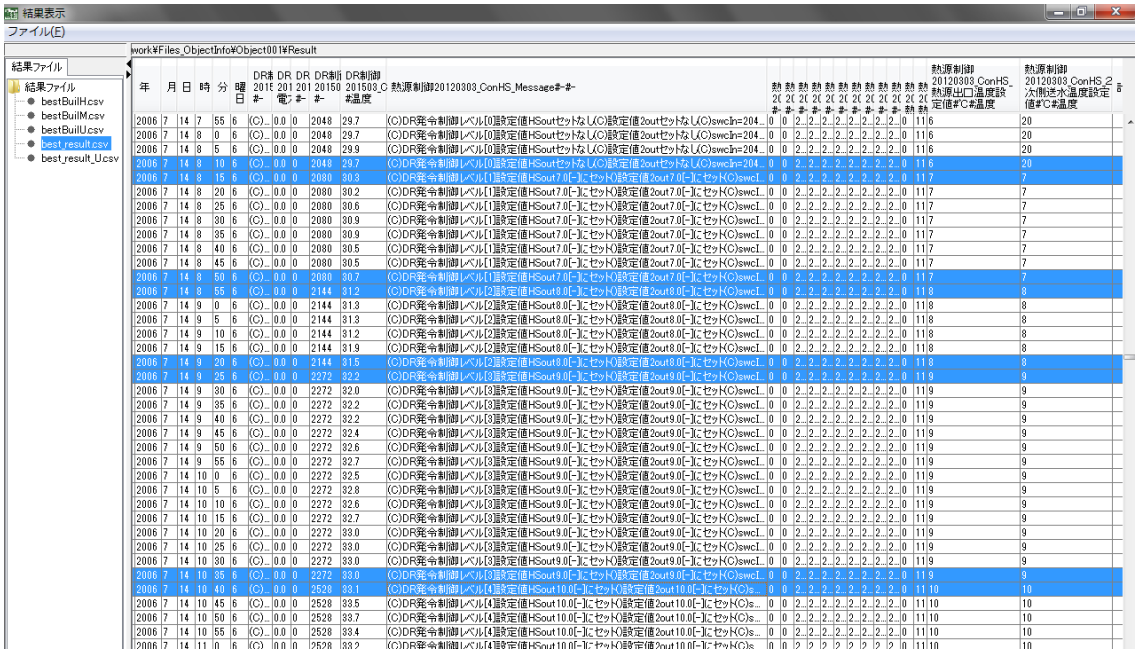
図 8.5-5　熱源制御の DR 制御入力例 3



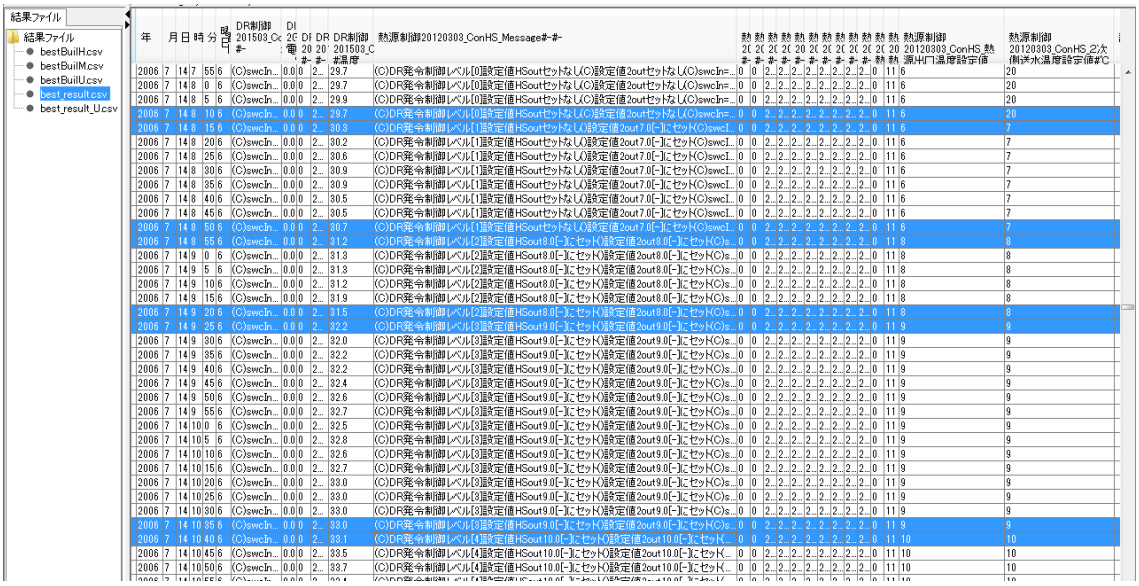
・熱源 DR 制御の動作の例（計算実行例）

入力条件は、ここまでに説明した DR 制御（親）と熱源制御（DR 制御（子））の図 8.5-3 熱源制御の DR 制御入力例 1 で示したものである。

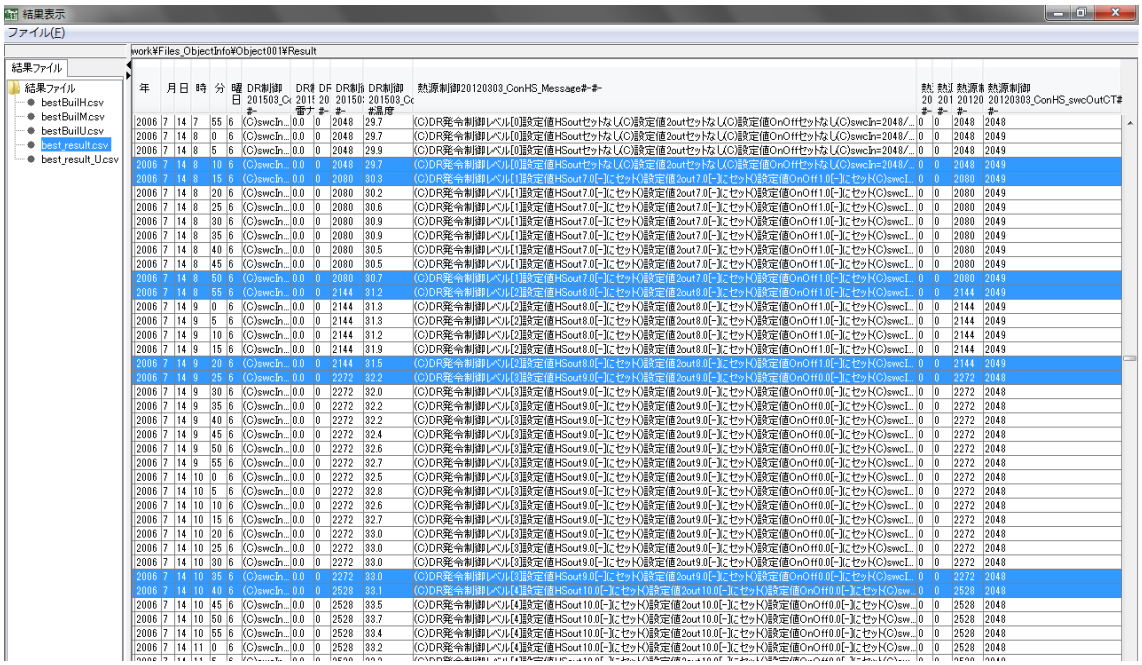
7 月 14 日の午前の制御結果の例で、設定した外気 DB によるレベル設定判断値により熱源出口水温・二次側送水温度は、レベル[0]（DR 発動なし）→レベル[1]7℃・7℃ →レベル[2]8℃・8℃ →レベル[3]9℃・9℃ →レベル[4]10℃・10℃　と送信している様子がわかる。

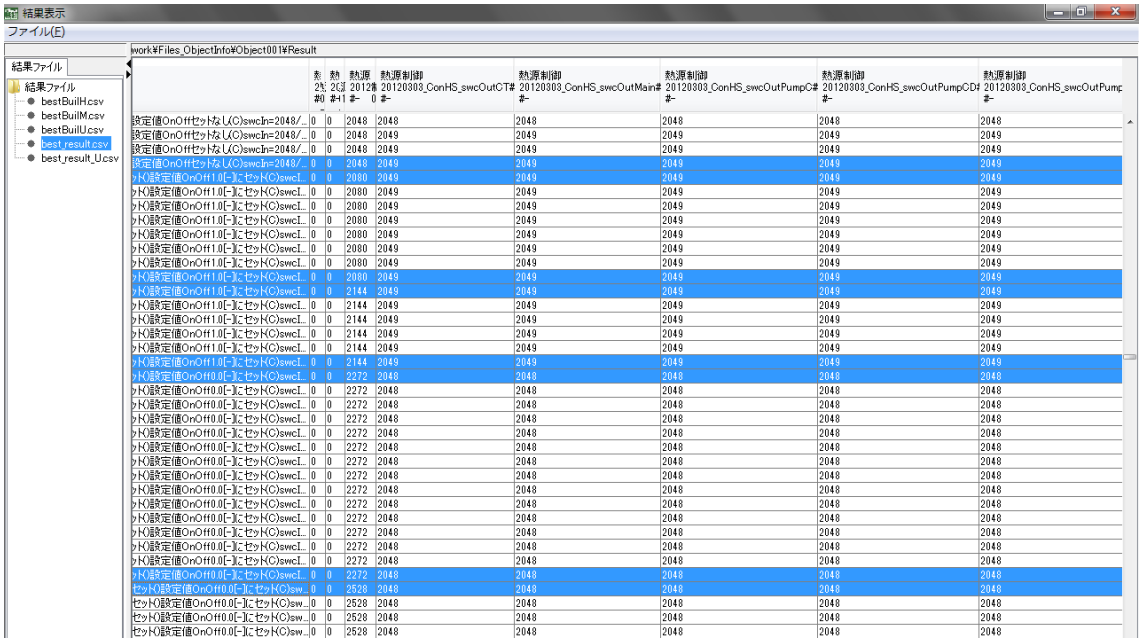


次は、図 8.5-4 熱源制御の DR 制御入力例 2 で示したもので、熱源出口水温の DR レベル 1 を負の値とし通常制御とした場合である。7/14 の 8 時 15 分から 8 時 50 分までレベル 1 が発令されているが通常制御時の 6℃が送信される。



次は、図 8.5-5 熱源制御の DR 制御入力例 3 で示したもので、熱源の発停を DR レベル 3 で「停止」とした場合である。7/14 の 9 時 25 分からレベル 3 が 10 時 40 分からレベル 4 が発令されていずれも停止信号が送信される。





☞ swcOut***の結果の数値は Airswc クラスの制御信号定数を合成したものである。

この場合 2049 は運転、2048 は停止信号を表す。（奇数は運転、偶数は停止）

名称　　：空調機制御 2015（場所：設備 2015／制御機器 2015）

モジュール名　：ControlAHUModule2015、ControlAHUSpec2015、ControlAHUCalc2015

(1) **入力画面**

図 1　空調機制御モジュールの入力画面

(2) **モジュールの概要**

本モジュールは、空調機の発停やシーズンなどのタイムスケジュールを制御するものである。「ControlAHUModule2015.java」「ControlAHUSpec2015.java」「ControlAHUCalc2015.java」を組み合わせて、一つのモジュールを構成している。「airswc・airmode」

特徴は以下の通りである。

- 運転期間とそれぞれの運用（運転モード）の設定を分けて入力可能。
- 期間未指定日は換気運転が可能。
- 上位コントローラからの modIn ノードの冷暖シーズンの使い分けを指定可能。
  →「この冷暖房期間を使用する」にチェックを入れると、ここで指定した冷暖房期間を適用。チェックがない場合は、modIn ノードに接続された上位コントローラからの mod 信号の冷暖期間を適用。
- CO_2 濃度制御のための指定を追加。
  →CO_2 濃度制御する場合は、「CO_2 制御する」にチェック。
- PID 制御モジュールと連携して目標温度等の設定値を月別に変更可能。
- 共通で作成したスケジュールが使用可能。
  →「Daily Annual Schedule を使用する」にチェックを入れ、スケジュール名に共通で作成したスケジュールの名前を入力（日スケジュールと接続）
- DR 制御のレベル別に温度設定を変更可能。

(3) **モジュールの入出力**

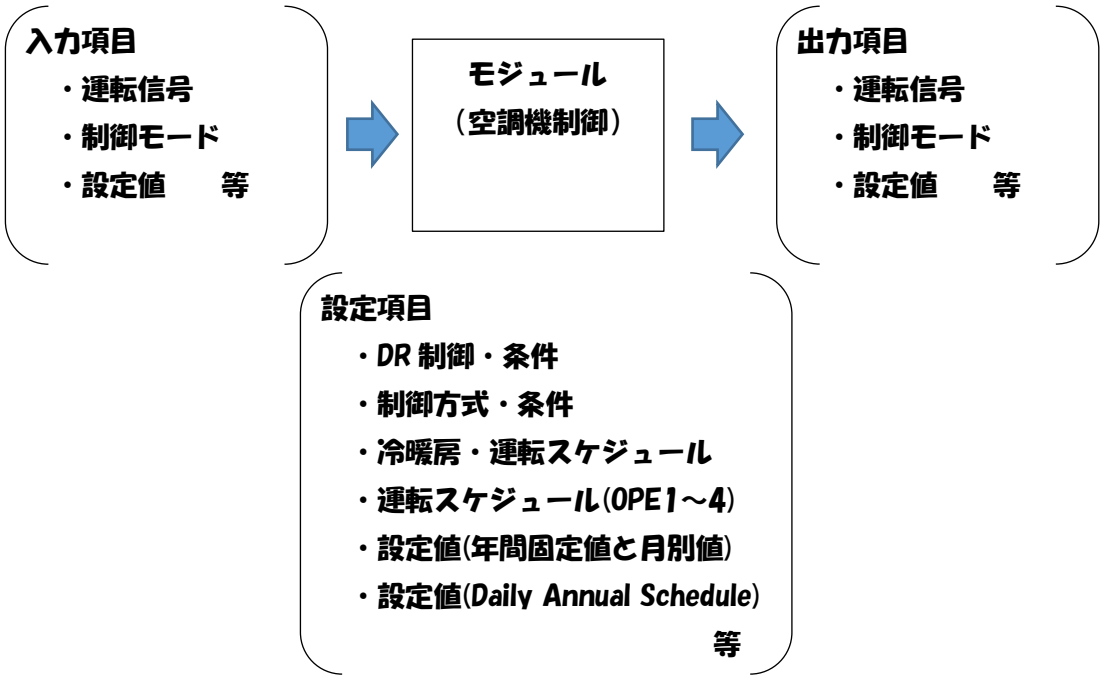


図 2　空調機制御モジュール

(4) スペック入力項目（図 2 における設定項目）

表 1　Duct 分岐モジュールのスペック入力項目

NO.	項目	型・クラス名	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
	DR 制御・条件								
0	OPE1_DR 制御を実施する	boolean		FALSE	[-]			△	上位からの swcIn 指令により OPE1_DR 制御を実施する時はチェック
1	OPE1_DR 制御時の設定温度リスト	String		26 27 28 29	[℃][℃]・・			△	OPE1_DR 制御するレベル別に目標温度を半角のスペースで区切って入力
2	OPE1_送信ノード	String		L0_valOutSP1_ConOpe	[-]			△	OPE1 の送信ノードを指定
3	OPE2_DR 制御を実施する	boolean		FALSE	[-]			△	上位からの swcIn 指令により OPE2_DR 制御を実施する時はチェック
4	OPE2_DR 制御時の設定温度リスト	String		22 21 20 19	[℃][℃]・・			△	OPE2_DR 制御するレベル別に目標温度を半角のスペースで区切って入力
5	OPE2_送信ノード	String		L0_valOutSP1_ConOpe	[-]			△	OPE2 の送信ノードを指定
6	OPE3_DR 制御を実施する	boolean		FALSE	[-]			△	上位からの swcIn 指令により OPE3_DR 制御を実施する時はチェック
7	OPE3_DR 制御時の設定温度リスト	String		24 25 26	[℃][℃]・・			△	OPE3_DR 制御するレベル別に目標温度を半角のスペースで区切って入力
8	OPE3_送信ノード	String		L0_valOutSP1_ConOpe	[-]			△	OPE3 の送信ノードを指定
	制御方式・条件								
9	このスケジュールを使用する	boolean		TRUE	[-]			◎	上位コントローラのスケジュールを使う場合はチェックをはずす
10	この冷暖房期間を使用する	boolean		TRUE	[-]			◎	この冷暖房期間を使用する場合はチェック
11	CO2 制御する	boolean		FALSE	[-]			○	CO2 制御する場合はチェック
12	DailyAnnualSchedule を使用する	boolean		FALSE	[-]			△	DailyAnnualSchedule を使用する場合はチェック
13	スケジュール名	String			[-]			△	DailyAnnualSchedule のスケジュール名を入力
	冷暖房・運転スケジュール								
14	OPE1/夏期　開始月日－終了月日	String		6/1-9/30	[月/日]-[月/日]			○	
15	OPE2/冬期　開始月日－終了月日	String		12/1-3/31	[月/日]-[月/日]			○	
16	OPE3/中間期　開始月日－終了月日	String		4/1-5/31 / 10/1-11/30	[月/日]-[月/日]			○	
17	OPE1/夏期の運用	String		1_冷房	[-]			○	
18	OPE2/冬期の運用	String		2_暖房	[-]			○	
19	OPE3/中間期の運用	String		0_停止	[-]			○	

	OPE4/未指定日の運用	String		0_停止	[-]			○	
	運転スケジュール								
	OPE1/夏期								
20	OPE1_日曜日	String		0:00-0:00	[時:分]-[時:分]			○	複数の on/off 時間は 半角のスペース＋[/]＋スペースの３文字[/]で区切る。
21	OPE1_月曜日	String		8:00-22:00	[時:分]-[時:分]			○	
22	OPE1_火曜日	String		8:00-22:00	[時:分]-[時:分]			○	
23	OPE1_水曜日	String		8:00-22:00	[時:分]-[時:分]			○	
24	OPE1_木曜日	String		8:00-22:00	[時:分]-[時:分]			○	
25	OPE1_金曜日	String		8:00-22:00	[時:分]-[時:分]			○	
26	OPE1_土曜日	String		0:00-0:00	[時:分]-[時:分]			○	
27	OPE1_祝日	String		0:00-0:00	[時:分]-[時:分]			○	
28	OPE1_特別日	String		0:00-0:00	[時:分]-[時:分]			○	
29	OPE1_外気カット時間	String		0:00-0:00	[分間]			○	起動開始時刻から外気カットを何分間行うかを入力
	OPE2/冬期								
30	OPE2_日曜日	String		0:00-0:00	[時:分]-[時:分]			○	
31	OPE2_月曜日	String		8:00-22:00	[時:分]-[時:分]			○	
32	OPE2_火曜日	String		8:00-22:00	[時:分]-[時:分]			○	
33	OPE2_水曜日	String		8:00-22:00	[時:分]-[時:分]			○	
34	OPE2_木曜日	String		8:00-22:00	[時:分]-[時:分]			○	
35	OPE2_金曜日	String		8:00-22:00	[時:分]-[時:分]			○	
36	OPE2_土曜日	String		0:00-0:00	[時:分]-[時:分]			○	
37	OPE2_祝日	String		0:00-0:00	[時:分]-[時:分]			○	
38	OPE2_特別日	String		0:00-0:00	[時:分]-[時:分]			○	
39	OPE2_外気カット時間	String		0:00-0:00	[分間]			○	
	OPE3/中間期、OPE4								
40	OPE3_日曜日	String		0:00-0:00	[時:分]-[時:分]			○	
41	OPE3_月曜日	String		8:00-22:00	[時:分]-[時:分]			○	
42	OPE3_火曜日	String		8:00-22:00	[時:分]-[時:分]			○	
43	OPE3_水曜日	String		8:00-22:00	[時:分]-[時:分]			○	
44	OPE3_木曜日	String		8:00-22:00	[時:分]-[時:分]			○	
45	OPE3_金曜日	String		8:00-22:00	[時:分]-[時:分]			○	
46	OPE3_土曜日	String		0:00-0:00	[時:分]-[時:分]			○	
47	OPE3_祝日	String		0:00-0:00	[時:分]-[時:分]			○	

48	OPE3_特別日	String		0:00-0:00	[時:分]-[時:分]			○	
49	OPE3_外気カット時間	String		0:00-0:00	[分間]			○	
	設定値の扱い								
	設定値（年間固定値と月別値）								
50	SP1_設定値を月で変更する	boolean		FALSE	[-]			○	SP1_設定値を月で変更する場合はチェック
51	SP1_名称と設定値（年間固定値 月別値）	String		目標_室乾球温度[℃] 24 22 22 23 24 24 25 26 26 26 25 24 23	[-] [-]・・			○	「名称 年間固定値 1 月 2 月 ～ 12 月」の 14 個を半角スペースで区切る（単位は BEST 媒体の規定単位）
52	SP2_設定値を月で変更する	boolean		FALSE	[-]			○	
53	SP2_名称と設定値（年間固定値 月別値）	String		目標_室相対湿度[-] 45 40 40 40 40 45 45 50 50 50 45 45 40	[-] [-]・・			○	
54	SP3_設定値を月で変更する	boolean		FALSE	[-]			○	
55	SP3_名称と設定値（年間固定値 月別値）	String		目標_送風温度[℃] 22 24 24 24 24 22 20 16 16 16 20 22 24	[-] [-]・・			○	
	設定値（DailyAnnualSchedule）								各計算ステップの設定値をスケジュール値（0~1）×基準値
56	SP1_設定値を DailyAnnualSchedule で変更する	boolean		FALSE				△	SP1_設定値をスケジュールで変更する場合はチェックし、名称と基準値を設定
57	SP1_設定値用スケジュール名	String						△	SP1_スケジュール名を選択
58	SP1_基準値	double		100				△	SP1_基準値を入力
59	SP2_設定値を DailyAnnualSchedule で変更する	boolean		FALSE				△	
60	SP2_設定値用スケジュール名	String						△	
61	SP2_基準値	double		100				△	
62	SP3_設定値を DailyAnnualSchedule で変更する	boolean		FALSE				△	
63	SP3_設定値用スケジュール名	String						△	
64	SP3_基準値	double		100				△	
	記録・グラフ表示								
65	グラフを表示する	boolean		FALSE	[-]			○	グラフを表示するときはチェック
66	最大同時表示ステップ数	int		100	[-]			○	グラフに同時表示する最大ステップ数を入力
67	記録を有効とする	boolean		FALSE	[-]			○	このモジュールの記録を有効とするときはチェック

(5) シーケンス接続（図 2 における入力項目、出力項目）

表 2　Duct 分岐モジュールのシーケンス接続項目

No.	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
0	記録	L2_recOut	REC		記録	メモリ	出口	空調記録モジュールへ発停信号等を出力する。
1	上位からの発停信号	L1_swcIn	Airswc		制御	On/Off 信号	入口	上位モジュールからの On/Off 信号を受け取る。
2	上位からの空調モード	L1_modIn	Airmod		制御	制御モード	入口	上位モジュールから「冷房、暖房、冷暖房、換気、外気カット、CO2 制御」のモード信号を受け取る
3	発停信号	L1_swcOut			制御	On/Off 信号	出口	On/Off 信号を出力する。
4	外気導入制御	L1_swcOutOA			制御	On/Off 信号	出口	外気導入の On/Off 信号を出力する
5	SA ファン運転制御	L1_swcOutFANSA			制御	On/Off 信号	出口	SA ファンの On/Off 信号を出力する
6	RA ファン運転制御	L1_swcOutFANRA			制御	On/Off 信号	出口	RA ファンの On/Off 信号を出力する
7	EA ファン運転制御	L1_swcOutFANEA			制御	On/Off 信号	出口	EA ファンの On/Off 信号を出力する
8	OA ファン運転制御	L1_swcOutFANOA			制御	On/Off 信号	出口	OA ファンの On/Off 信号を出力する
9	冷温水コイル運転制御	L1_swcOutCOIL			制御	On/Off 信号	出口	冷温水コイルの On/Off 信号を出力する
10	冷水コイル運転制御	L1_swcOutCOILC			制御	On/Off 信号	出口	冷水コイルの On/Off 信号を出力する
11	温水コイル運転制御	L1_swcOutCOILH			制御	On/Off 信号	出口	温水コイルの On/Off 信号を出力する
12	加湿器運転制御	L1_swcOutSPRAY			制御	On/Off 信号	出口	加湿器の On/Off 信号を出力する
13	熱交換器運転制御	L1_swcOutTHE			制御	On/Off 信号	出口	熱交換器の On/Off 信号を出力する
14	フィルタ運転制御	L1_swcOutFILTER			制御	On/Off 信号	出口	フィルタの On/Off 信号を出力する
15								
16	空調モード	L1_modOut			制御	制御モード	出口	「冷房、暖房、冷暖房、換気、外気カット、CO2 制御」のモード信号を出力する
17	上位からの設定値 1	L0_valInSP1_ConOpe			状態	double 値	入口	他の制御モジュールから設定値 1（SP1）を受けとる
18	上位からの設定値 2	L0_valInSP2_ConOpe			状態	double 値	入口	他の制御モジュールから設定値 2（SP2）を受けとる
19	上位からの設定値 3	L0_valInSP3_ConOpe			状態	double 値	入口	他の制御モジュールから設定値 3（SP3）を受けとる
20	設定値 1 出力	L0_valOutSP1_ConOpe			状態	double 値	出口	設定値 1（SP1）を出力する
21	設定値 2 出力	L0_valOutSP2_ConOpe			状態	double 値	出口	設定値 2（SP2）を出力する
22	設定値 3 出力	L0_valOutSP3_ConOpe			状態	double 値	出口	設定値 3（SP3）を出力する

(6) **記録項目**

表 3　Duct 分岐モジュールの記録項目

No.	項目	出力内容	単位	分類
0	ConAHU_Message	メッセージ	-	メッセージ
1	ConAHU_swcIn	On/Off 信号	-	入口
2	ConAHU_modIn	On/Off 信号	-	入口
3	ConAHU_modOut	On/Off 信号	-	出口
4	ConAHU_swcOut	On/Off 信号	-	出口
5	ConAHU_swcOutOA	On/Off 信号	-	出口
6	ConAHU_swcOutFANSA	On/Off 信号	-	出口
7	ConAHU_swcOutFANRA	On/Off 信号	-	出口
8	ConAHU_swcOutFANEA	On/Off 信号	-	出口
9	ConAHU_swcOutFANOA	On/Off 信号	-	出口
10	ConAHU_swcOutCOIL	On/Off 信号	-	出口
11	ConAHU_swcOutSPRAY	On/Off 信号	-	出口
12	ConAHU_swcOutTHE	On/Off 信号	-	出口
13	ConAHU_swcOutFILTER	On/Off 信号	-	出口
14	ConAHU_valInSP1	設定値	-	入口
15	ConAHU_valInSP2	設定値	-	入口
16	ConAHU_valInSP3	設定値	-	入口
17	ConAHU_valOutSP1	設定値	-	出口
18	ConAHU_valOutSP2	設定値	-	出口
19	ConAHU_valOutSP3	設定値	-	出口

(7) **計算方法**

ユーザーの入力画面の運転スケジュールの設定条件により、出口側の swc および mod 制御信号の値を設定し送信する。計算時刻の条件が冷房期間から外気カットまでの組合せの時の、出口側の swc および mod 制御信号の状態値を構成する Airswc 定数と Airmod 定数を表している。

☞ 主な Airswc 定数と Airmod 定数は次の通りである

Airswc 定数		Airmod 定数	
OFF	=0	COOL	＝1
ON	=1	HEAT	＝2
OPE1	=2048	OACUT	＝1024
OPE2	=4096	VENTILATE	=16384
OPE3	=8192	CO2	
		FC	

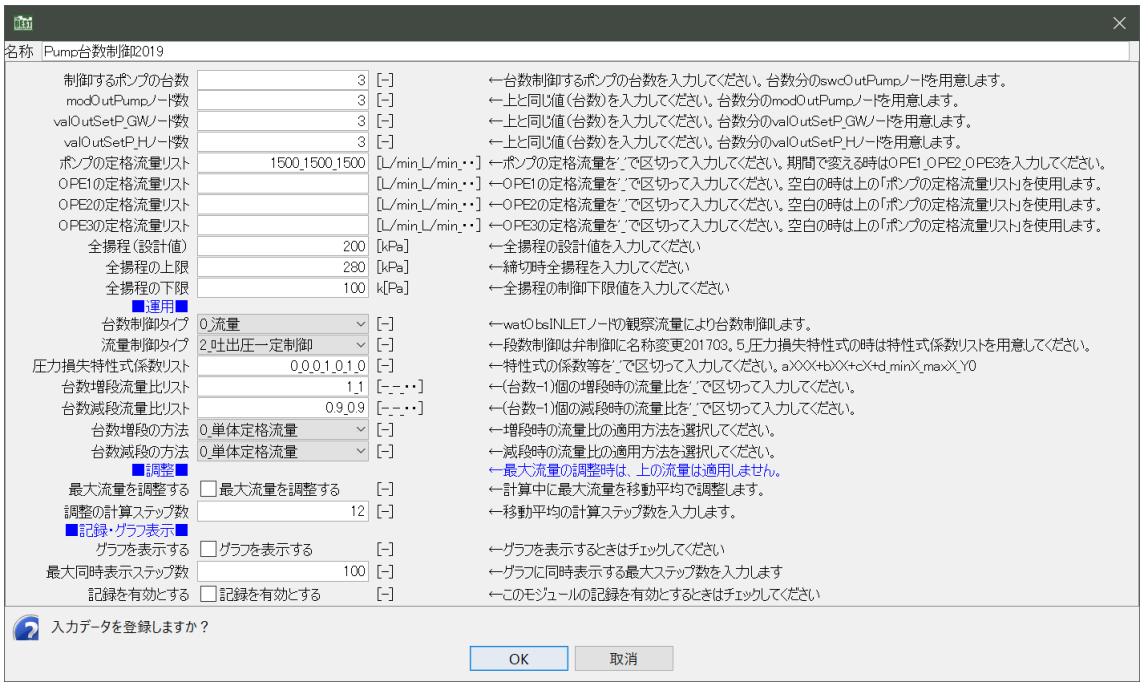
☞ 例えば、swcOut の構成 Airswc 定数が ON と OPE1 の場合、swcOut ノードから送信される状態値は、ON+OPE1=1＋2048＝2049 となる。
modOut についても swcOut と同様で、構成 Airmod 定数が COOL と OACUT の場合、modOut ノードから送信される状態値は、COOL+OACUT＝1＋1024＝1025 となる。

「Pump 台数制御 2019」（場所：設備 2015／制御機器 2015）

モジュール名	Pump 台数制御 2019
クラス	ControlnOpeFPPump2019

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、複数のポンプからなるポンプ群に対して必要流量に応じた運転台数を制御するモジュールである。通常、ポンプ群の入口流量を観察し、運転台数を算出する。

個々のポンプへは、発停信号、運転モード信号、運転流量および運転揚程を渡す。

運転揚程は、指定された流量制御方式に関連付けられた圧力損失特性から算出する。

前のポンプ台数制御モジュール（ControlnUnitsPumpFPOperatingModule20111111）から次の点を改良した。

・台数切替えの増段と減段の流量比率をそれぞれ設定できるように改良した。*1

・増段と減段の方法を 2 種類用意した。*1

・流量制御方式に圧力損失の特性式（ポンプ群の流量負荷率の 3 次式）を追加した。*1

・流量負荷率を OPE1～OPE3 の期間ごとの最大流量に対して算定していたものを、全期間の最大流量に対して算定するよう変更した。

☞ *1 の機能改良は、SHASE のシミュレーション評価ガイドラインのトライアルのポンプ台数制御の計算条件に対応したものです。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

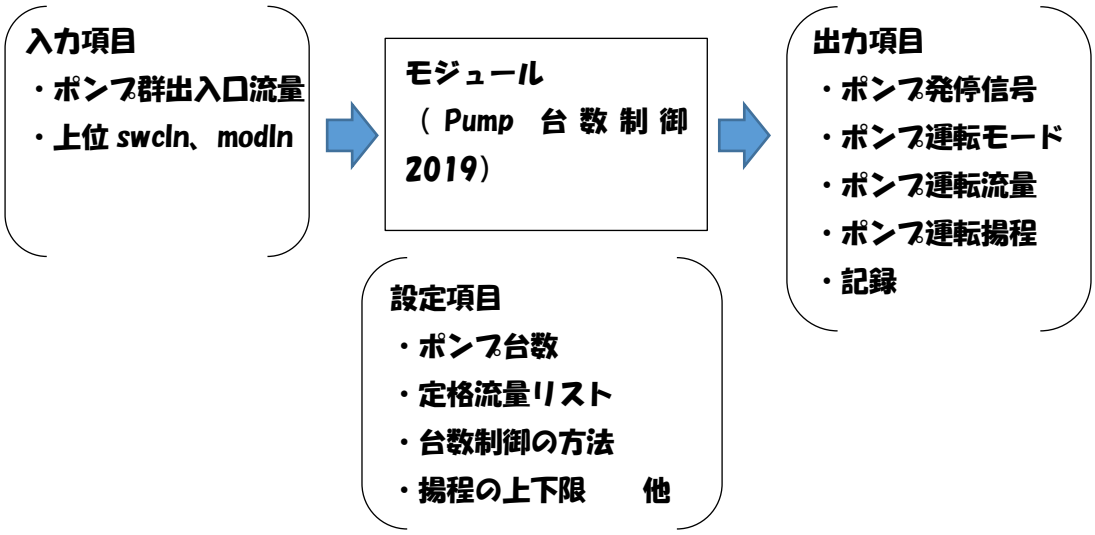


図 1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name			―	―		
1	制御するポンプの台数	Int	swcOUtPump[]	3	[-]	―	―		←台数制御するポンプの台数を入力してください。台数分の swcOutPump ノードを用意します。
2	modOutPump ノード数	Int	modOutPump[]	3	[-]	―	―		←上と同じ値（台数）を入力してください。台数分の modOutPump ノードを用意します。
3	valOutSetP_GW ノード数	Int	valOutSetP_GW[]	3	[-]	―	―		←上と同じ値（台数）を入力してください。台数分の valOutSetP_GW ノードを用意します。
4	valOutSetP_H ノード数	Int	valOutSetP_H[]	3	[-]	―	―		←上と同じ値（台数）を入力してください。台数分の valOutSetP_H ノードを用意します。
5	ポンプの定格流量リスト	String	SPEC_M_Pump	1500_1500_1500	[L/min_L/min_・・]	―	―		←ポンプの定格流量を'_'で区切って入力してください。期間で変える時は OPE1_OPE2_OPE3 を入力してください。
6	OPE1 の定格流量リスト	String	SPEC_M_OPE1_Pump		[L/min_L/min_・・]	―	―		←OPE1 の定格流量を'_'で区切って入力してください。空白の時は上の「ポンプの定格流量リスト」を使用します。
7	OPE2 の定格流量リスト	String	SPEC_M_OPE2_Pump		[L/min_L/min_・・]	―	―		←OPE2 の定格流量を'_'で

									区切って入力してください。空白の時は上の「ポンプの定格流量リスト」を使用します。
8	OPE3 の定格流量リスト	String	SPEC_M_OPE3_Pump		[L/min_L/min_・・]	—	—		←OPE3 の定格流量を'_'で区切って入力してください。空白の時は上の「ポンプの定格流量リスト」を使用します。
9	■運用■								
10	台数制御タイプ	String	SPEC_conType	0_流量、 X_台数制御なし	[-]	—	—		←watObsINLET ノードの観察流量により台数制御します。
11	流量制御タイプ	String	SPEC_FlowconType	0_定流量、 1_弁制御、 1_段数制御、 2_吐出圧一定制御、 3_末端差圧一定制御、 4_予想末端差圧制御、 5_圧力損失特性式	[-]	—	—		←段数制御は弁制御に名称変更 201703。5_圧力損失特性式の時は特性式係数リストを用意してください。
12	圧力損失特性式係数リスト	String	SPEC_CoefficientPLossFunction	0_0_0_1_0_1_0	[-]	—	—		←特性式の係数等を'_'で区切って入力してください。aXXX+bXX+cX+d_minX_maxX_Y0
13	全揚程（設計値）	Double	SPEC_Design_H	200	[kPa]	—	0		←全揚程の設計値を入力してください
14	全揚程の上限	Double	SPEC_Upper_H	280	[kPa]	—	0		←締切時全揚程を入力してください
15	全揚程の下限	Double	SPEC_Lower_H	100	k[Pa]	—	0		←全揚程の制御下限値を入力してください
16	台数増段流量比リスト	String	SPEC_r_Increase	1_1	[-_-_・・]	—	—		←(台数-1)個の増段時の流

									量比を'_'で区切って入力してください。
17	台数減段流量比リスト	String	SPEC_r_Decrease	0.9_0.9	[-_-_・・]	―	―		←(台数-1)個の減段時の流量比を'_'で区切って入力してください。
18	台数増段の方法	String	SPEC_DTypeIncrease	0_単体定格流量、 1_合計定格流量	[-]	―	―		←増段時の流量比の適用方法を選択してください。
19	台数減段の方法	String	SPEC_DTypeDecrease	0_単体定格流量、 1_合計定格流量	[-]	―	―		←減段時の流量比の適用方法を選択してください。
20	■調整■								←最大流量の調整時は、上の流量は適用しません。
21	最大流量を調整する	Boolean	SPEC_isAdjust	FALSE	[-]	―	―		←計算中に最大流量を移動平均で調整します。
22	調整の計算ステップ数	Int	SPEC_NumAdjustSteps	12	[-]	―	―		←移動平均の計算ステップ数を入力します。
23	■記録・グラフ表示■								
24	グラフを表示する	Boolean	SPEC_isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
25	最大同時表示ステップ数	Int	SPEC_maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
26	記録を有効とする	Boolean	SPEC_isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください
27	全記録を有効とする	Boolean	SPEC_isRecordALL	FALSE	[-]	―	―		←このモジュールの全記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する
2	発停	L1_swcIn	SWC	-	制御	On/Off 信号	入口	上位からの発停信号
3	運転モード	L1_modIn	MOD	-	制御	mod 信号	入口	上位からの mod 信号
4	入口の水の観察	LO_watObsINLET	WAT	-	状態	水	観察	ポンプ群への wat 入口を観察する
5	出口の水の観察	LO_watObsOUTLET	WAT	-	状態	水	観察	ポンプ群からの wat 出口を観察する
6	発停	L1_swcOutPump[0] ・・ L1_swcOutPump[n-1]	SWC	-	制御	On/Off 信号	出口	各ポンプ、配管分岐、配管集合などへの発停信号 ポンプ、配管分岐、配管集合などの L1_swcIn へ接続する
7	モード	L1_modOutPump[0] ・・ L1_modOutPump[n-1]	MOD	-	制御	Mod 信号	出口	各ポンプ、配管分岐、配管集合などへの mod 信号 ポンプ、配管分岐、配管集合などの L1_modIn へ接続する
8	運転流量[g/s]	LO_valOutSetP_GW[0] ・・ LO_valOutSetP_GW[n-1]	VAL	g/s	値	値	出口	各ポンプへの運転流量[g/s]　LO_valInSetP_GW へ接続する 配管分岐の分岐流量[g/s]　LO_valInMwatOut[]へ接続する
9	運転揚程[Pa]	LO_valOutSetP_H[0] ・・ LO_valOutSetP_H[n-1]	VAL	Pa	値	値	出口	各ポンプへの運転揚程[Pa]　LO_valInSetP_H へ接続する

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ ConnPumps_Message#-#-	メッセージ	−	メッセージ
2	swcIn ConnPumps_swcIn#-#-	発停信号	−	入口
3	modIn ConnPumps_modeIn#-#-	モード信号	−	入口
4	ConnPumps_ポンプ群入口流量#g/s#質量流量	質量流量	g/s	入口
5	ConnPumps_ポンプ群出口流量#g/s#質量流量	質量流量	g/s	出口
6	ConnPumps_ポンプ群入口温度#℃#温度	温度	℃	入口
7	ConnPumps_ポンプ群出口温度#℃#温度	温度	℃	出口
8	ConnPumps_swcOutPump[＊]#-#-	発停信号（ポンプの台数分）	−	出口
9	ConnPumps_modOutPump[＊]#-#-	モード信号（ポンプの台数分）	−	出口
10	ConnPumps_valOutSetP_GW[＊]#g/s#質量流量	運転流量（ポンプの台数分）	g/s	出口
11	ConnPumps_valOutSetP_H[＊]#Pa#揚程	運転揚程（ポンプの台数分）	Pa	出口
12	ConnPumps_ポンプ群調整最大流量#g/s#質量流量	質量流量	g/s	
13	ConnPumps_ポンプ群調整ステップ平均流量#g/s#質量流量	質量流量	g/s	

（7）計算フロー・計算内容・入力方法

(7-1)入力項目の説明

① 制御台数、定格流量リスト、揚程

- **制御するポンプの台数、modOutPump ノード数、valOutSetP_GW ノード数、valOutSetP_H ノード数**

制御するポンプの台数	3	[-]
modOutPumpノード数	3	[-]
valOutSetP_GWノード数	3	[-]
valOutSetP_Hノード数	3	[-]

このモジュールで制御するポンプ群のポンプ台数を入力する。すべて同じ値を入力します。

これらの入力値により各ポンプを制御するための接続ノードを UI が自動で用意します。UI が用意するのは swcOutPump、modOutPump、valOutSetP_GW、valOutSetP_H の 4 種類の接続ノードです。

☞ 制御するポンプの台数を 3 と入力した場合、UI はポンプ毎に 4 種類の接続ノードを次のような名称で用意します。

ポンプ 1 : L1_swcOutPump[0] L1_modOutPump[0] L0_valOutSetP_GW[0] L0_valOutSetP_H[0]

ポンプ 2 : L1_swcOutPump[1] L1_modOutPump[1] L0_valOutSetP_GW[1] L0_valOutSetP_H[1]

ポンプ 3 : L1_swcOutPump[2] L1_modOutPump[2] L0_valOutSetP_GW[2] L0_valOutSetP_H[2]

上の例のように接続ノードの最後に通し番号[＊]が付け足されます。

この通し番号は[0]から始まります。台数が n 台の時、[n-1]で終わります。

運転の優先順番は接続ノードの通し番号の順です。

上の例では、[0]ポンプ 1→[1]ポンプ 2→[2]ポンプ 3 という優先運転順位となります。

- **ポンプの定格流量リスト、OPE1 定格流量リスト、OPE2 定格流量リスト、OPE3 定格流量リスト**

ポンプの優先運転の順に、制御する各ポンプの定格流量[L/min]を半角のアンダースコア[_]または半角スペース[ ]で区切って並べます。

ポンプの定格流量リスト	1500_1500_1500	[L/min_L/min_‥]
OPE1の定格流量リスト		[L/min_L/min_‥]
OPE2の定格流量リスト		[L/min_L/min_‥]
OPE3の定格流量リスト		[L/min_L/min_‥]

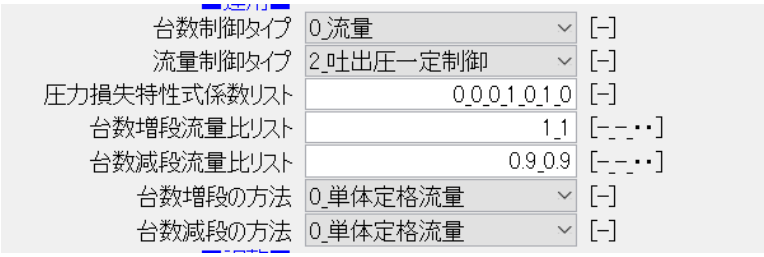
「制御するポンプの台数」で入力した数の定格流量データを入力します。
入力例：ポンプ 3 台の定格流量が 300L/min、400L/min、500L/min の場合、
[300_400_500]あるいは[300 400 500]と入力します。
台数制御するポンプ群の送水系の圧力損失を算出する時に、この「ポンプの定格流量リスト」の合計流量を設計最大流量とみなします。
ポンプ群の流量負荷率は次の式となります。
ポンプ群の流量負荷率[-]＝負荷流量[g/s]／ポンプ群の設計最大流量[g/s]
設計最大流量[g/s]＝ポンプの定格流量リストの流量合計[L/min]／0.06
「OPE1 定格流量リスト」、「OPE2 定格流量リスト」、「OPE3 定格流量リスト」は、期間別に適用するリストです。期間別に定格流量リストを作成することができます。入力方法は「ポンプの定格流量リスト」と同じです。これらが空白の場合は、「ポンプの定格流量リスト」で入力したリストを使います。
OPE1 から OPE3 の定格流量リストの流量は、ポンプの定格流量リストと同じ値にする必要はありません。OPE＊期間での運転流量を OPE＊期間の定格流量として入力します。

- **全揚程（設計値）**
  ポンプ群の全揚程の設計値を入力します。
  入力例：設計全揚程が 200kPa の時[200]と入力します。

- **全揚程の上限**
  ポンプの全揚程の制御上限値を入力します。
  ポンプの全揚程の上限値は、定格条件を設定するとポンプモジュールの特性によって自動できまるものですが、台数制御モジュールではその情報を取得できないこと、異容量の複数のポンプの場合には共通の上限値を設定する必要があること、ポンプ特性の上限値より小さく制御上限値を設定したい場合があること、に対応しています。
  入力例：全揚程の上限が 280kPa の時[280]と入力します。

- **全揚程の下限**
  ポンプの全揚程の制御下限値を入力します。
  入力例：全揚程の下限が 100kPa の時[100]と入力します。

② 運用



・ **台数制御タイプ**

台数制御タイプの選択肢は 0_流量、X_台数制御なし　です。

接続ノードの watObsINLET（ポンプ群入口）の観察流量により台数制御をおこないます。

・ **流量制御タイプ**

流量制御タイプの選択肢は次の 5 種類です。

0_定流量

1_弁制御、1_段数制御

2_吐出圧一定制御

3_末端差圧一定制御

4_予想末端差圧制御

5_圧力損失特性式

下図は各制御方法の流量と揚程の運転ポイントの変化を表しています。

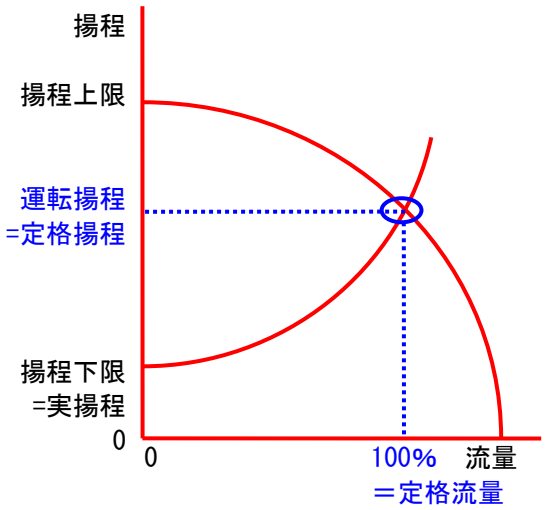


図 3.4.0 定流量

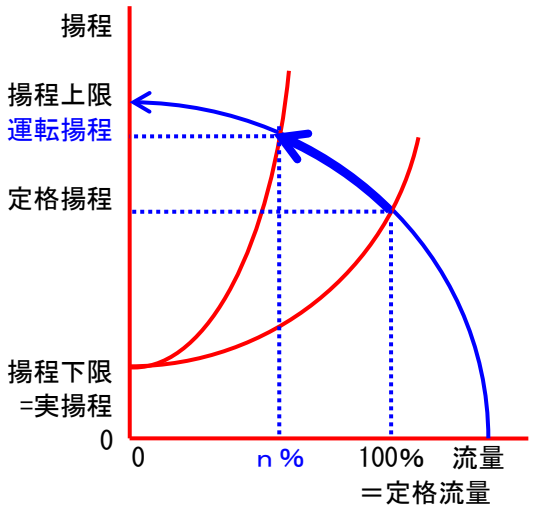


図 3.4.1 弁制御、段数制御

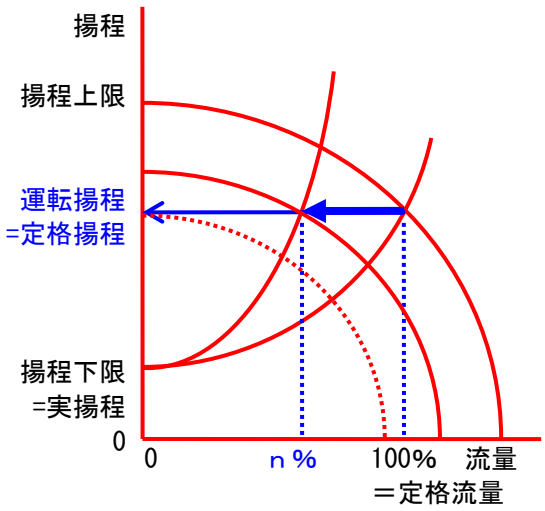


図 3.4.2 吐出圧一定制御

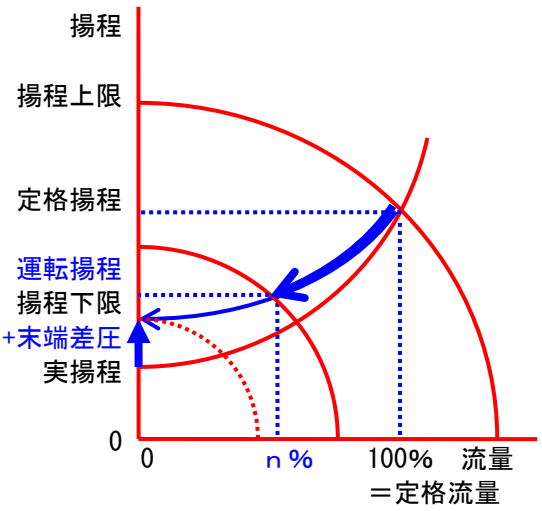


図 3.4.3 末端差圧一定制御

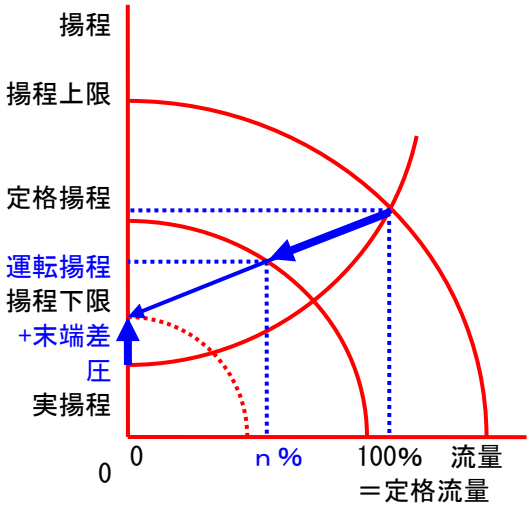


図 3.4.4 予想末端差圧制御

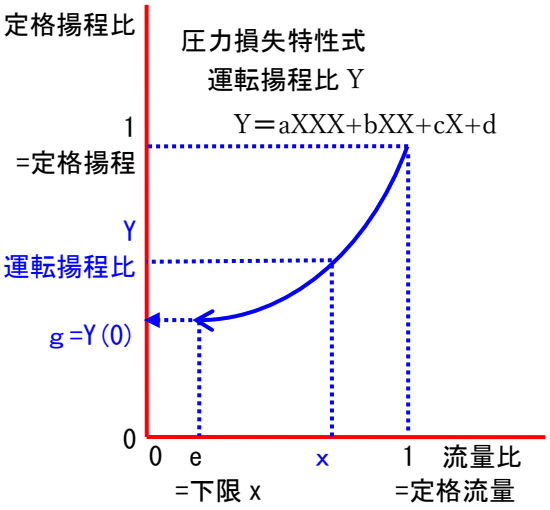


図 3.4.5 圧力損失特性式

・ **圧力損失特性式係数リスト**

「流量制御タイプ」を 5_圧力損失特性式とした場合に、この圧力損失特性式係数リストを特使式に適用し圧力損失（運転揚程）を計算します。

リストは 7 個の数値で、入力数値[a_b_c_d_e_f_g]と計算式は次の通りです。

圧力損失（運転揚程）[Pa]＝運転揚程比[-]×定格揚程[Pa]

運転揚程比[-]＝aXXX＋ｂXX＋ｃX+d

X：流量負荷率[-]＝負荷流量／合計定格流量

e ：計算式の適用下限流量負荷率[-]

f ：計算式の適用上限流量負荷率[-]

g：流量負荷率＝0 の時の定格揚程に対する圧力損失の比（運転揚程の比）[-]

☞運転揚程比は、定格流量比 X=1 と定格揚程比 Y=1 を通るように係数を設定します。(図 3.4.5 を参照)

・ **台数増段流量比リスト、台数減段流量比リスト**

ポンプの運転台数の増段時、減段時の流量比のリストで、ポンプ台数-1 個の数値です。台数増段流量比リストと台数減段流量比リストとで、ポンプ台数切替えのディファレンシャルを形成します。

有効なディファレンシャルとするには、次の大小関係で入力する必要があります。

台数増段流量比の値 ＞ 台数減段流量比の値

適切なディファレンシャルを設けることで、台数増減時の発停の繰り返しを避けることができます。

・ **台数増段の方法、台数減段の方法**

選択肢として次の 2 種類があります。選択肢は台数増段と台数減段に共通です。

0_単体定格流量

1_合計定格流量

「0_単体定格流量」を指定すると、対象の単体の定格流量に対して台数増段の流量比、を適用します。台数減段時も同様です。

例えば、定格流量が[1500_1500_1500]、台数減段流量比リストが[0.9_0.9]の場合、

3 台運転から 2 台運転への減段流量は、1,500＋1,500×0.9＝2,850L/min

2 台運転から 1 台運転への減段流量は、1,500×0.9＝1,350L/min

となります。

「1_合計定格流量」を指定すると、対象の合計定格流量に対して台数増段の流量比、を適用します。台数減段時も同様です。

例えば、定格流量が[1500_1500_1500]、台数減段流量比リストが[0.9_0.9]の場合、

3 台運転から 2 台運転への減段流量は、(1,500＋1,500)×0.9＝2,700L/min

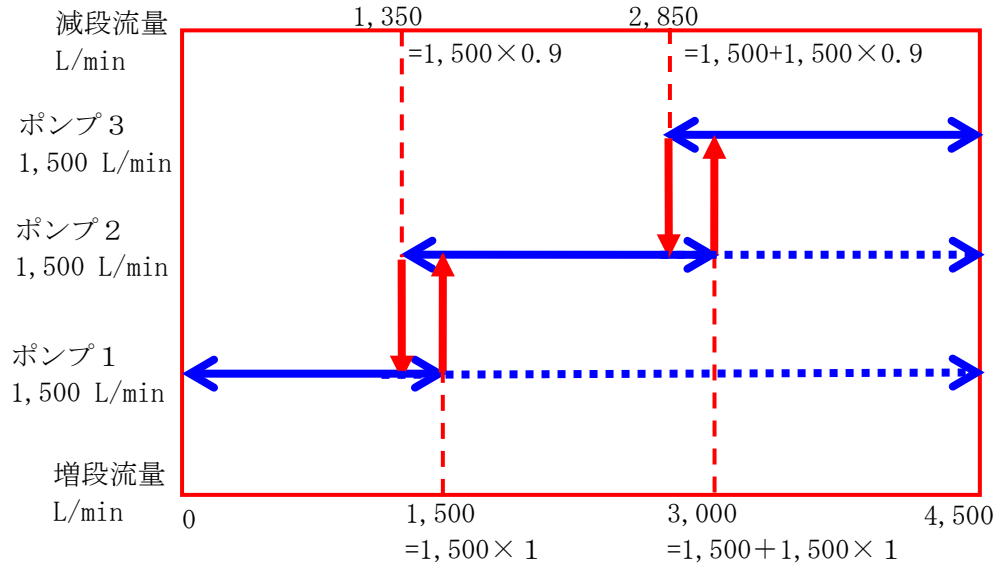
2 台運転から 1 台運転への減段流量は、1,500×0.9＝1,350L/min

となります。

ポンプ台数制御の入力と制御例

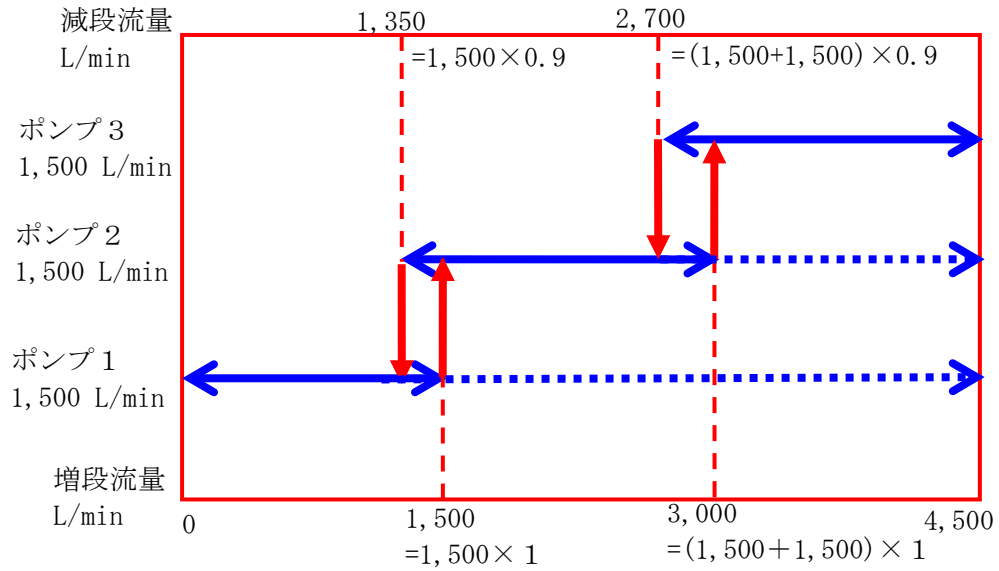
入力例１・・台数増段、減段の方法を単体定格流量とした場合

ポンプの定格流量リスト	1500_1500_1500	[L/min_L/min_・・]
台数増段流量比リスト	1_1	[-_-_・・]
台数減段流量比リスト	0.9_0.9	[-_-_・・]
台数増段の方法	0_単体定格流量	[-]
台数減段の方法	0_単体定格流量	[-]



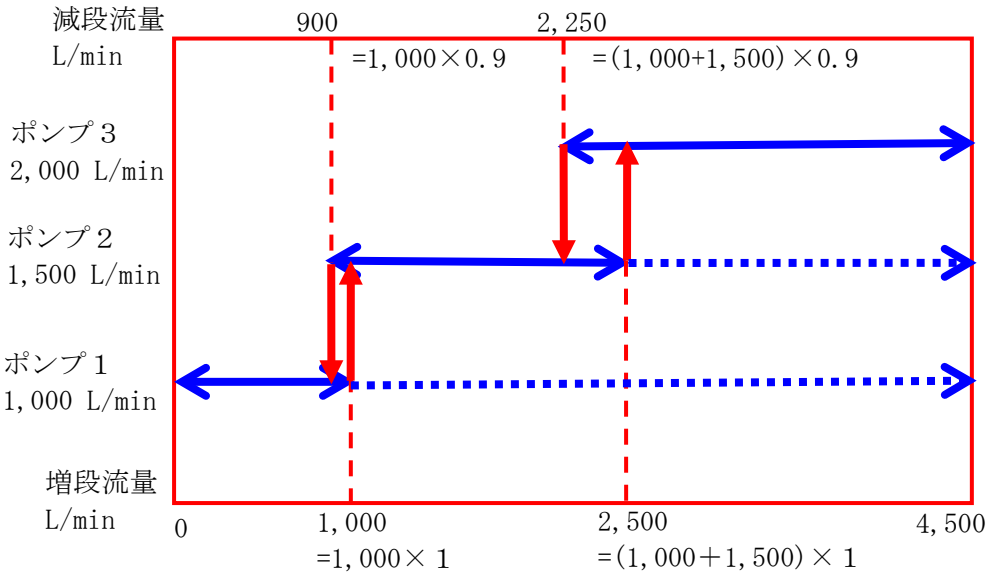
入力例２・・台数増段、減段の方法を合計定格流量とした場合

ポンプの定格流量リスト	1500_1500_1500	[L/min_L/min_・・]
台数増段流量比リスト	1_1	[-_-_・・]
台数減段流量比リスト	0.9_0.9	[-_-_・・]
台数増段の方法	1_合計定格流量	[-]
台数減段の方法	1_合計定格流量	[-]



入力例３・・台数増段、減段の方法を合計定格流量とした場合

ポンプの定格流量リスト	1000_1500_2000	[L/min_L/min_・・]
台数増段流量比リスト	1_1	[-_-_・・]
台数減段流量比リスト	0.9_0.9	[-_-_・・]
台数増段の方法	1_合計定格流量	[-]
台数減段の方法	1_合計定格流量	[-]



③ 記録・グラフ表示

グラフを表示する	☐グラフを表示する	[-]
最大同時表示ステップ数	100	[-]
記録を有効とする	☐記録を有効とする	[-]

・ **グラフを表示する**

グラフ表示をする　を有効とすると計算中のこのモジュールの状態を表す図 3.1.4 のグラフが表示されます。

表示は、ポンプ群の観察している出入口の温度[℃]、観察している２ポイント間に加えられた熱量[W]、入口質量流量および各ポンプへの運転質量流量[g/s]、各ポンプへの転揚程[Ps]、上位からの swc 信号と mod 信号です。

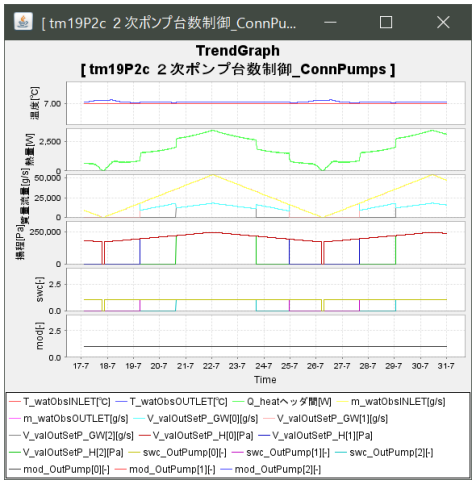


図 3.1.4　計算中のグラフ表示

・ **最大同時表示ステップ数**

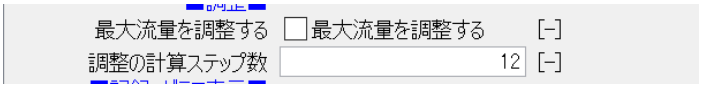
計算中に表示するグラフの時間軸のステップ数を入力します。

５分間隔計算の場合 100 ステップは 8.3 時間分の情報、300 ステップとすると約１日分の情報を表示することができます。

・ **記録を有効とする**

計算結果ファイル best_result.csv へこのモジュールの記録を出力します。

④ 調整



・ **最大流量を調整する**

有効とすると、観察ポイントの流量の移動平均値を算出し、台数制御する各ポンプの（定格）最大流量を見直します。

・ **調整の計算ステップ数**

最大流量の調整は、観察ポイントの流量の移動平均値により行います。平均値を求める観察データ数（＝計算ステップ数）を入力します。

計算時間間隔が 5 分の時、計算ステップ数=12 は 5×12=60 分となり直近の 12 個のデータの 1 時間の移動平均値によって調整を行うこととなります。

(7-2)接続ノードの説明

図 3.2.1 に接続情報の画面を示します。

制御ポンプの台数を 3 台とした場合のもので説明しています。

図 3.2.2 にはポンプ群への接続例を示します。

このモジュールが接続できるポンプモジュールは流量揚程モデルの（接続ノードに L0_valInSetP_GW と L0_ValInSetP_H がある）ものです。ポンプモジュールの接続ノード L1_swcIn、 L1_modIn、 L0_valInSetP_GW、 L0_valInSetP_H へ、ポンプ台数制御モジュールの L1_swcOutPump[0]、 L1_modOutPump[0]、 L0_valOutSetP_GW[0]、 L0_valOutSetP_H[0] を接続します。

また、ポンプ台数制御モジュールの L0_valOutSetP_GW[0]～[2]は、ポンプ群入口側へッダである「配管分岐バイパス入口付き台数制御用」モジュールの L0_valInMwatOut[0]～[2]へも接続し、台数制御モジュールで決めた各ポンプへの分岐流量を渡します。

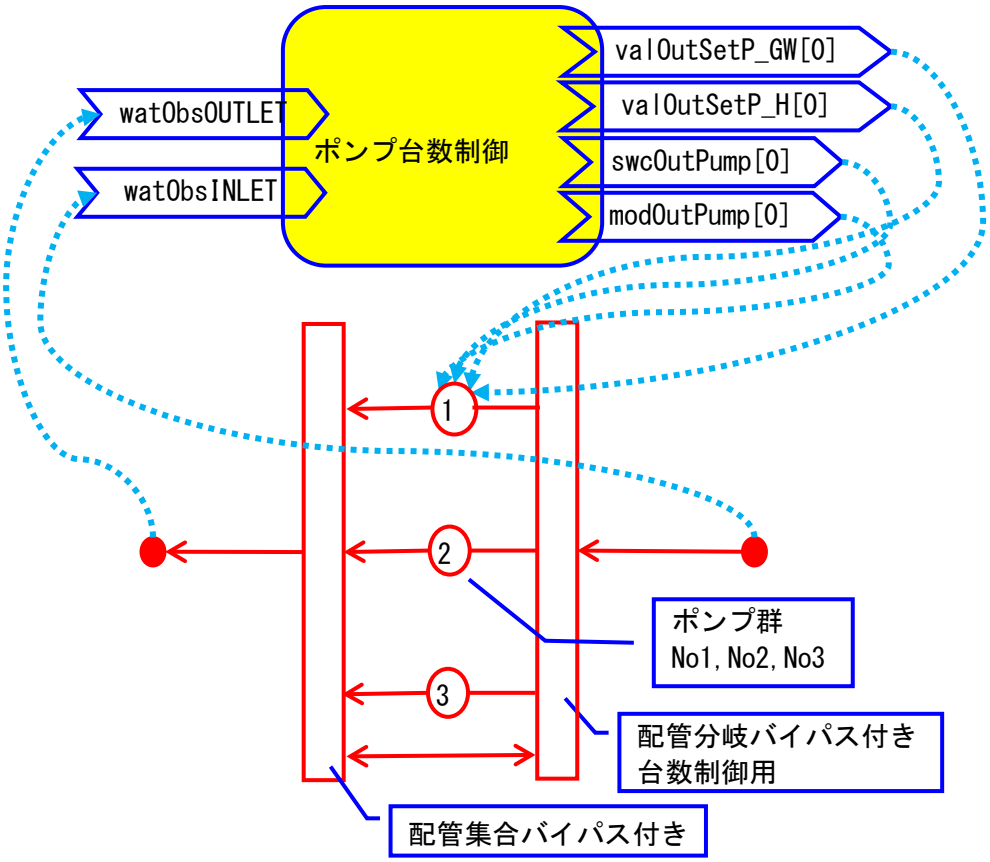


図 3.2.2　ノードの接続例　ポンプ群

(7-3)計算内容の説明

・outputs()の処理動作

ユーザーの入力画面の定格流量リストや制御タイプなどの設定条件により、観察ポイント watObsINLET の負荷流量に応じて必要なポンプの運転台数、流量、揚程を求め、制御対象の各ポンプへ swcOutPump 接続ノードから発停信号、modOutPump 接続ノードから運転モード信号、valOutSetP_GW 接続ノードから運転流量、valOutSetP_H 接続ノードから運転揚程の値を送信します。

ユーザーの入力条件と出口制御信号の状態値の関係を表 3.4.1 に示します。

☞ 主な Airswc 定数は次の通りです

Airswc 定数　　OFF=0、　ON=1

表 3.4.1　入力条件と出口制御信号の状態値の関係

L1_swcIn	ON	OFF	
L1_modIn	*	*	
L1_swcOut	*ON*	*OFF*	
L1_modOut	*	*	
L1_swcOutPump[0] ～[n-1]	*ON or OFF* *台数制御*	*OFF*	
L1_modOutPump[0] ～[n-1]	*	*	
L0_valOutSetP_GW[0] ～[n-1]	*ポンプが on の時：運転流量[g/s]* *Off の時：=0*	*=0*	
L0_valOutSetP_H[0] ～[n-1]	*ポンプが on の時：運転揚程[Pa]* *Off の時：=0*	*=0*	

図 3.1.2 の要領でポンプの運転台数を決めた後に、各ポンプの揚程と流量を算定します。流量制御タイプによって、送水系統の圧力損失特性を変えてポンプ揚程を求めています。圧力損失特性は次のようにしています。

0_定流量　の時

ポンプ揚程と流量は変動しないものとして、ユーザーが入力した「全揚程（設計値）」を L0_valOutSetP_GW[]を通して、流量は各ポンプの定格流量を L0_valOutSetP_H[]を通して

各ポンプへ渡します。

運転するポンプNo.iの揚程と流量は次の通りです。

ポンプ[i]揚程 ＝ 全揚程（設計値）

ポンプ[i]流量 ＝ ポンプ[i]定格流量

☞図 3.4.0

1_段数制御　の時

ポンプ揚程と流量は次の式により求めます。

ポンプ[i]揚程
＝(全揚程設計値－全揚程上限値)×ポンプ[i]運転流量^2／ポンプ[i]定格流量^2＋全揚程上限値

ポンプ[i]流量 ＝ ポンプ群入口流量 × ポンプ[i]定格流量 ／ 運転ポンプ定格流量合計

☞ 図 3.4.1

2_吐出圧一定制御　の時

ポンプ揚程と流量は次の式により求めます。

ポンプ[i]揚程 ＝ 全揚程（設計値）

ポンプ[i]流量 ＝ ポンプ群入口流量 × ポンプ[i]定格流量 ／ 運転ポンプ定格流量合計

☞ 図 3.4.2

3_末端差圧一定制御　の時

ポンプ揚程と流量は次の式により求めます。

ポンプ[i]揚程
＝(全揚程設計値－全揚程下限値)×ポンプ[i]運転流量^2／ポンプ[i]定格流量^2＋全揚程下限値

ポンプ[i]流量 ＝ ポンプ群入口流量 × ポンプ[i]定格流量 ／ 運転ポンプ定格流量合計

☞ 図 3.4.3

4_予想末端差圧制御　の時

ポンプ揚程と流量は次の式により求めます。

ポンプ[i]揚程
＝(全揚程設計値－全揚程下限値)×ポンプ[i]運転流量／ポンプ[i]定格流量＋全揚程下限値

ポンプ[i]流量 ＝ ポンプ群入口流量 × ポンプ[i]定格流量 ／ 運転ポンプ定格流量合計

☞ 図 3.4.4

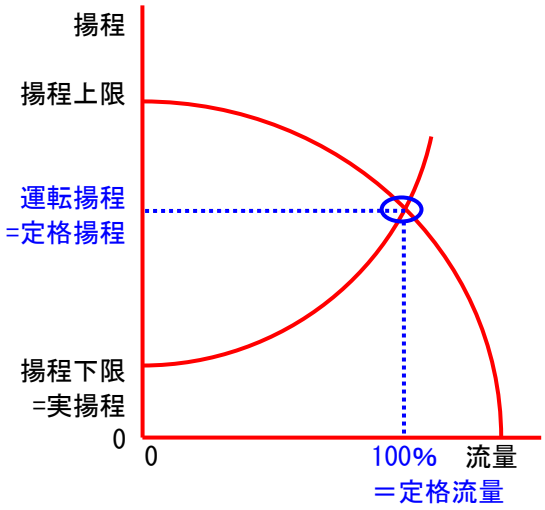


図 3.4.0 定流量

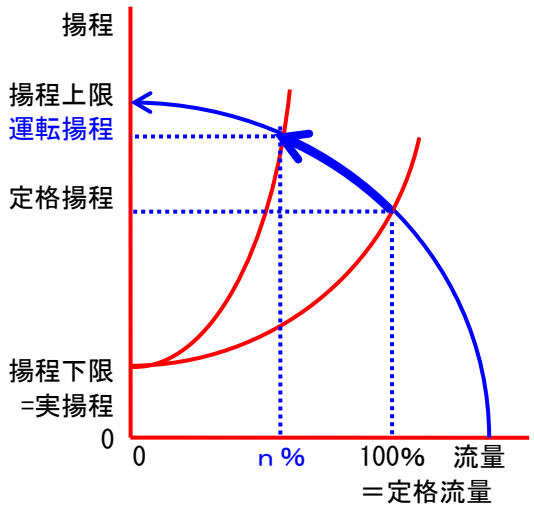


図 3.4.1 段数制御

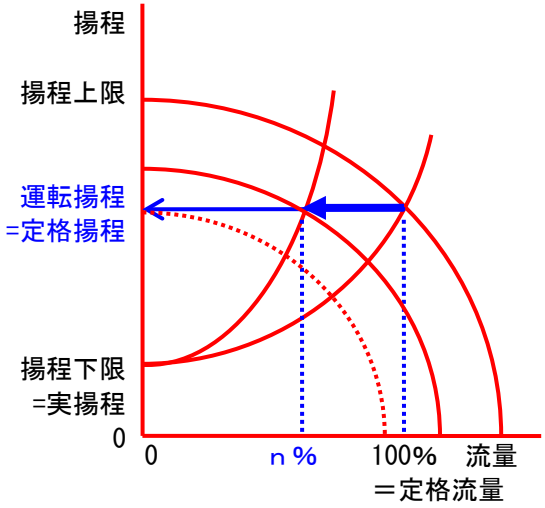


図 3.4.2 吐出圧一定制御

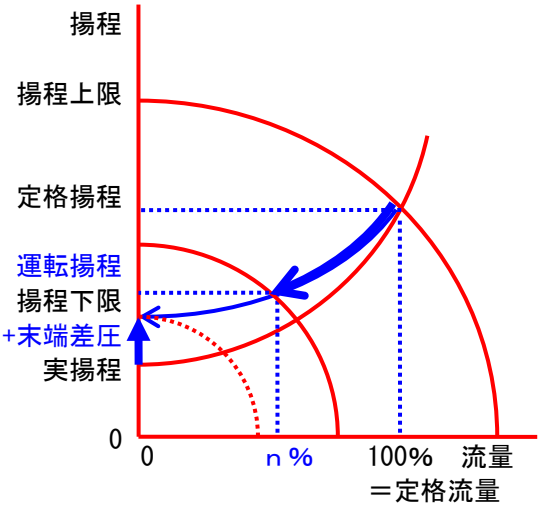


図 3.4.3 末端差圧一定制御

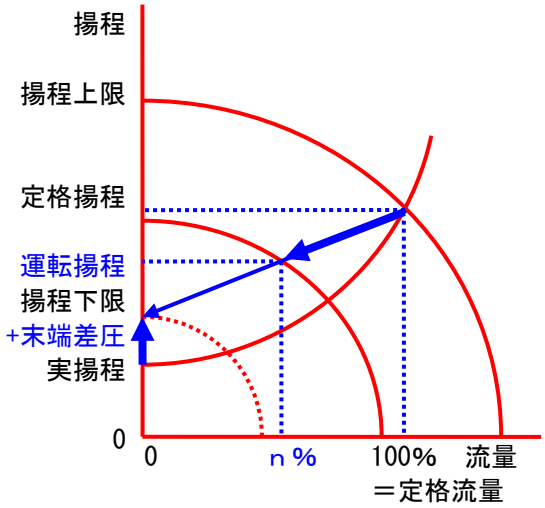


図 3.4.4 予想末端差圧制御

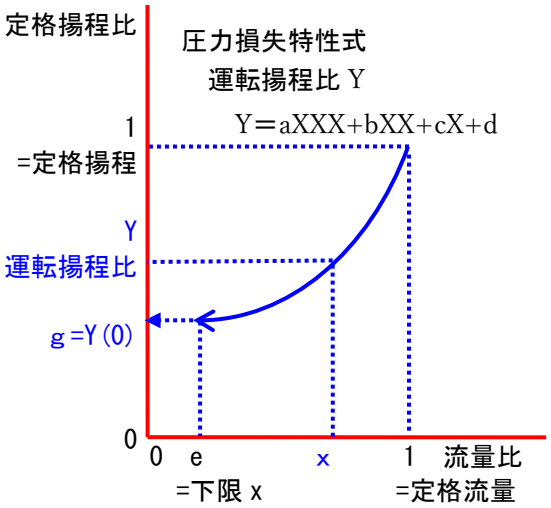


図 3.4.5 圧力損失特性式

・update()の処理動作

このモジュールでは update()の処理動作はありません。

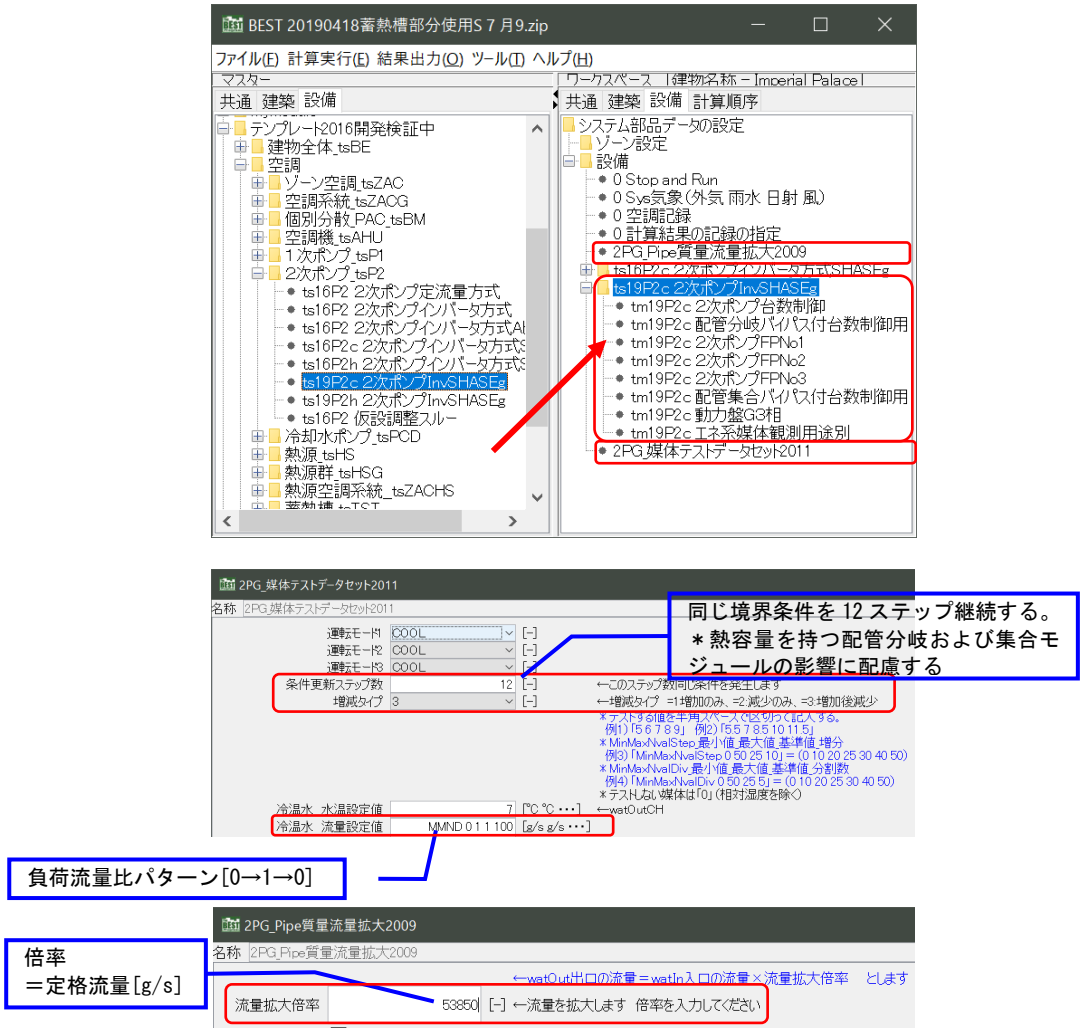
(7-4)　計算例

このポンプ台数制御モジュールを組み込んだ２次ポンプテンプレートを作成した。これと境界条件モジュールを用いた計算例を紹介します。

・２次ポンプテンプレート「ts19P2c　２次ポンプ InvSHASEg」

SHASE のシミュレーション評価法ガイドラインのトライアル用に作成したテンプレートの一つで、２次側へ送水する冷水ポンプ３台で構成する２次ポンプ群に対応するものです。

このテンプレートへ送る負荷流量は、外部境界条件モジュールの「媒体テストデータセット」モジュールからの流量負荷率に相当する wat 発生させ、これを「質量流量拡大」モジュールで負荷流量（wat）に調整したものです。



・媒体テストデータセットモジュールの入力

境界条件として、媒体テストデータセットモジュールで流量負荷率が 0 → 1 → 0 と増加・減少のパターンで 12 ステップごとに 100 分割＝0.01 刻みで変化させる設定としています。

・質量流量拡大モジュールの入力

質量流量拡大モジュールには、制御するポンプ 3 台分の流量 1077L/min × 3 台≒3232L/min＝53850g/s をセットしています。

・ポンプ台数制御モジュールの入力

計算例のポンプ台数制御モジュールの基本入力は次の通りです。

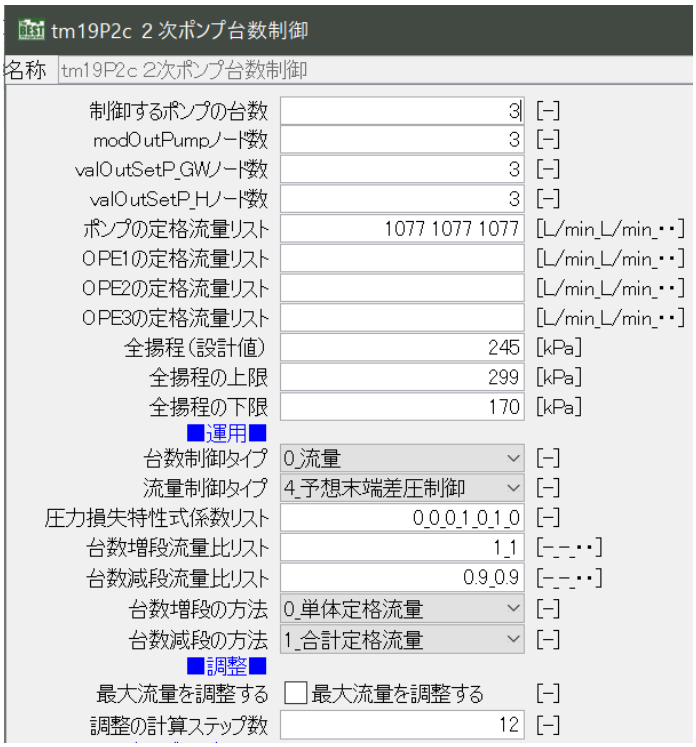
制御台数＝3 台、定格流量＝[1077_1077_1077]

全揚程＝245kPa、上限＝299kPa、下限＝170kPa

台数制御タイプ＝0_流量、流量制御タイプ=4_予想末端差圧制御

台数増段の方法＝0_単体定格流量、台数減段の方＝1_合計定格流量

最大流量を調整する＝しない、調整の計算ステップ数＝12



(7-4-1)・台数制御タイプ　0_流量　X_台数制御なし

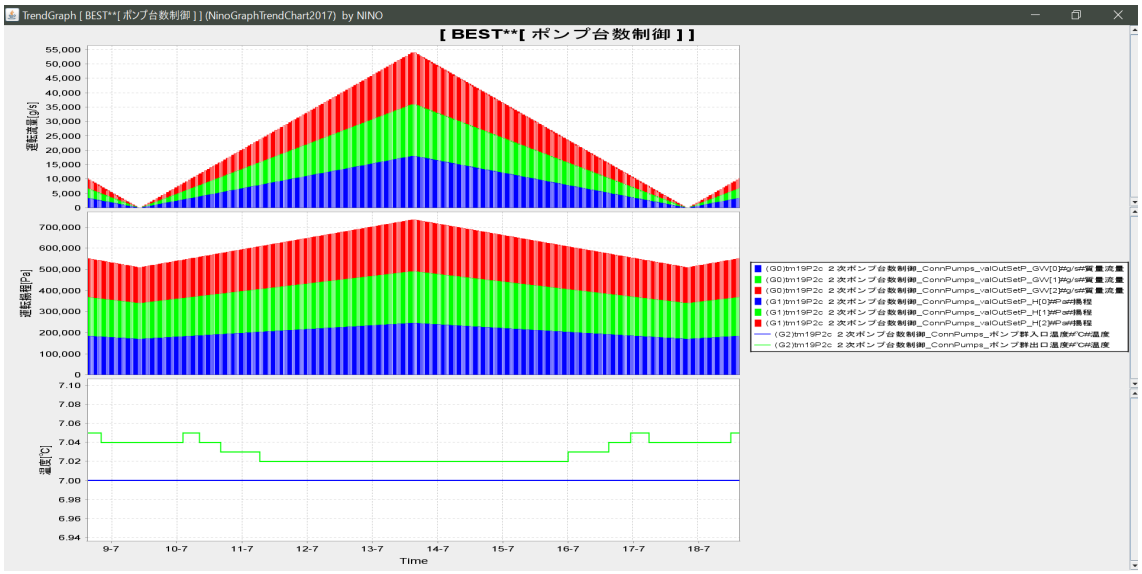
台数制御タイプの選択肢、台数制御あり・なしの比較の計算例です。

下図は境界条件の負荷流量の変化の１サイクル分の計算結果で、上段に各ポンプの運転流量[g/s]、中段に各ポンプの運転揚程[Pa]、下段にポンプ群の出入り口温度を示し、運転流量と運転揚程は積上げ棒グラフで表示している。

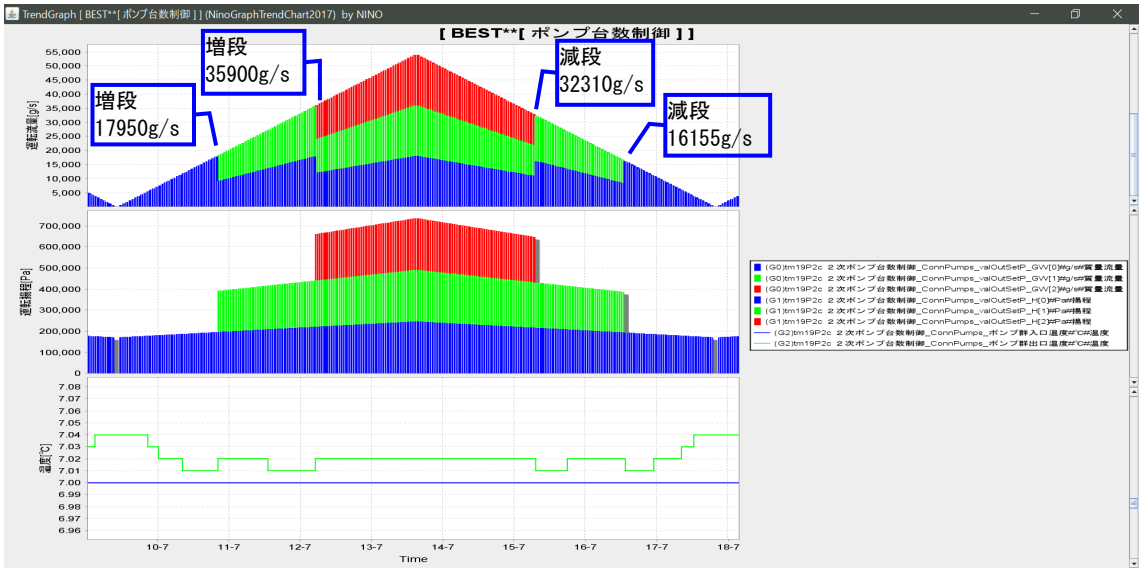
台数制御なしの場合、常時３台運転で各ポンプは均等に負荷流量を分担します。

流量で台数制御した場合、台数の増段、減段の流量比リストに従って、運転台数が決定されています。

X_台数制御なし



1_流量　による台数制御



(7-4-2)・台数増段、台数減段の流量比、その方法

台数増段と減段の方法を変えた比較です。

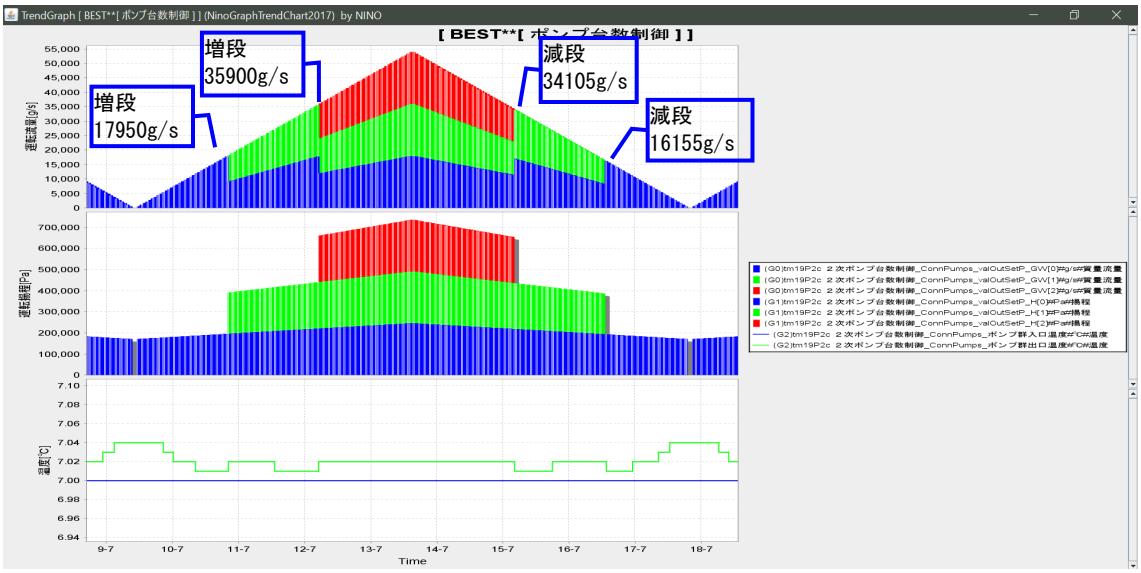
台数増段　1_1、0_単体定格流量　台数減段　0.9_0.9、0_単体定格流量　と

台数増段　1_1、1_合計定格流量　台数減段　0.9_0.9、1_合計定格流量　の比較です。

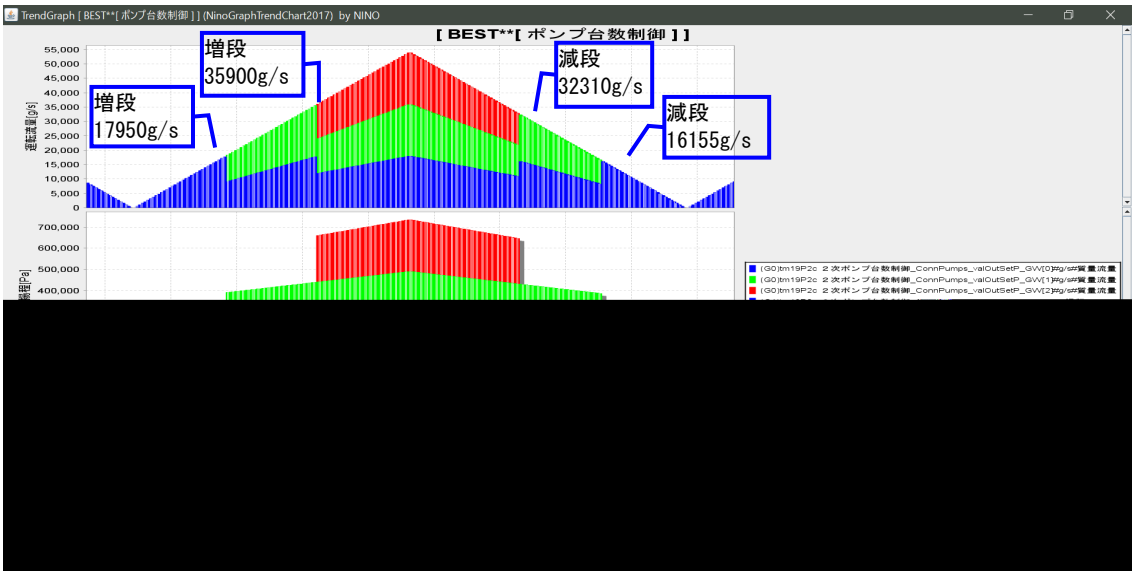
台数増段の方は、台数増段の流量比リストが 1_1 であるため増段の方法の違い（単体定格流量か合計定格流量か）の影響を受けていません。

台数減段の方は、減段時の流量の違いが表れています。

台数増段　1_1、0_単体定格流量　台数減段　0.9_0.9、0_単体定格流量　による台数制御



台数増段　1_1、1_合計定格流量　台数減段　0.9_0.9、1_合計定格流量　による台数制御



(7-4-3)・流量制御タイプ

流量制御タイプの比較です。次の 5 方式について計算例を示しています。

0_定流量

1_弁制御

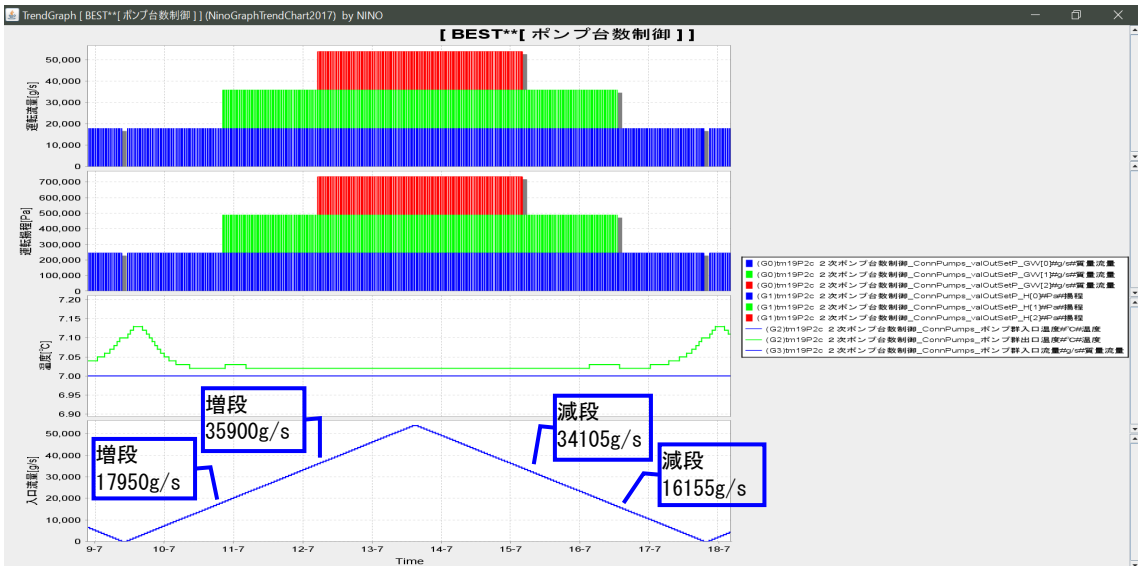
2_吐出圧一定制御

3_末端差圧一定制御

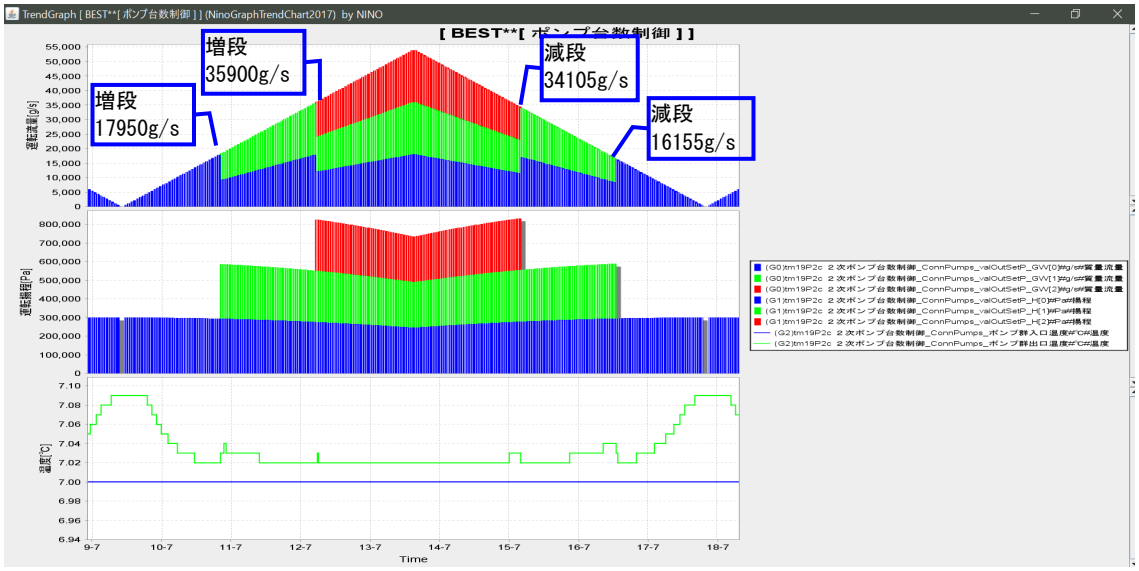
4_予想末端差圧制御

5_圧力損失特性式

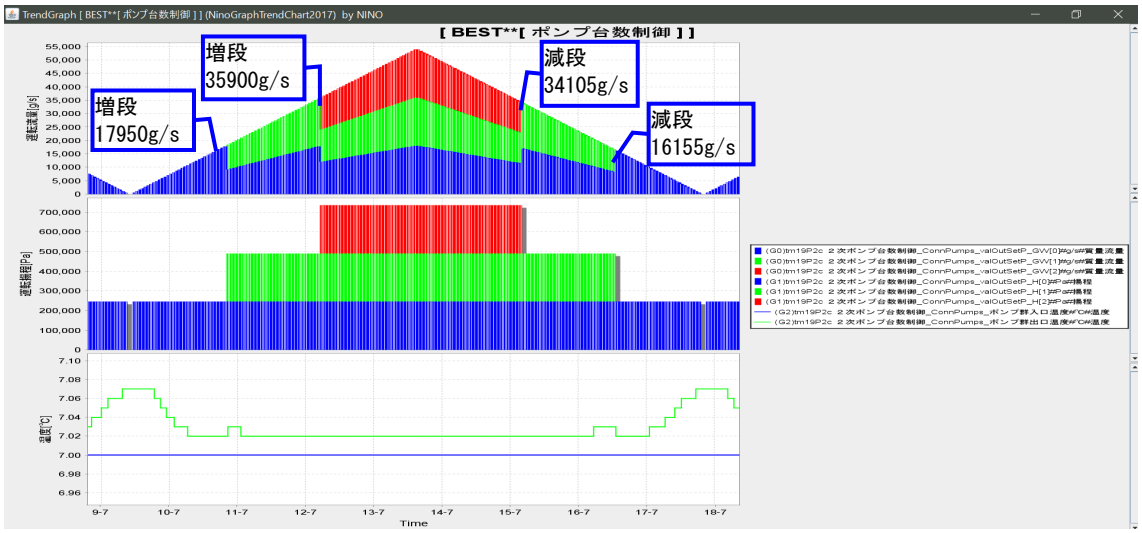
0_定流量　による台数制御



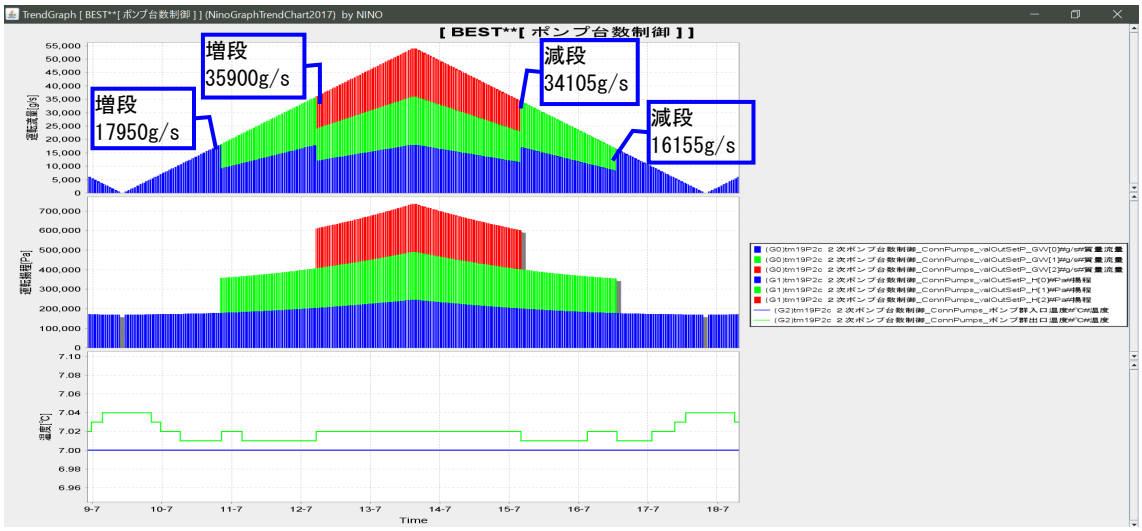
1_弁制御　による台数制御



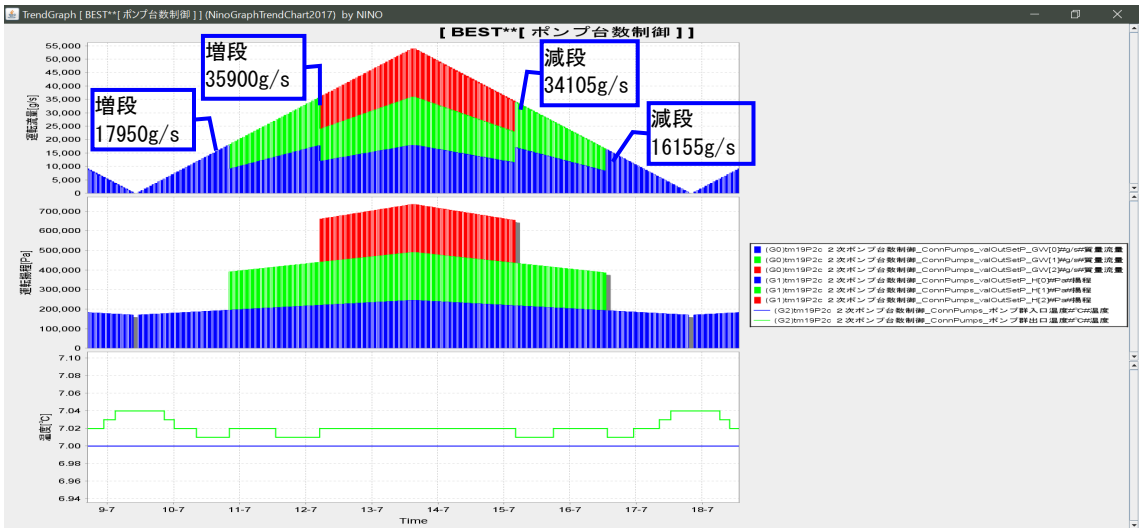
2_吐出圧一定制御　による台数制御



3_末端差圧一定制御　による台数制御

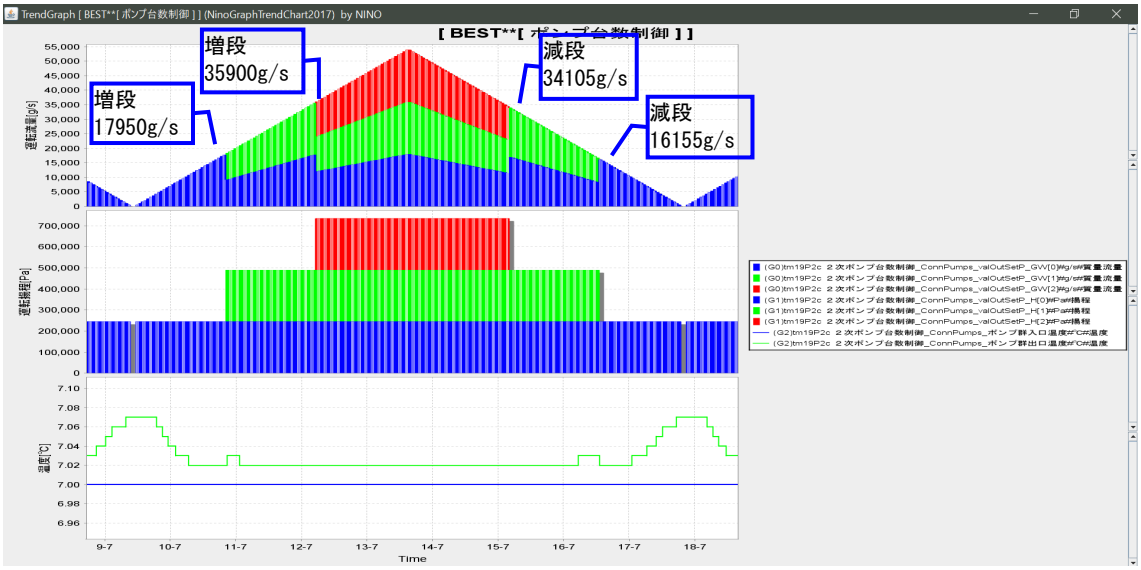


4_予想末端差圧制御　による台数制御



5_圧力損失特性式　[0_0_0_1_0_1_0]　による台数制御

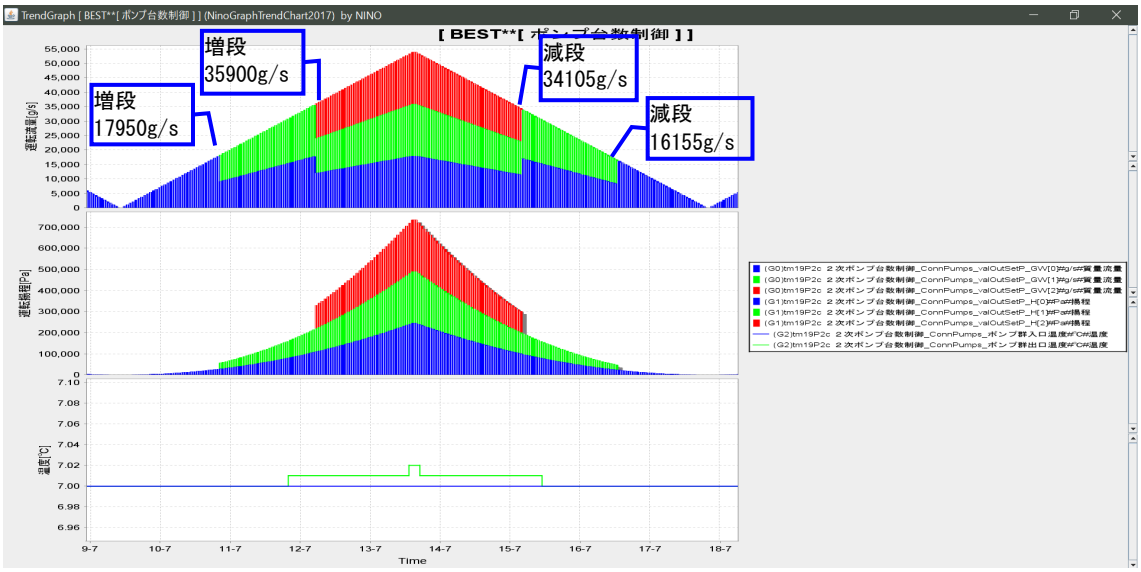
圧力損失特性式係数リストが[0_0_0_1_0_1_0]の時、特性式は　Y(X)＝1　と定数となります。これは流量負荷率 X に関係なく 1 であるので常に定格揚程が渡されることになり、吐出圧力一定制御と同じ計算となります。



5_圧力損失特性式　[0_1_0_0_0_1_0]　による台数制御

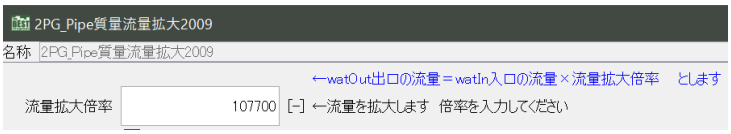
圧力損失特性式係数リストが[0_1_0_0_0_1_0]の時、特性式は　Y(X)＝XX　となります。これは原点（0,0）と定格ポイント(1,1)を通る流量負荷率 X の二次関数です。

運転揚程は下図のようになります。



(7-4-4)・定格流量を超える流量が入ってきたとき

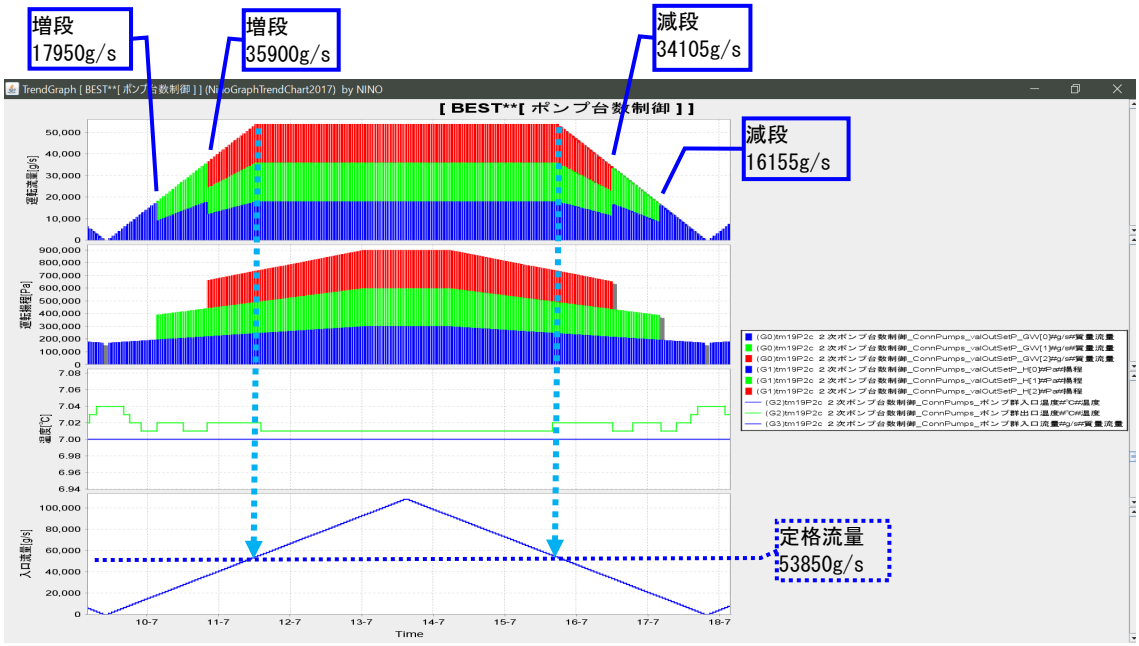
定格流量（=53,850g/s）の 2 倍の流量（=107,700g/s）が入口流量としてポンプ群に流れる場合の計算例です。



ポンプ台数制御モジュールから各ポンプへの運転流量の指示は、定格流量リストに入力された各ポンプの定格流量を上限値とします。上限値を超える流量の場合、ポンプへは定格流量を運転流量として指示します。

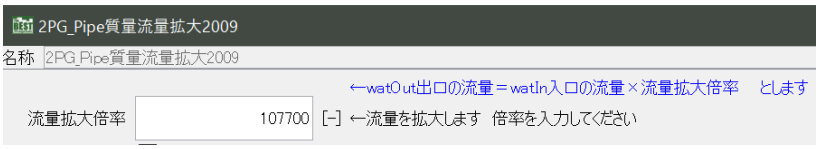
定格流量を超える部分については、ポンプ群を構成する配管分岐バイパス管付モジュールのバイパス管を通って配管集合バイパス管付モジュールへ流れることになります。

入口流量に定格流量の 2 倍までの流量を与えたときの台数制御

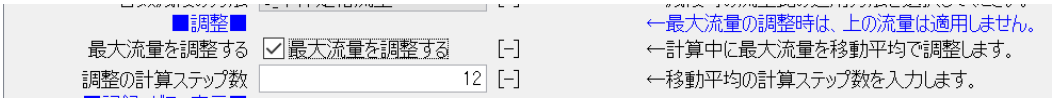


(7-4-5)・最大流量の調整

定格流量（=53,850g/s）の 2 倍の流量（=107,700g/s）が入口流量としてポンプ群に流れる場合のポンプ台数制御の最大流量の調整の計算例です。



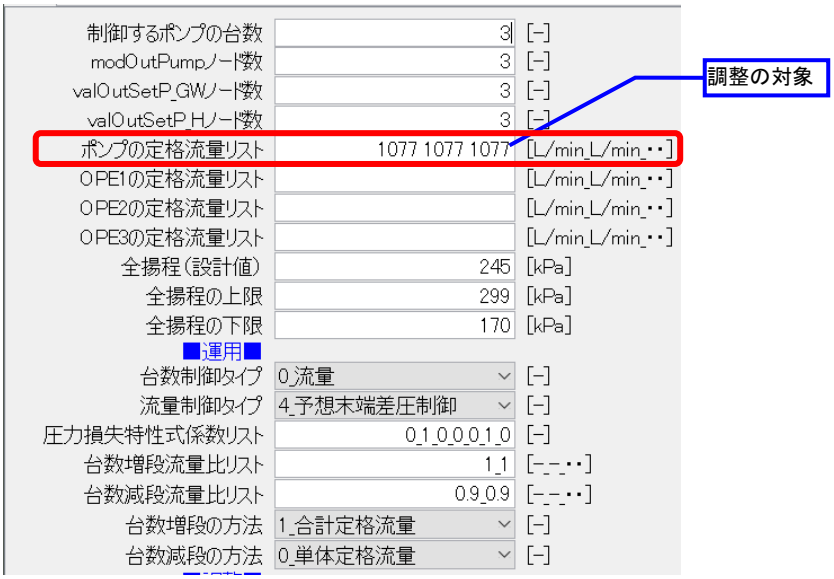
最大流量の調整の計算ステップ数は 12 ステップで計算しています。



最大流量の調整では、下図のポンプ台数制御モジュールの入力項目である、制御するポンプの台数、ポンプの定格流量リストから得られるポンプの定格流量比、全揚程・上限・下限、台数制御タイプ、流量制御タイプ、台数増段流量比リスト、台数減段流量比リスト、台数増段の方法および台数減段の方法の設定値が、調整計算時にも有効です。

これらの入力された設定条件で台数制御しながら調整を行います。

調整で変化するのは、ポンプの定格流量リストです。



下図の計算例のグラフでは、流量を「0→最大→0→最大→0→最大→0」と増加減少を３サイクル分を表示しています。

１サイクル目の入口流量増加時の調整は、各ポンプの最大流量の調整となります。調整は、流量増大方向の調整です。ポンプの定格流量リストの定格流量に関係なく、各ポンプは初期値=10g/s を調整定格流量として始まり、移動平均値の負荷流量がこれを超える場合、調整定格流量を増大更新します。

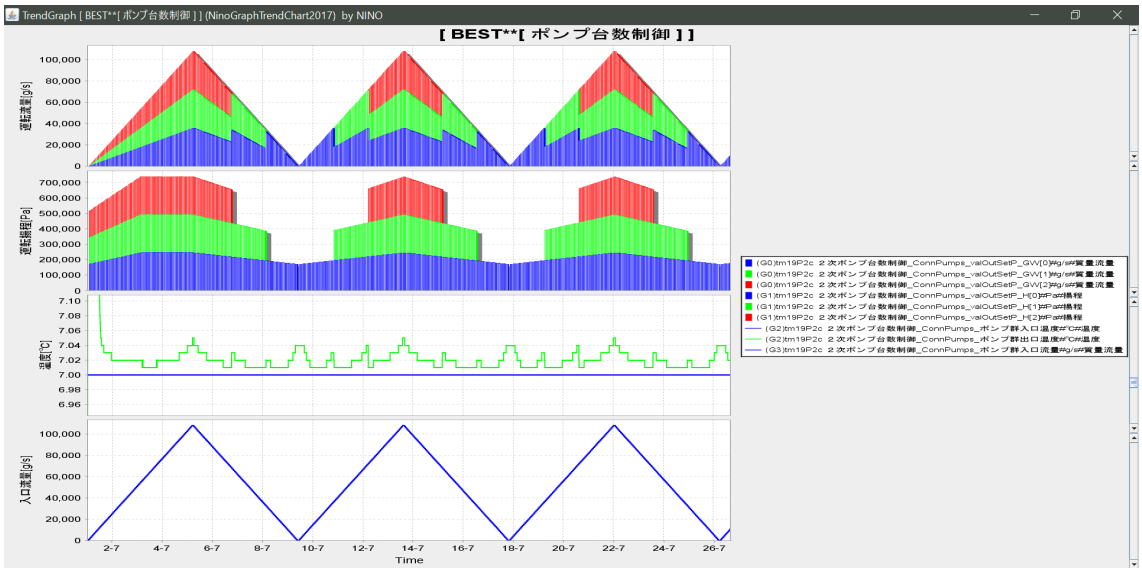
１サイクル目の入口流量が最大流量となった後の各ポンプの調整定格流量は＝35,900g/s＝2,154L/min でした。これは、最大流量 107,700g/s の３分の１の流量であり、適切に調整されています。

１サイクル目の入口流量減少時は、調整は行われず、増加時に調整された調整定格流量で台数制御が行われています。台数の減段の制御は、調整後の調整定格流量をもとに行われています。

２サイクル目の入口流量増加時は、調整後の定格流量リストで求まる定格最大流量をもとに流量負荷率を算定し、流量制御タイプに応じた運転揚程が計算されています。

２サイクル目の入口流量減少時は、２サイクル目の入口流量増加時に最大流量の更新がないので、１サイクル目の入口流量減少時と同じ台数制御となっています。

同様に、３サイクル目は２サイクル目と同じ台数制御となっています。

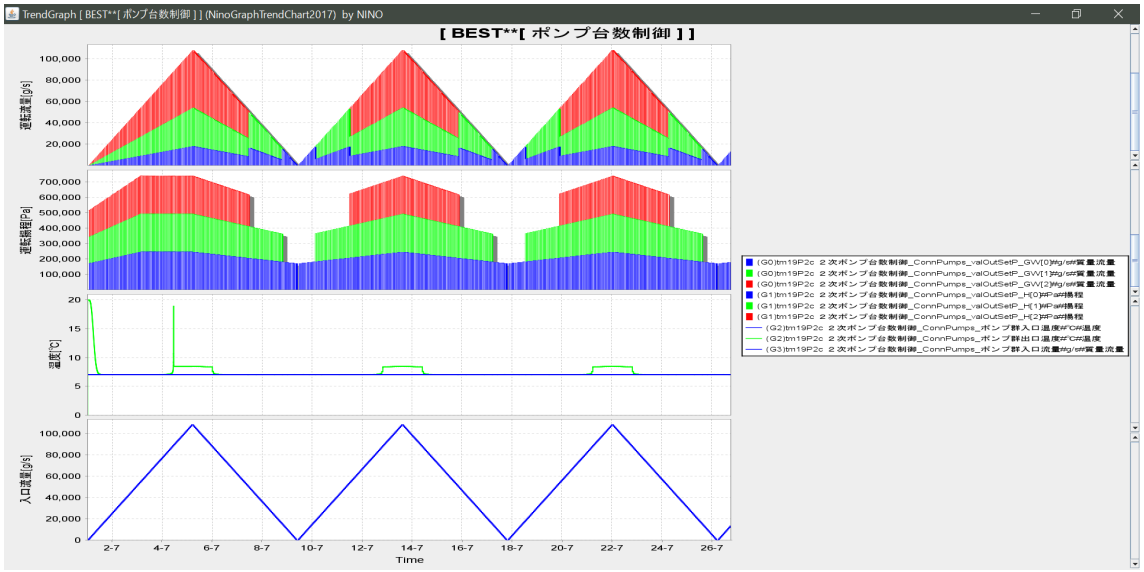


前の例では、ポンプの定格流量リストでは 3 台ともに 1077L/min と同じ流量であるため、調整も同じ最大流量となります。

ポンプの定格流量リストを、3 台の合計流量は変えず、1 ： 2 ： 3 の流量比となるように次のように設定します。

ポンプの定格流量リスト	538 1077 1616	[L/min_L/min_･･]

このポンプの定格流量リストで調整を行うと次のようになります。



名称　　：OA チャンバー外気冷房 2009（場所：設備 2015／空調機 2015）

モジュール名　：OAchamberFreeCoolingModule20091212

(1) 入力画面

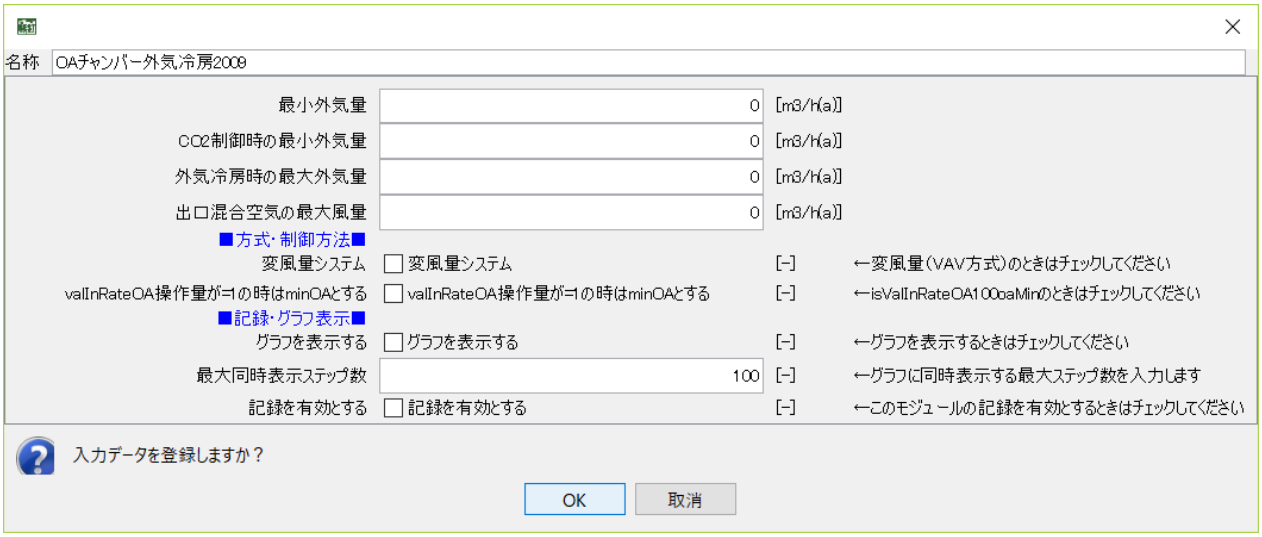


図 1　OA チャンバー外気冷房 2009 モジュールの入力画面

(2) **モジュールの概要**

本モジュールは、外気冷房制御および CO2 制御による外気導入量の変化を再現するモジュールである。VAV 制御時に最小風量となった場合において、設定した最小外気量が確保されるものとしている。

- 最小外気量(定格外気量)
  CAV 方式では、ここで入力した最小外気量を空調時間帯に導入する計算を行う。
  CO2 濃度制御時は、導入外気の最大風量となる。
  VAV 方式で送風量が変動する場合においても、この最小外気量(定格外気量)を導入する。
- CO2 制御時の最小外気量
  CO2 濃度制御時の最小外気量を入力する。
- 外気冷房時の最大外気量
  外気冷房時に導入できる最大外気量を入力する。
- 出口混合空気の最大風量
  チャンバーからの出口混合空気（還気と外気の混合）の最大風量を入力する。
- 一般に次の大小関係となる。
  CO2 制御時の最小外気量 ＜ 最小外気量(定格外気量) ＜ 外気冷房時の最大外気量

(3) **モジュールの入出力**

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

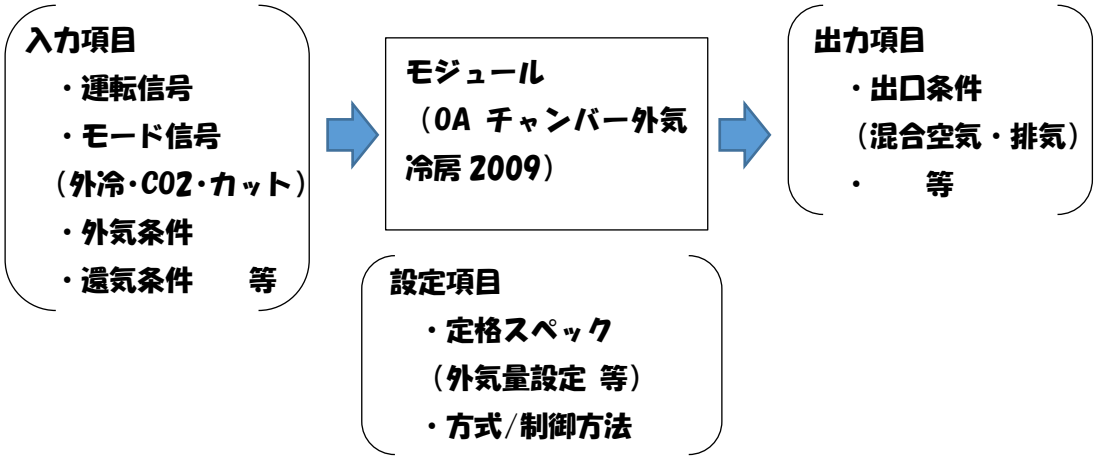


図 2　OA チャンバー外気冷房 2009 モジュール

(4) スペック入力項目（図 2 における設定項目）

表 1　OA チャンバー外気冷房 2009 モジュールのスペック入力項目

NO.	項目	型・クラス名	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	最小外気量	double	MA_OA_min	0	[m3/h(a)]		0	◎	定格時の外気量を入力
1	CO2 制御時の最小外気量	double	MA_OA_minCO2	0	[m3/h(a)]		0	◎	CO2 制御時の外気量制御の下限値を設定
2	外気冷房時の最大外気量	double	MA_OA_maxFC	0	[m3/h(a)]		0	◎	外気冷房制御時の外気量制御の上限値を設定
3	出口混合空気の最大風量	double	MA_MIX_max	0	[m3/h(a)]		0	◎	OA チャンバーで RA・OA 混合後の最大風量を設定
	方式・制御方法	Label							
4	変風量システム	CheckBox		FALSE	[-]			◎	変風量（VAV 方式）のときはチェック
5	valInRateOA 操作量が=1 の時は minOA とする	CheckBox		FALSE	[-]				isValInRateOA100oaMin のときはチェック →使用方法がわかりません。外調機？
	記録・グラフ表示	Label							
6	グラフを表示する	CheckBox		FALSE	[-]				グラフを表示するときはチェック
7	最大同時表示ステップ数	TextField		100	[-]				グラフに同時表示する最大ステップ数を入力
8	記録を有効とする	CheckBox		FALSE	[-]				このモジュールの記録を有効とするときはチェック

(5) シーケンス接続（図 2 における入力項目、出力項目）

表 2　OA チャンバー外気冷房 2009 モジュールのシーケンス接続項目

No.	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
0	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ運転状態等を出力する。
1	発停	L1_swcIn	SWC	-	制御	On/Off 信号	入口	空調制御モジュールからの On/Off 信号を受け取る。
2	モード	L1_modIn	MODE	-	制御	制御モード	入口	外気冷房制御モジュールからモード信号（）を受け取る
3	外気導入の有無	L1_swcInOA	SWCOA	-	制御	On/Off 信号	入口	空調制御モジュールからの外気導入/外気カット信号を受け取る。
4	還気空気状態	L0_airInRA	RA	℃、g/g、ppm、g/s	状態	空気	入口	DB、X、CO2、FR
5	外気空気状態（定格・CO2）	L0_airInOA	OA	℃、g/g、ppm、g/s	状態	空気	入口	DB、X、CO2、FR
6	FC 外気空気状態(外冷用)	L0_airInOAFC	FCOA	℃、g/g、ppm、g/s	状態	空気	入口	DB、X、CO2、FR
7	混合空気状態	L0_airOut	MIX	℃、g/g、ppm、g/s	状態	空気	出口	DB、X、CO2、FR
8	排気空気状態	L0_airOutEA	EA	℃、g/g、ppm、g/s	状態	空気	出口	DB、X、CO2、FR
9	FC 排気空気状態(外冷用)	L0_airOutEAFC	FCEA	℃、g/g、ppm、g/s	状態	空気	出口	DB、X、CO2、FR
10	外気冷房時外気質量流量比	L0_valInRateOA	FR_FC_OA	-	状態	double 値	入口	PID モジュールからの信号（0～1）で外気量を制御する場合に用いる。
11	CO2 制御時外気質量流量比	L0_valInRateOACO2	FR_CO2_OA	-	状態	double 値	入口	PID モジュールからの信号（0～1）で外気量を制御する場合に用いる。
12	VAV 制御時給気質量流量	L0_valInFlowRateVAV	FR_VAV	g/s	状態	double 値	入口	
13	最終外気導入質量流量	L0_valOutOAFlowRate	FR_OA	g/s	状態	double 値	出口	

(6) **記録項目**

表 3　OA チャンバー外気冷房 2009 モジュールの記録項目

No.	項目	出力内容	単位	分類
0	メッセージ	メッセージ	-	メッセージ
1	還気温度	温度	℃	入口
2	還気絶対湿度	質量流量	g/s	入口
3	還気風量	絶対湿度	g/g'	入口
4	還気 CO2 濃度	質量流量	ppm	入口
5	外気温度	温度	℃	入口
6	外気絶対湿度	質量流量	g/s	入口
7	外気風量	絶対湿度	g/g'	入口
8	外気 CO2 濃度	質量流量	ppm	入口
9	FC 外気温度	温度	℃	入口
10	FC 外気絶対湿度	質量流量	g/s	入口
11	FC 外気風量	絶対湿度	g/g'	入口
12	FC 外気 CO2 濃度	質量流量	ppm	入口
13	排気温度	温度	℃	出口
14	排気絶対湿度	質量流量	g/s	出口
15	排気風量	絶対湿度	g/g'	出口
16	排気 CO2 濃度	質量流量	ppm	出口
17	FC 排気温度	温度	℃	出口
18	FC 排気絶対湿度	質量流量	g/s	出口
19	FC 排気風量	絶対湿度	g/g'	出口
20	FC 排気 CO2 濃度	質量流量	ppm	出口
21	混合空気温度	温度	℃	出口
22	混合空気絶対湿度	質量流量	g/s	出口
23	混合空気風量	絶対湿度	g/g'	出口
24	混合空気 CO2 濃度	質量流量	ppm	出口
25	外気冷房制御操作量	操作量	-	入口
26	CO2 制御操作量	操作量	-	入口
27	VAV 風量	操作量	-	入口

(7) 計算方法

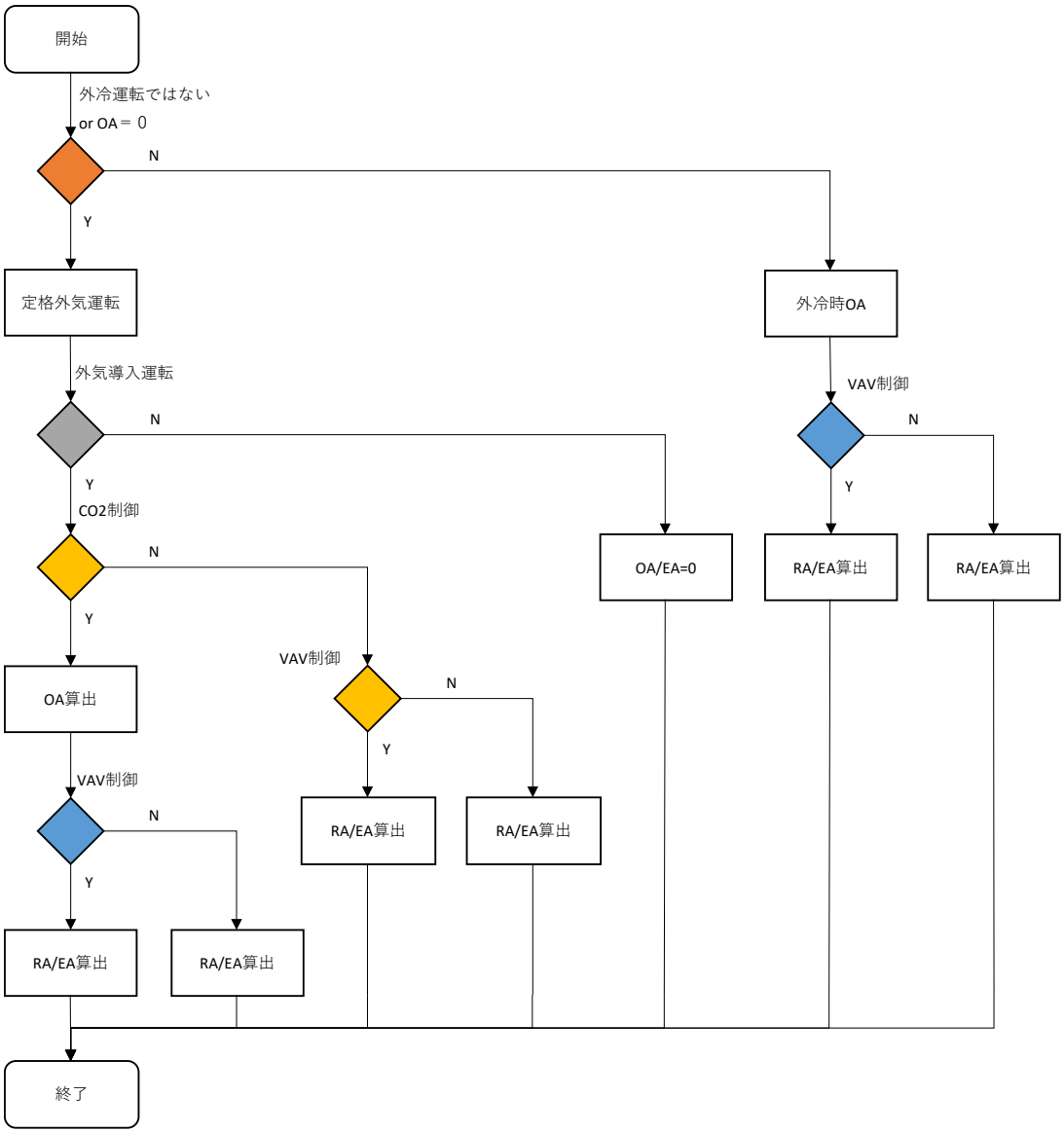


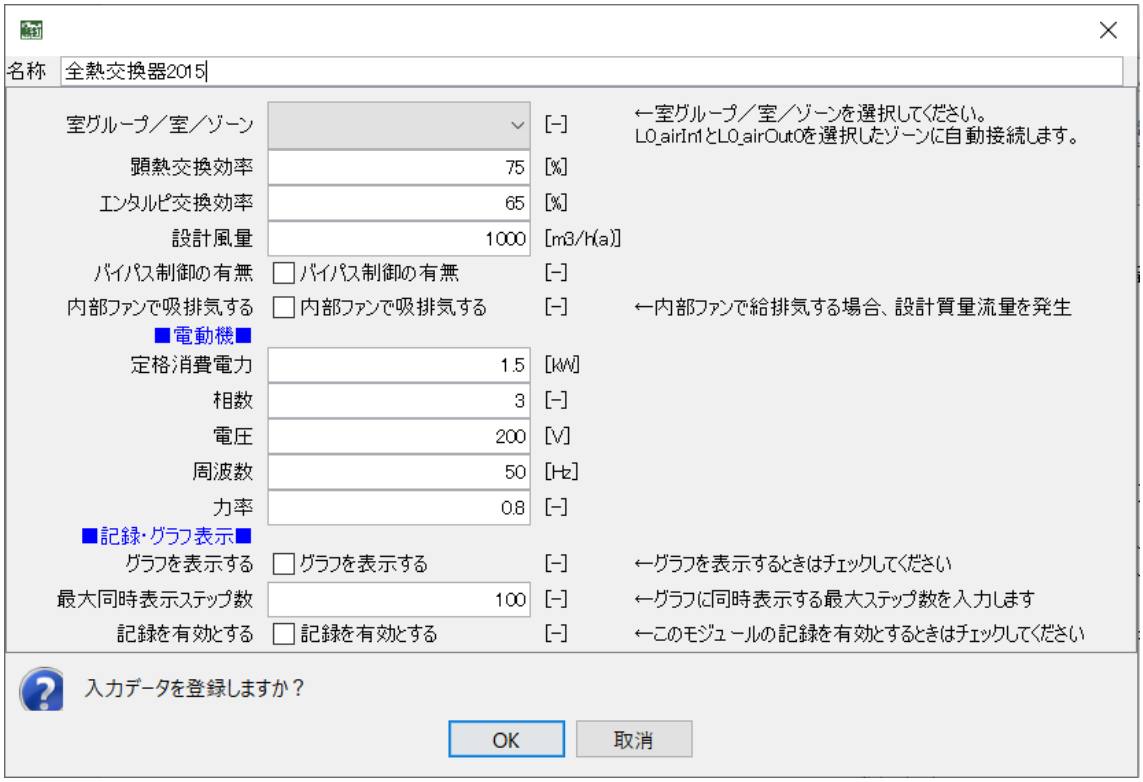
図 3　計算フロー

「全熱交換器」（場所：設備 2015／空調機 2015）

モジュール名	全熱交換器 2015
クラス	TotalHeatExchangerRotor2015

（1）入力画面

・スペック



（２）モジュールの概要

本モジュールは、全熱交換器を再現するモジュールであり、全熱交換器への入口空気状態（OA、RA）と風量比から交換効率を計算し、出口の空気状態を決定するとともに処理熱量の計算を行う。

（３）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

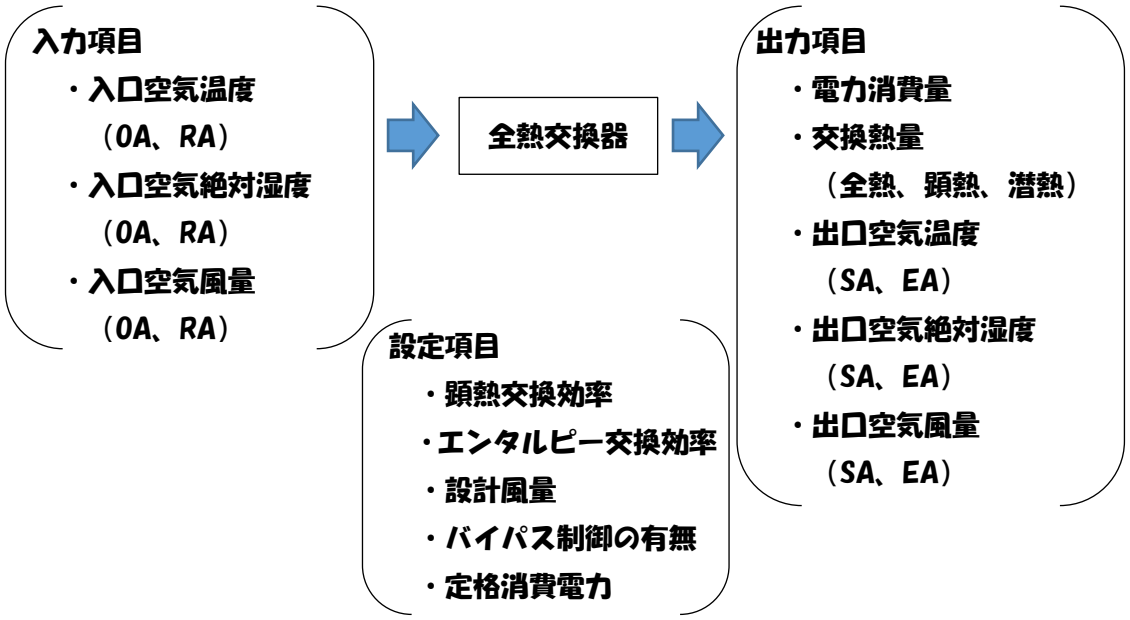


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー	◎	複数種類設定する場合は変更が必要。
1	室グループ／室／ゾーン	String	grzName		[-]	ー	ー	△	
2	顕熱交換効率	double	deffiS	75	[-]	ー	ー	◎	
3	エンタルピー交換効率	double	deffiE	65	[-]	ー	ー	◎	
4	設計質量流量	double		1000	[g/s]	ー	ー	◎	
5	バイパス制御の有無	boolean	isBypass	FALSE(チェック無し)	[-]	ー	ー	○	
2	内部ファンで給排気する	boolean	isOpeMyFan	FALSE(チェック無し)	[-]	ー	ー	○	内部ファンで給排気する場合はチェックしてください。
①	■電動機■								
1	定格消費電力	double	dPEini	1.5	[kW]	ー	ー	○	
2	相数	int	phase	3	[-]	ー	ー	△	
3	電圧	double	voltage	200	[V]	ー	ー	△	
4	周波数	double	frequency	50	[Hz]	ー	ー	△	
5	力率	double	PowerFactor	0.8	[-]	ー	ー	△	
⑤	■記録・グラフ表示■								
1	グラフを表示する	boolean	isGVisible	FALSE(チェック無し)	[-]			○	グラフを表示するときはチェック
2	最大同時表示ステップ数	int	maxItemCount	100	[-]			○	グラフに同時表示する最大ステップ数を入力
3	記録を有効とする	boolean	isRecord	FALSE(チェック無し)	[-]			○	このモジュールの記録を有効とするときはチェック

（5）シーケンス接続（図１における入力項目、出力項目）

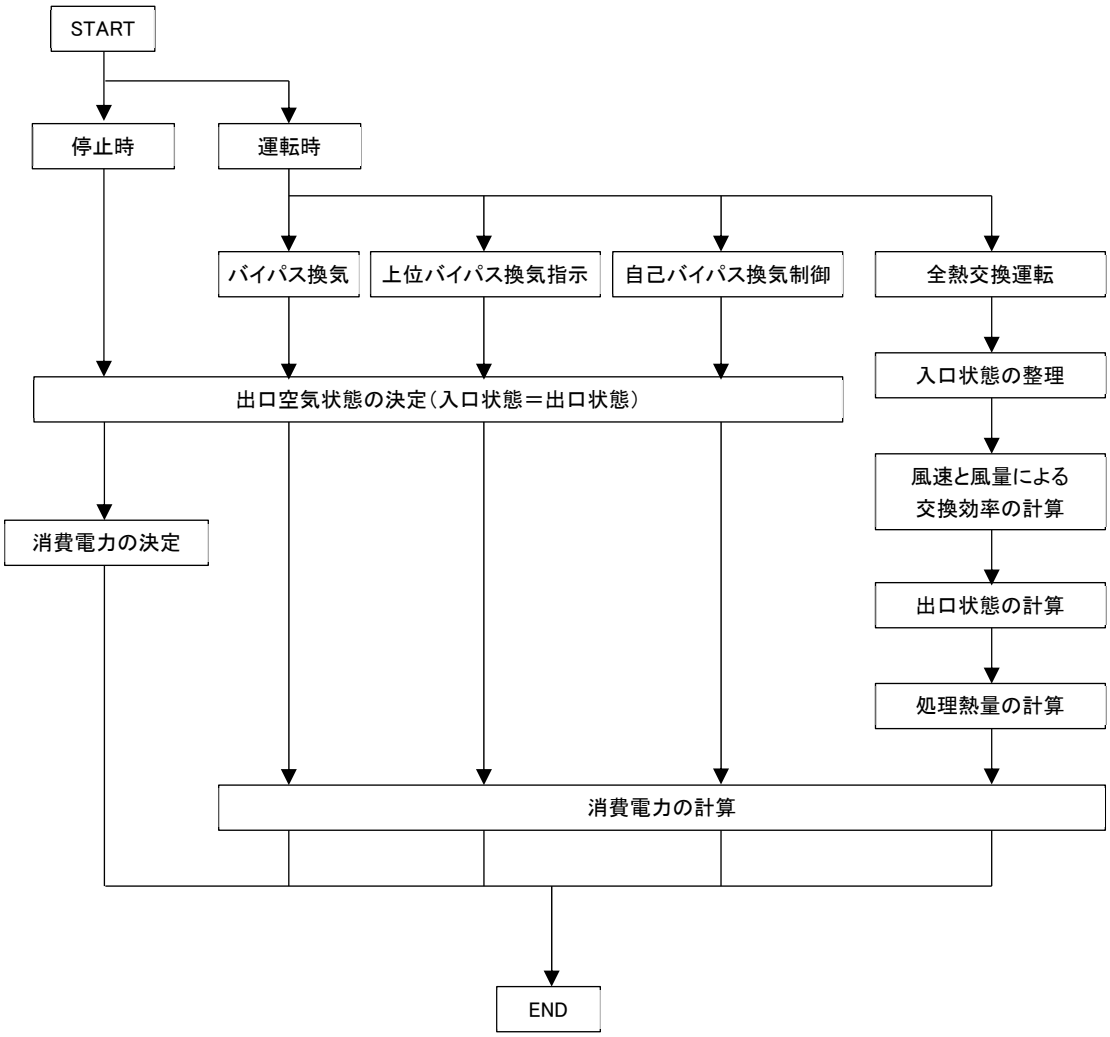
No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	入口空気 0	L0_airIn0	DB_{in0}		状態	空気	入口	
2	入口空気 1	L0_airIn1	DB_{in1}		状態	空気	入口	
3	出口空気 0	L0_airOut0	DB_{out0}		状態	空気	出口	
4	出口空気 1	L0_airOut1	DB_{out1}		状態	空気	出口	
5	電力	L0_eleIn	EP		状態	電気	入口	
7	運転状態	L1_swcIn	SWC		制御	On/Off 信号	入口	制御モジュールからの On/Off 信号を受け取る。
8	外気 On・Off	L1_swcInOA	SWC_{OA}		制御	On/Off 信号	入口	制御モジュールからの外気 On/Off 信号を受け取る。
9	空調モード	L1_modIn	MODE		制御	制御モード	入口	制御モジュールからの冷房・暖房別の信号を受け取る。
10	記録	L2_recOut	REC		記録	メモリ	出口	空調記録モジュールへ運転状態等を出力する。

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
2	空気 0 入口温度	温度	°C	入口
3	空気 0 入口絶対湿度	絶対湿度	g/g'	入口
4	空気 0 入口風量	質量流量	g/s	入口
5	空気 1 入口温度	温度	°C	入口
6	空気 1 入口絶対湿度	絶対湿度	g/g'	入口
7	空気 1 入口風量	質量流量	g/s	入口
8	空気 0 出口温度	温度	°C	出口
9	空気 0 出口絶対湿度	絶対湿度	g/g'	出口
10	空気 0 出口風量	質量流量	g/s	出口
11	空気 1 出口温度	温度	°C	出口
12	空気 1 出口絶対湿度	絶対湿度	g/g'	出口
13	空気 1 出口風量	質量流量	g/s	出口
14	消費電力	電力	W	エネルギー
15	空気 0 処理顕熱量	熱量	W	My
16	空気 0 処理潜熱量	熱量	W	My
17	空気 0 処理全熱量	熱量	W	My

（7）計算フロー・計算内容

モジュール内での計算フローを示す。



名称　　：冷温水コイル 2015（場所：設備 2015／空調機 2015）

モジュール名　：CoilwithValveModule201502

(1) 入力画面

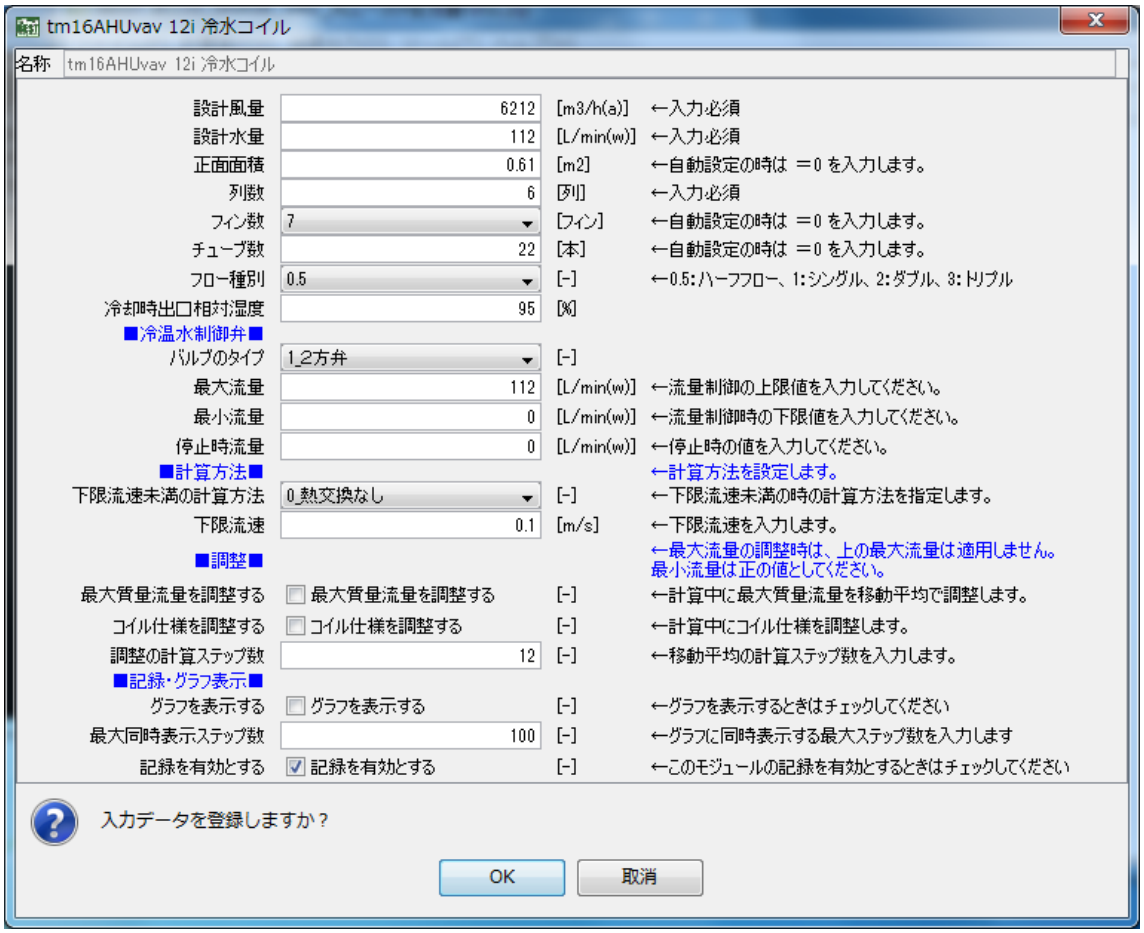


図 1 冷温水コイルモジュールの入力画面

(2) **モジュールの概要**

本モジュールは、冷温水コイルおよび制御弁を再現するモジュールであり、「CoilwithValveModule201502.java」「CoilSpec2015.java」「ValveSpec2015.java」「CoilCalc2015.java」を組み合わせて、一つのモジュールを構成している。

表 1 冷温水コイルのモジュール構成

モジュール	説明
CoilwithValveModule201502.java	制御弁付き（２方弁・３方弁）のコイルモジュール
CoilCalc2015.java	コイル計算モジュール
CoilSpec2015.java	コイル SPEC クラス
ValveSpec2015.java	Valve 2W２方弁

(3) **モジュールの入出力**

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

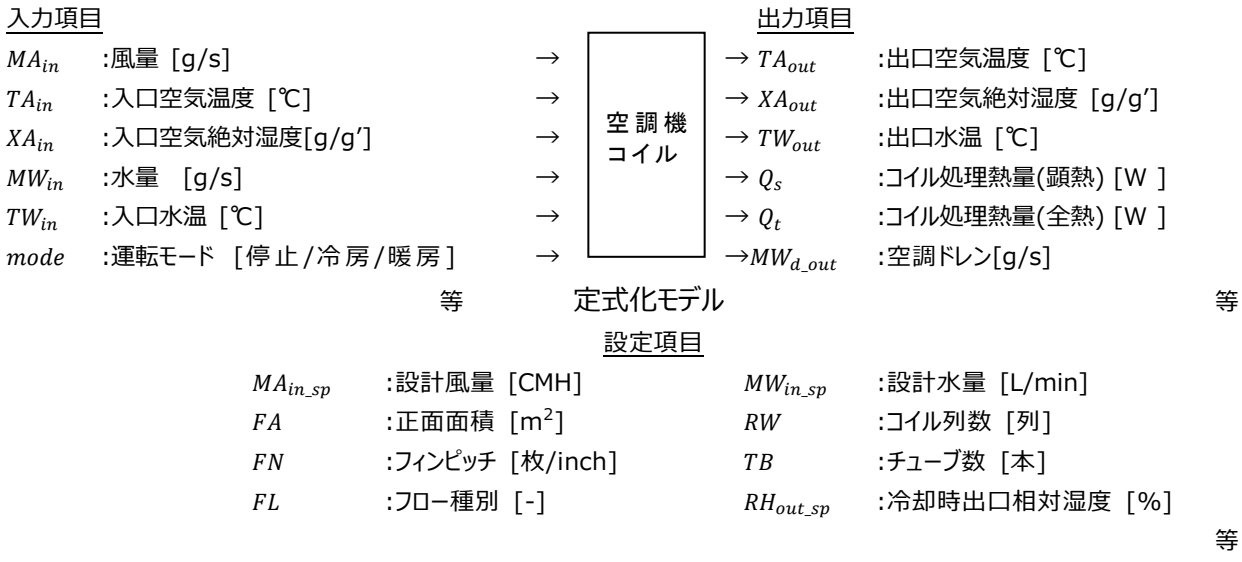


図 2 冷温水コイルモジュールの計算概要

(4) スペック入力項目（図 2 における設定項目）

表 2 空調機コイルモジュールのスペック入力項目

NO.	項目	型・クラス名	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
1	設計風量	double	MA_{sp}	4500	[m3/h(a)]		0	◎	入力必須
2	設計水量	double	MW_{sp}	60	[L/min(w)]		0	◎	入力必須
3	正面面積	double	FA	0.625	[m2]		0	△	自動設定の時は ＝0 を入力します。
4	列数	double	RW	6	[列]		0	◎	入力必須
5	フィン数	double(プルダウン) 0,7,8,12	FN	7	[フィン]		0	△	自動設定の時は ＝0 を入力します。
6	チューブ数	double	TB	20	[本]		0	△	自動設定の時は ＝0 を入力します。
7	フロー種別	double(プルダウン) 0,0.5,1,2,3	FLW	0.5	[-]		0	○	0.5：ハーフフロー,1：シングル,2：ダブル,3：トリプル
8	冷却時出口相対湿度	double	RH_{out_sp}	95	[%]	100	1	○	
9	■冷温水制御■								
10	バルブのタイプ	String(プルダウン) 0_なし,1_２方弁,2_３方弁		1_２方弁	[-]			◎	
11	最大流量	double		100	[L/min(w)]		0	◎	流量制御の上限値を入力してください。
12	最小流量	double		0	[L/min(w)]		0	○	流量制御時の下限値を入力してください。
13	停止時流量	double		0	[L/min(w)]		0	△	停止時の値を入力してください。
14	■計算方法■								
15	下限流速未満の計算方法	String(プルダウン)0_熱交換なし,1_下限流速の能力を補間		,1_下限流速の能力を補間	[-]			◎	下限流速未満の時の計算方法を指定します。
16	下限流速	double		0.1	[m/s]			△	下限流速を入力します。
17	■調整■								
18	最大質量流量を調整する	boolean		FALSE	[-]			△	計算中に最大質量流量を移動平均で調整します。
19	コイル仕様を調整する	boolean		FALSE	[-]			△	計算中にコイル仕様を調整します。
20	調整の計算ステップ数	int		12	[-]			△	移動平均の計算ステップ数を入力します。
21	■記録・グラフ表示■								
22	グラフを表示する	boolean		FALSE	[-]			○	グラフを表示するときはチェックしてください
23	最大同時表示ステップ数	int		100	[-]			○	グラフに同時表示する最大ステップ数を入力します
24	記録を有効とする	boolean		FALSE	[-]			○	このモジュールの記録を有効とするときはチェックしてください

(5) シーケンス接続（図 2 における入力項目、出力項目）

表 3 空調機コイルモジュールのシーケンス接続項目

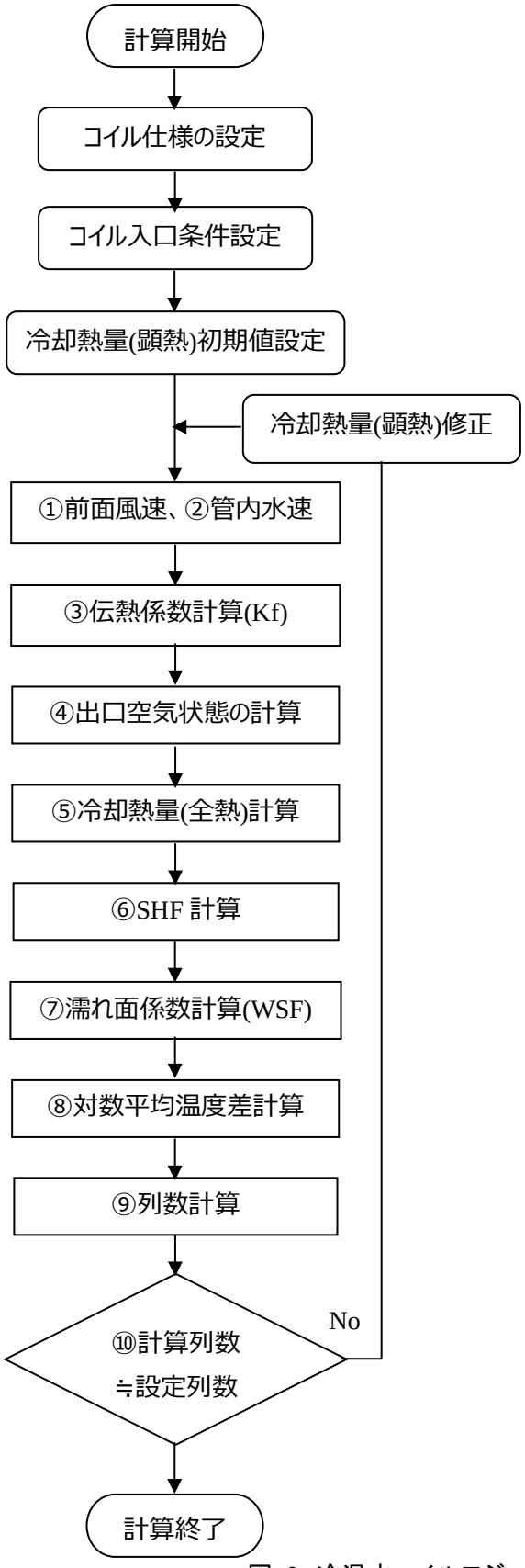
No.	項目	接続端子名	記号	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut		記録	メモリ	出口	空調記録モジュールへ運転状態等を出力する。
2	上位モジュールからの運転状態	L1_swcIn		制御	On/Off 信号	入口	外気冷房制御（空調機制御）モジュールから On/Off 信号を受け取る。
3	上位モジュールからの空調制御	L1_modIn		制御	制御モード	入口	外気冷房制御（空調機制御）モジュールから冷房/暖房モード信号を受け取る。
4	コイル入口水	L0_watInCH	MW_{in}, TW_{in} 等	状態	水	入口	
5	コイル出口水	L0_watOutCH	MW_{out}, TW_{out} 等	状態	水	出口	
6	コイル入口空気	L0_airIn	MA_{in}, TA_{in} 等	状態	空気	入口	
7	コイル出口空気	L0_airOut	MA_{out}, TA_{out} 等	状態	空気	出口	
8	コイル出口凝縮水	L0_watOutD	MW_{d_out} 等	状態	水	出口	
9	バルブ操作値入力	L0_valInCtrl		状態	double 値	入口	PID モジュールからの操作指示値
10	運転時の質量流量	L0_valOutCtrlFlowRate	MA_{out}	状態	double 値	出口	

(6) **記録項目**

表 4 空調機コイルモジュールの記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	-	メッセージ
2	入口空気温度	乾球温度	℃	入口
3	入口絶対湿度	絶対湿度	g/g'	入口
4	入口空気流量	質量流量	g/s	入口
5	出口空気温度	乾球温度	℃	出口
6	出口絶対湿度	絶対湿度	g/g'	出口
7	出口空気流量	質量流量	g/s	出口
8	入口冷温水温度	温度	℃	入口
9	入口冷温水流量	質量流量	g/s	入口
10	出口冷温水温度	温度	℃	出口
11	出口冷温水流量	質量流量	g/s	出口
12	出口ドレン温度	温度	℃	出口
13	出口ドレン流量	質量流量	g/s	出口
14	冷却負荷	負荷	W	MY
15	加熱負荷	負荷	W	MY
16	処理熱負荷	負荷	W	MY
17	処理顕熱負荷	負荷	W	MY
18	面風速	風速	m/s	MY
19	流速	流速	m/s	MY
20	調整最大風量	質量流量	g/s	MY
21	調整ステップ平均風量	質量流量	g/s	MY
22	調整最大水量	質量流量	g/s	MY
23	調整ステップ平均水量	質量流量	g/s	MY

(7) **計算方法**



列数、フィン数、前面面積、チューブ数、フロー
出口 RH、伝熱係数の近似係数

風量、水量、入口空気温度・湿度、入口水温

$$KF = 1/\{1/(C1 \times {V_W}^{C2}) + 1/(C3 \times {V_A}^{C4}) + C5\}$$

C1～C4 : 係数　　C5 : 管壁の熱抵抗

エンタルピー差より全熱量計算

$$C_{wsh} = C6 \times {A_{shf}}^2 - C7 \times A_{shf} + C8$$

冷却コイルの計算設定は以下の４種類とした。
(フラグで設定する)

1)顕熱固定
部屋側からの顕熱要求に合わせて、水量をコントロールする。
2)水量固定(CAV)
風量、水量を固定し、出口側を計算する。
3)出口空気温度固定(VAV)
風量と出口温度を固定し、水量をコントロールする。
4) 出口空気湿度固定(再熱等に利用)
風量と出口湿度を固定し、水量をコントロールする。

尚、加熱コイルは 1)、2)、3)の設定とした。

図 3 冷温水コイルモジュールの計算フロー

① 前面風速の計算

V_A：コイル面風速[m/s]

$V_A = MA/3600/FA$

ここで

MA_{sp}：設計風量[CMH]

FA：冷却コイルの正面面積[m²]（ただし、FAが未入力の場合 $FA = MA_{sp}/3600/2.5$ ）

② コイル管内流速の計算

V_W：コイル管内流速[m/s]

$V_W = MW/(10.8/TB/FLW)$

ここで、

MW_{sp}：設計水量[L/min]

TB：チューブ数

FLW：フロー数 フロー種別 FL により決定（HF=0,5, SF=1, DF=2, TF=3）

ただし、FLWが未入力の場合

$d1 = MW_{sp}/(TB \times 10.8)$

if($d1$<=1.01) { $FLW = 0.5$

} else if ($d1$<=2.01) { $FLW = 1$

} else if ($d1$<=4.02) { $FLW = 2$

} else if ($d1$<=6.03) { $FLW = 3$

} else { $FLW = 3$;}

③ 伝熱係数の計算

$KF = 1/\{1/(C1 \times {V_W}^{C2}) + 1/(C3 \times {V_A}^{C4}) + C5\}$

ここで

KF：伝熱係数[W/m²・K・列]

$C1$～5：伝熱係数決定パラメータ(FNにより係数変更)

FN：フィンピッチ（7,10,12 枚/inch から選択）

V_W：コイル管内流速[m/s]

V_A：コイル面風速[m/s]

④ 出口空気状態の計算

入口空気の状態点より、顕熱冷却能力を仮定して、出口乾球温度を計算。

$A_{sh} = 5.0 \times MW \times 60 / 0.86 \times 0.7$

$TA_{out} = TA_{in} - A_{sh} /MA/ 0.335$

ここで、

A_{sh}：顕熱冷却能力[W]

MW：水量[L/min]

MA：風量[CMH]

TA_{out}：出口乾球温度[℃]

TA_{in}：入口乾球温度[℃]

A)　設定した冷却時出口相対湿度での絶対湿度を計算。

$XA_{out} = Psychrometrics.FNxtr(TA_{out}, RH_{out\_sp})$

ここで、

TA_{out}：出口乾球温度[℃]

$Psychrometrics.FNxtr$：温度・相対湿度より絶対湿度を計算する関数

RH_{out_sp}：冷却時出口相対湿度[%RH]

B)　出口絶対湿度 XA_{out}[g/g‘]＞ 入口絶対湿度 XA_{in}[g/g‘]の場合は以下の修正を行う。

出口絶対湿度XA_{out}[g/g‘]＝入口絶対湿度XA_{in}[g/g‘]

（これにより空気線図上は真横への動きになる。）

C)　出口絶対湿度 XA_{out}[g/g‘]＜ 入口絶対湿度 XA_{in}[g/g‘]の場合は減湿冷却として扱う。

⑤　コイル処理熱量（全熱）の計算

A)　入口エンタルピ、出口エンタルピを計算。

$IA_{in} = 1.006 \times TA_{in} + (1.805 * TA_{in} + 2501) \times XA_{in}$

$IA_{out} = 1.006 \times TA_{out} + (1.805 * TA_{out} + 2501) \times XA_{out}$

ここで、

IA_{in}：入口エンタルピ[kJ/kg']

TA_{in}：入口乾球温度[℃]

XA_{in}：入口絶対湿度[kg/kg']

IA_{out}：出口エンタルピ[kJ/kg']

TA_{out}：出口乾球温度[℃]

XA_{out}：出口絶対湿度[kg/kg‘]

B)　エンタルピ差よりコイル処理熱量（全熱）を計算。

$A_{th} = (IA_{in} - IA_{out}) / 4.186 \times MA \times 1.2 / 0.86$

ここで、

A_{th}：全冷却能力[W]

IA_{in}：入口エンタルピ[kJ/g']

IA_{out}：出口エンタルピ[kJ/g']

MA：風量[CMH]

⑥　顕熱比の計算

$A_{shf} = A_{sh} / A_{th}$

ここで、

A_{shf}：SHF[-]

A_{sh}：顕熱冷却能力[W]

A_{th}：全冷却能力[W]

⑦　濡面係数の計算

$C_{wsh} = C6 \times {A_{shf}}^2 - C7 \times A_{shf} + C8$

ここで、

C_{wsh}：濡面係数[-]

$C6$～8：濡れ面係数決定パラメータ

⑧　対数平均温度差の計算

$\Delta t_m = (\Delta t_1 - \Delta t_2)/ln(\Delta t_1/\Delta t_2)$

Δt_m：空気の乾球温度と冷水温度の対数平均温度差

$\Delta t_1 = (\ TA_{in} - TW_{out})$

$\Delta t_2 = (\ TA_{out} - TW_{in})$

ここで、

TA_{in}：入口乾球温度[℃]

TA_{out}：出口乾球温度[℃]

TW_{in}：入口乾球温度[℃]

TA_{out}：出口乾球温度[℃]

⑨　奥行き方向の必要列数の計算

$N = A_{th}/(KF \times C_{wsh} \times FA \times \Delta t_m)$

ここで、

N：奥行き方向の必要列数[列]

A_{th}：全冷却能力[W]

KF：伝熱係数[W/m^2・K・列]

C_{wsh}：濡面係数[-]

FA：冷却コイルの正面面積[m^2]

⑩　収束計算によるコイル処理熱量（顕熱）の決定

算出コイル列数≒設定コイル列数となるまで収束計算を行い、コイル除去熱量を決定

（収束判定：列数誤差≦0.01）$N - RW \leqq 0.01$

ここで、

N：奥行き方向の必要列数[列]

RW：コイル列数[列]

算出コイル列数≒設定コイル列数の場合、計算終了。

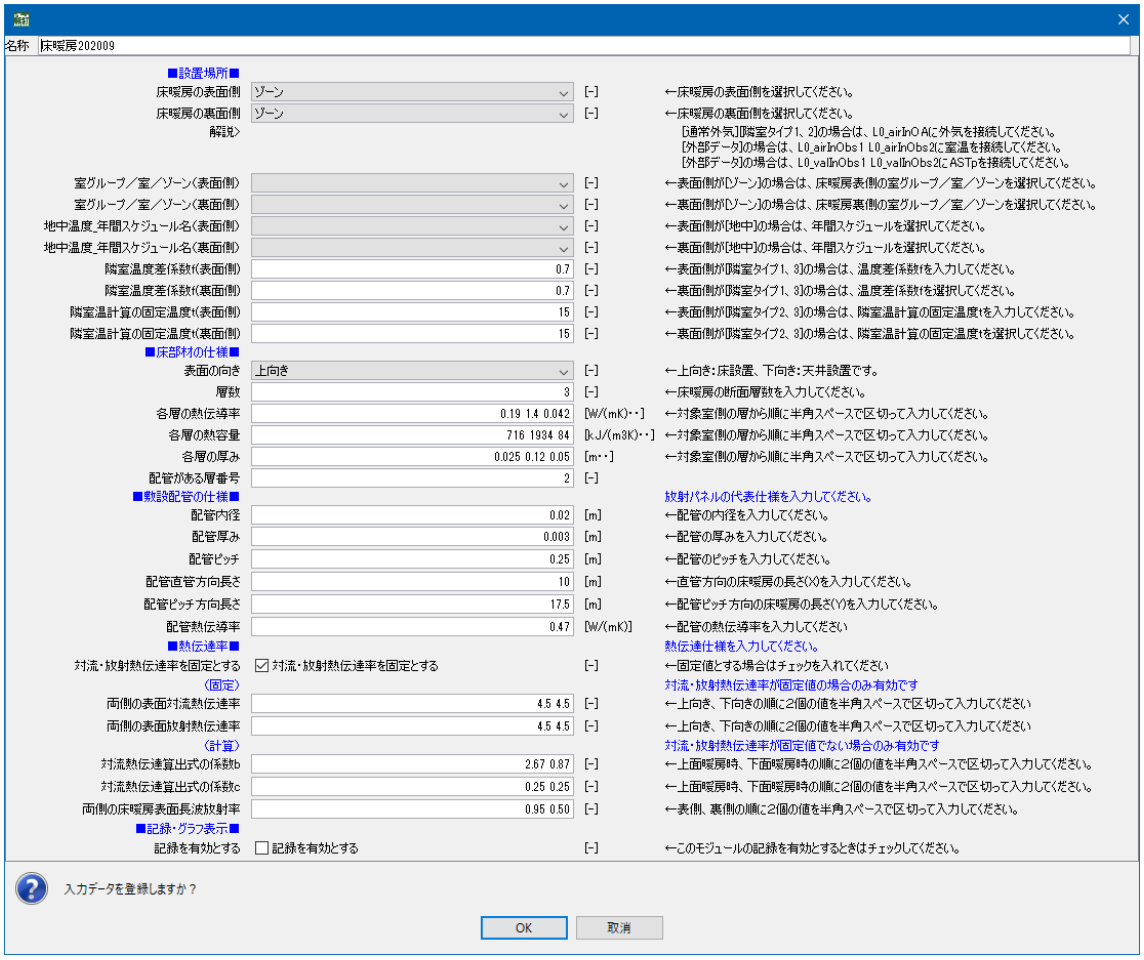
算出コイル列数≠設定コイル列数の場合、①に戻り、前時点の算出列数から求めたコイル処理熱量（顕熱）を用いて再計算。

「温水式床暖房」（場所：空調・換気設備／空調機）

モジュール名	床暖房
クラス	RadFloorHeaterModule2019

（1）入力画面

・スペック



（２）モジュールの概要

差分法によるスラブ内部温度及び表面温度の計算[文 1)]を行う。一次元の熱流のみを対象とし、一様な面発熱を仮定することで、発熱面温度を算出する。躯体埋め込みを想定した温水床暖房(冷房利用も可能)モジュールである。モジュールの概要を図１に示す。

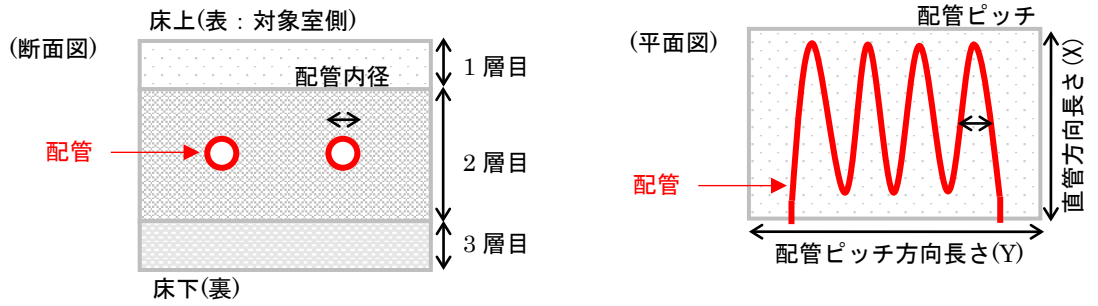


図１　モジュールの概要

（３）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を図２及び次項より示す。

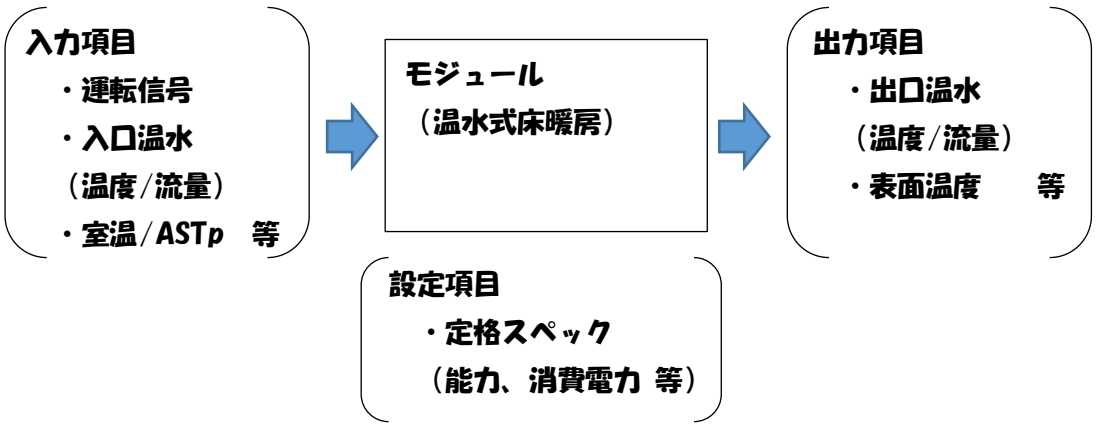


図２　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 2 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	―	―	[-]	―	―	○	複数台設定する場合は変更が必要。
①	■設置場所■								
1	床暖房の表面側	String(ゾーン、通常外気、地中、外部データ（室温：LO_airInObs1 ASTp：LO_valInObs1)、隣室タイプ 1、隣室タイプ 2、隣室タイプ 3)	―	ゾーン	[-]	―	―	◎	[外部データ(‥]を選択した場合は、LO_airInObs1 又は 2 に室温、LO_valInObs 又は 2 に床暖房の表面から見た室内表面温度の面積加重平均値 ASTp[℃]を接続してください。 [通常外気]を選択した場合は、LO_airInOA に外気を接続してください。
2	床暖房の裏面側	String(ゾーン、通常外気、地中、外部データ（室温：LO_airInObs2 ASTp：LO_valInObs2)、隣室タイプ 1、隣室タイプ 2、隣室タイプ 3)	―	ゾーン	[-]	―	―	◎	
[ゾーン]を選択した場合									
3	室グループ/室/ゾーン（表面側）	String	―	―	[-]	―	―	○	表面側又は裏面側が[ゾーン]の場合は、室グループ／室／ゾーンを選択してください。
4	室グループ/室/ゾーン（裏面側）	String	―	―	[-]	―	―	○	
[地中]を選択した場合									
5	地中温度_年間スケジュール名（表面側）	String	―	―	[-]	―	―	○	表面側又は裏面側が[地中]の場合は、年間スケジュールを選択してください。
6	地中温度_年間スケジュ	String	―	―	[-]	―	―	○	

	ール名（裏面側）								
[隣室温度差係数 1,2,3]を選択した場合									
5	隣室温度差係数 f(表面側)	String	―	0.3	[-]	―	―	○	隣室タイプ 1,3 の場合は、温度差係数を入力してください。
6	隣室温度差係数 f(裏面側)	String	―	0.3	[-]	―	―	○	
5	隣室温度差係数の固定温度(表面側)	String	―	15	[-]	―	―	○	隣室タイプ 2,3 の場合は、隣室温計算の固定温度を入力してください。
6	隣室温度差係数の固定温度(裏面側)	String	―	15	[-]	―	―	○	
②	■床部材の仕様■								
7	表面の向き	String (上向き,下向き)	―	530	[-]	―	―	◎	上向き：床設置、 下向き：天井設置
8	層数	Int	M	3	[-]	―	1	◎	床暖房の断面層数
9	各層の熱伝導率	String	λ[]	0.19 1.4 0.042	[W/mK …]	―	―	◎	対象室側の層から順に半角スペースで区切って入力
10	各層の熱容量	String	Cρ[]	716 1934 84	[kJ/(m3K) …]	―	―	◎	
11	各層の厚み	String	ΔXρ[]	0.025 0.12 0.05	[m…]	―	―	◎	
12	配管がある層番号	int	p	2	[-]	―	1	◎	配管がある床部材の層番号
③	■敷設配管の仕様■								
13	配管内径	double	D	0.02	[m]	1	0	△	配管の内径
14	配管厚み	double	t	0.003	[m]	1	0	△	配管の厚み
15	配管ピッチ	double	Wpit	0.25	[m]	1	0	△	配管のピッチ
16	配管直管方向長さ	double	Lh	10	[m]	1000	0	◎	直管方向の床暖房の長さ(X)
17	配管ピッチ方向長さ	double	Lw	17.5	[m]	1000	0	◎	配管ピッチ方向の床暖房の長さ(Y)
18	配管熱伝導率	double	λp	0.47	[W/(mK)]	―	0	△	配管の熱伝導率
④	■熱伝達率■								
19	対流・放射熱伝達率を固定とする	Boolean	―	False	[-]	―	―	○	固定値とする場合は True
固定値=True の場合									

20	両側の表面対流熱伝達率	String	α_c	4.5 4.5	[(W/㎡K) (W/㎡K)]	―	―	△	上向き、下向きの順に２個の値を半角スペースで区切る
21	両側の表面放射熱伝達率	String	α_r	4.5 4.5	[(W/㎡K) (W/㎡K)]	―	―	△	
固定値=False の場合									
22	対流熱伝達算出式の係数 b	String	B	0.31 0.25 0.32	[-]	―	0	○	（計算）
23	対流熱伝達算出式の係数 c	String	C	0.28 0.14 1.78	[-]	―	0	○	
24	両側の表面長波放射率	String	$\varepsilon_0, \varepsilon_1$	0.95 0.50	[-]	―	0	○	表側、裏側の順に２個の値を半角スペースで区切る
⑤	■記録・グラフ表示■								
25	グラフを表示する	Boolean	―	False	[-]	―	―	―	グラフを表示するときはチェック
26	最大同時表示ステップ数	int	―	100	[-]	―	―	△	グラフに同時表示する最大ステップ数
27	記録を有効とする	Boolean	―	False	[-]	―	―	○	このモジュールの記録を有効とするときはチェック

（5）シーケンス接続（図 2 における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	運転状態	L1_swcIn	ー	ー	制御	On/Off 信号	入口	
2	空調モード	L1_modIn	ー	ー	制御	制御モード	入口	
3	入口温水	L0_watIn	Twi, Gw	℃,g/s	状態	水	入口	給湯機から供給される温水
4	対象室側(表)の室温[℃]	L0_airInObs1	RM_DB	ー	状態	空気	入口	外部データから取り込む=True の場合のみ有効
5	対象室側(表)の床暖房表面から見た室内表面温度の面積加重平均値 ASTp [℃]	L0_valInObs1	RM_ASTp	℃	状態	double 値	入口	
6	床下側(裏)の室温[℃]	L0_airInObs2	RM_DB	ー	状態	空気	入口	
7	床下側(裏)の床暖房表面から見た室内表面温度の面積加重平均値 ASTp [℃]	L0_valInObs2	RM_ASTp	℃	状態	double 値	入口	
8	記録	L2_recOut	ー	ー	記録	メモリ	出口	空調記録モジュールへ運転状態等を出力する。
9	出口温水	L0_watOut	Two, Gw	℃,g/s	状態	水	出口	給湯機への還り温水
10	床暖房の対象室側(表)の表面温度[℃]	L0_valOutObs	Ta(0)	℃	制御	double 値	出口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
2	処理顕熱量	熱量	W	負荷
3	床暖房_出口水温	温度	℃	出口
4	床暖房_入口水温	温度	℃	入口
5	床暖房_入口流量	流量	g/s	入口
6	床暖ｼｽﾃﾑ_対流熱_表	熱量	W	My
7	床暖ｼｽﾃﾑ_放射熱_表	熱量	W	My
8	床暖ｼｽﾃﾑ_顕熱_表	熱量	W	My
9	床部材_表側表面温度 ts	温度	℃	My
10	床暖房_表側室温 ta	温度	℃	My
11	床暖房_表側 ASTp	温度	℃	My

（７）計算方法

1）床暖房部材の計算

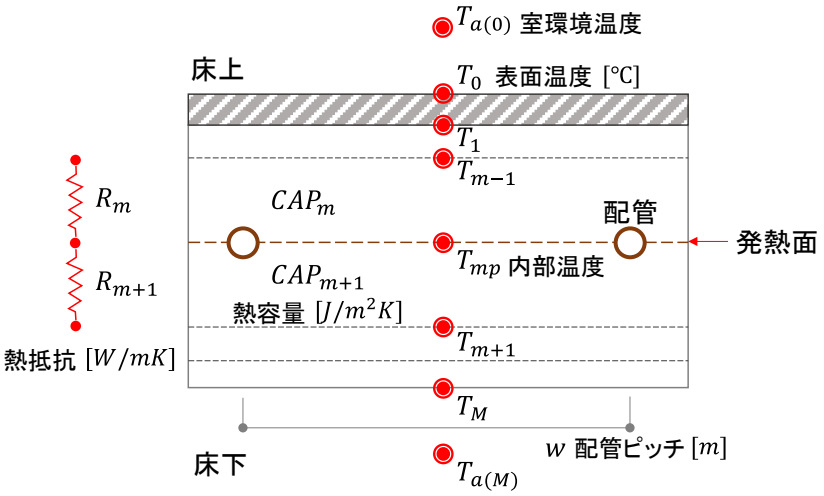


図３　床暖房の計算モデル断面

床暖房の計算モデル断面を図 3 に示す。m は接点番号であり、最大が M 層とする。

接点 m 及び m=0,M における熱容量 CAP と熱抵抗 R を表 1 に示す。尚、$c_{\rho m}$は、接点 m の対象室側の部材の熱容量[J/m³K]、Δx_mは、接点 m の対象室側の部材の厚み[m]、λ_mは、接点 m の対象室側の部材の熱伝導率[W/mK]である。

表１　各層の熱容量 CAP と熱抵抗 R

接点	m=0 (対象室側表面)	m = m	m =M (床下側裏面)
熱容量　$CAP[J/m^2K]$	0	$c_{\rho m} \times \Delta x_m$	0
熱抵抗　$R[W/mK]$	$1/\alpha_0$	$\Delta x_m/\lambda_m$	$1/\alpha_M$

差分法により、スラブ内部及び表面温度は、式(1)にて算出可能である。尚、接点 m = p の部分に温水配管による一様な面発熱を仮定しており、一次元熱流と仮定している。

$$\frac{\partial T}{\partial t} = \frac{1}{0.5(CAP_m + CAP_{m+1})}\left\{\frac{1}{R_{m+1}}(T_{m+1} + T_m) - \frac{1}{R_m}(T_m - T_{m-1}) + \frac{H_p}{A_{床}}\right\} \cdots (1)$$

$A_{床}$: *床暖房面積*$[m^2](L_w \times L_h)$

加熱量 H_p は、管表面平均温度 T_{px} と温水入口温度 T_{wi} との関数であり式(2)で表される。

$$H_p = c_w G_w \varepsilon_{PNL}(T_{wi} - T_{PX}) \cdots (2)$$

c_w: *水の比熱*$[J/kgK]$,

G_w: *水量*$[kg/s]$,

ε_{PNL}: *配管から発熱面への熱通過有効度* [–],

T_{px}: *管表面平均温度* [℃]

温水配管のピッチが非常に小さい場合は、T_{PX}=T_m であるが、一般的に 10~50cm の間隔が想定されるため、フィン効率 η_p を用いる。尚、T_m とは、発熱面上の平均温度を意味する。フィン効率を用いると、発熱量と上下面への放熱量には、式(3)の関係がある。

$$\frac{H_p}{A_{床}} = \eta_p \left\{ \frac{1}{R_m}(T_{PX} - T_{m-1}) + \frac{1}{R_{m+1}}(T_{PX} - T_{m+1}) \right\} = \frac{1}{R_m}(T_m - T_{m-1}) + \frac{1}{R_{m+1}}(T_m - T_{m+1}) \cdots (3)$$

温水による加熱量は、式(2)に示す通り、T_{PX} の関数となっているが、式(3)にて式(4)の関係が得られる。

$$T_m = T_m - \frac{\frac{H_p}{A_{床}}}{\frac{1}{R_m} + \frac{1}{R_{m+1}}} \left(\frac{1}{\eta_p} - 1 \right) \cdots\cdots (4)$$

$(m = p)$,

これを式(3)に代入することで、加熱量を T_m の関数として表す。

$$\frac{H_p}{A_{床}} = \frac{\epsilon_{PNL}(C_w G_w)}{A_{床}}(T_{wi} - T_m) = W_p(T_{wi} - T_m) \cdots\cdots (5)$$

$$\epsilon_{PNL} = \frac{\varepsilon_{PX}}{1 + \left(\frac{\varepsilon_{PX} C_w G_w}{A C_f} \right) \left(\frac{1}{\eta_p} - 1 \right)} \cdots (6-1)$$

$$W_p = \frac{\varepsilon_{PNL}(C_w G_w)}{A_{床}} \cdots (6-2)$$

$$C_f = \frac{1}{R_m} + \frac{1}{R_{m+1}} \cdots (7)$$

フィン効率は、発熱面(m=p)から上・下の接点 m-1、m+1 への放熱を考え、また、管 1 本あたりの片面当たりの放熱面は配管ピッチ w_{pit}[m]で代表することとすれば式(8)で示される。

$$\eta_P = \frac{1}{w_{pit}} \left\{ D + (w_{pit} - D) \frac{tanhZ}{Z} \right\} \cdots (8)$$

$$Z = \frac{w_{pit} - D}{2} \sqrt{\frac{C_f}{\lambda \delta}} \cdots (9)$$

ここで λ は配管がある層の熱伝導率であり、発熱面の上下は同じ部材とする。また、δは、フィン厚[m]であり、ここでは管径 D と等しいと仮定する。ε_{PX}は、管内の温水から管外部までの

熱貫流率 K_p を用いて式(10)で計算する。

$$\varepsilon_{PX} = 1 - \exp\left(-\frac{K_P A_{床}}{C_w G_w}\right) \cdots (10)$$

$$\frac{1}{K_P} = \frac{A_{床}}{L}\left(\frac{1}{\pi D \alpha_w} + R_b\right) \cdots (11)$$

ここでは、α_w は管内熱伝達率[W/m²K]、R_b はパネルと管との間の単位管当たり熱抵抗（管材の熱抵抗と接触熱抵抗の合計）[W/mK]、L は管の全長[m]である。

$$\alpha_w = 1057(1.352 + 0.0198 \times T_{wi}) \times {G_w}^{0.8}/D^{0.2} \cdots (12)$$

$$R_b = t/(\lambda_p \pi D) \cdots (13)$$

$$L = \left(\frac{L_w}{W_{pit}}\right) \times L_h \cdots (14)$$

λ_p: *配管の熱伝導率*$[W/mK]$

t: *配管の厚み*$[m]$

(1)　前進差分法による節点温度の計算

前進差分法を用いて発熱面温度を式(15)～(19)より算出する[文1)]（発熱面以外の温度は$P_c = 0$）。

$$T_{mp} = u_{Lm} T^*_{m-1} + \left(1 - u_{Lm} - u_{Rm} - P_c W_p\right) T^*_m + u_{Rm} T^*_{m+1} + P_c W_p T^*_{wi} \cdots (15)$$

$$P_c = \frac{\Delta t}{0.5\left(CAP_m + CAP_{m,+1}\right)} \cdots (16)$$

$$W_p = \frac{\varepsilon_{PNL}(C_w G_w)}{A_{床}} \cdots (17)$$

$$u_{Lm} = \frac{\Delta t}{0.5(CAP_m + CAP_{m+1})R_m} \cdots (18)$$

$$u_{Rm} = \frac{\Delta t}{0.5(CAP_m + CAP_{m+1})R_{m+1}} \cdots (19)$$

Δt: *計算時間間隔*$[sec]$

また、式(20)の安定条件を満足しない場合は、モジュール内で短い時間間隔で繰り返し計算を行うこととした。

$$\Delta t_m \leq \frac{0.5(CAP_m + CAP_{m+1})}{1/R_m + 1/R_{m+1}} \cdots (20)$$

(2)　室の環境温度 Ta(0)(及び Ta(M))の計算

環境温度 Ta(0)は式(21)、総合熱伝達率 α(0)は式(22)にて求める。室温 RM_DB と床暖房パネルから見た室内の平均放射温度 RM_AST_p は、建築側プログラムより取得する。

$$\mathrm{Ta}(0) = RM\_DB\frac{\alpha_c}{\alpha(0)} + RM\_AST_p\frac{\alpha_r}{\alpha(0)} \cdots (21)$$

$$\alpha(0) = \alpha_c + \alpha_r \cdots (22)$$

α_c: *対流熱伝達率*$[W/m^2K]$

α_r: *放射熱伝達率*$[W/m^2K]$

尚、屋外の場合は Ta(0)又は Ta(M)を外気とし、 α(0)又は α(M)は 23W/㎡ K とする。地中温度の場合は T(0)又は T(M)を地中温度とする。隣室温度差係数の場合は、Ta(0)又は Ta(M)を隣室温度とする。尚、隣室温度は、下式の通り算出する。

隣室タイプ 1：隣室温=隣室温度差係数 f×外気温＋(1-f)×自室温

隣室タイプ 2：外気温+固定温度

隣室タイプ 3：f×固定温度＋(1-f)×自室温

床暖表面から室への対流熱伝達率 α_c は、「対流・放射熱伝達率を固定とする」が True の場合は、固定的な値（パラメータとして指定）を用いることとし、False の場合は、床暖表面温度 T_0 と室温 RM_DB の関数の式(23)[文 2)]にて算出する。c 値及び b 値を表 2 に示す。

$$\alpha_c = c(T_0 - RM\_DB)^b \cdots (23)$$

表 2　対流熱伝達率算出のための b、c 値

部位	b 値	c 値
上面暖房 若しくは 下面冷房	0.31	2.13
下面暖房 若しくは 上面冷房	0.25	0.134
壁面暖房 若しくは 壁面冷房	0.32	1.78

床暖表面から室への放射熱伝達率 α_r は、「対流・放射熱伝達率を固定とする」が True の場合は、固定的な値（パラメータとして指定）を用いることとし、False の場合は、無限平行 2 平面における放射熱伝達率[W/m²K]と仮定[文 3)]し、式(24)にて算出する。

$$\alpha_r = 5.7 \times \frac{1}{\frac{1}{eps_1} + \frac{1}{eps_2} - 1} \cdots (24)$$

eps_1：*床の長波放射率[-]（入力値（例：0.9））*

eps_2：*相対するもう一方の面の長波放射率[-](0.9)*

尚、対流熱伝達率と放射熱伝達率は各計算ステップにて更新する。

(3)　室への対流熱伝達量と放射熱伝達量の計算

(1)にて算出した床表面温度 T_0 と対流・放射熱伝達率より、床暖房の対流熱伝達量 Wc[W]と放熱熱伝達量 Wr[W]を式(25),(26)にて算出する。

$$W_c = \alpha_c(T_0 - RM\_DB) \times A_{床} \cdots (25)$$

$$W_r = \alpha_r\left(T_0 - RM\_AST_p\right) \times A_{床} \cdots (26)$$

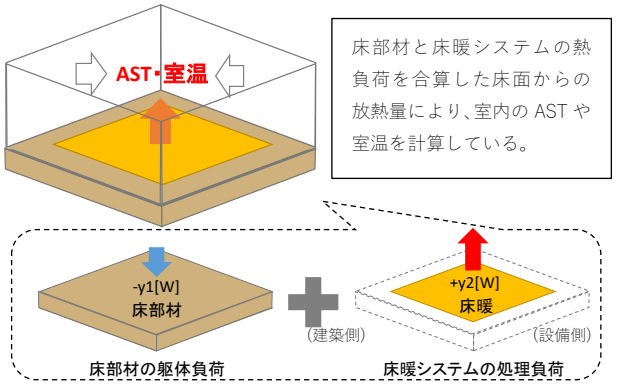
(4)　出口水温 T_{wo} の計算

加熱量 H_p と水量 Gw より、式(27)にて求める。

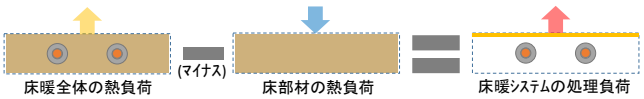
$$T_{wo} = T_{wi} - H_p/(\mathrm{G}_w \times 4.18605) \cdots (27)$$

(5)　建築側プログラムへの受け渡しデータ

建築側プログラムでは、本モジュールで計算した放射、対流熱量を受け取り、建築床部材の躯体負荷と合成することで室温や AST 等の計算を行っている(図 4(a))。そのため、本モジュールでは、配管のある床部材(床暖全体)と配管の無い床部材の計算を行い、その差分を建築側へ受け渡す方法とした(図 4(b))。尚、透過日射や内部発熱から受ける放射性分は建築側で考慮している。



(a) 建築側の室温・AST



(b)モジュールが建築側へ受け渡す熱負荷

図 4　計算イメージ

(6)　全体の計算フロー

図 5 のような計算順序にて計算を行う[文 4)]。各時間ステップの処理は、以下の流れとなる。

1) 建築側で、前ステップにて床暖房から見た平均表面温度(RM_ASTp)が計算される。

2) 建築側で、前ステップにて現ステップの室内温度(RM_DB)が計算される。

3）建築側から床暖房へ、前ステップの RM_ASTp と現ステップの RM_DB を受け渡す。

4）床暖房で、対流熱伝達量と放射熱伝達量を計算する。

5）床暖房から建築側へ、対流熱伝達量[W]と放射熱伝達量[W]、床暖房パネルの向き（上向き/下向き）情報を受け渡す。

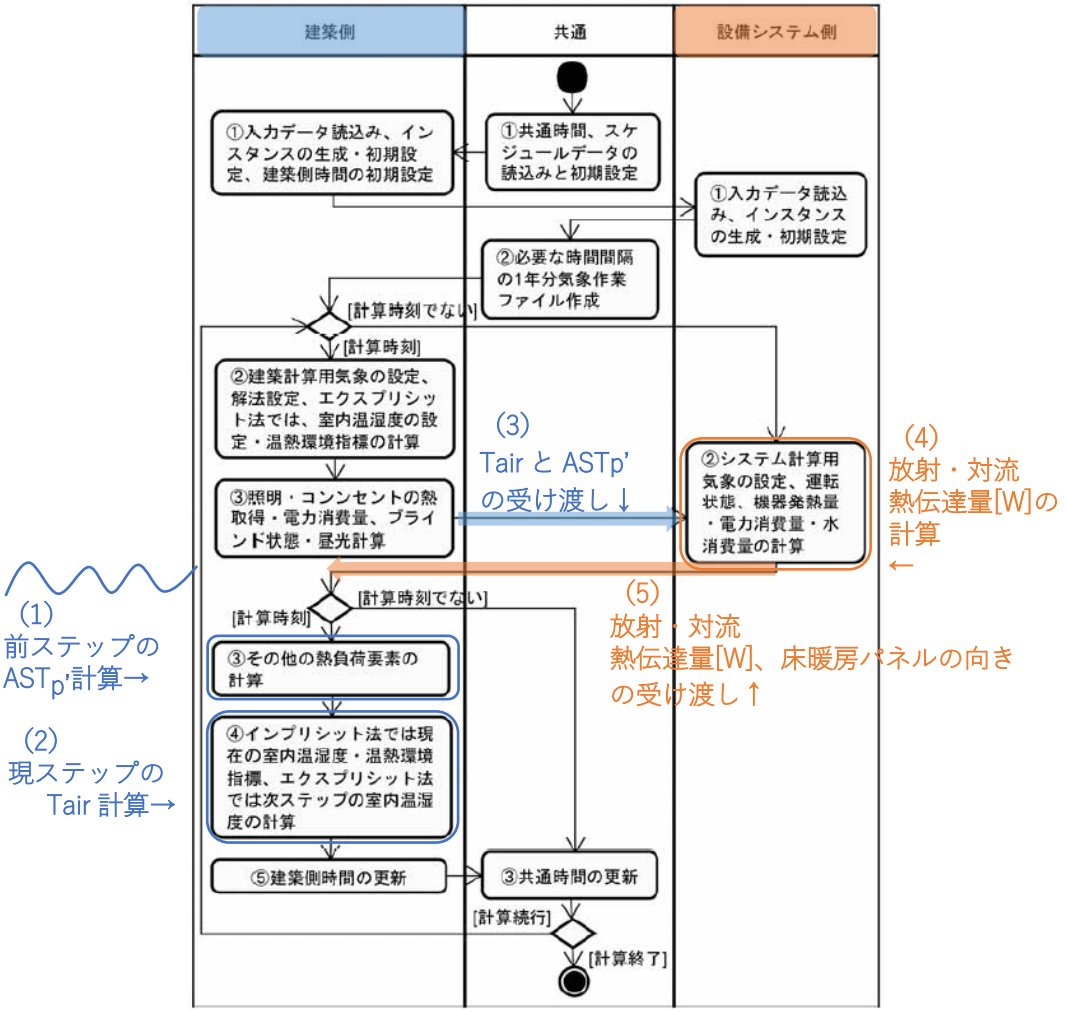


図 5 建築・設備システムの計算順序[文 4)]

文献

1）パソコンによる空気調和計算法,pp134-141,224-228 宇田川光弘著

2）ASHRAE Handbook, System and Equipment(SI),CHAPTER 6 (2004),p6.4

3）建築環境工学 熱環境と空気環境, 朝倉書店, 2009, 宇田川他

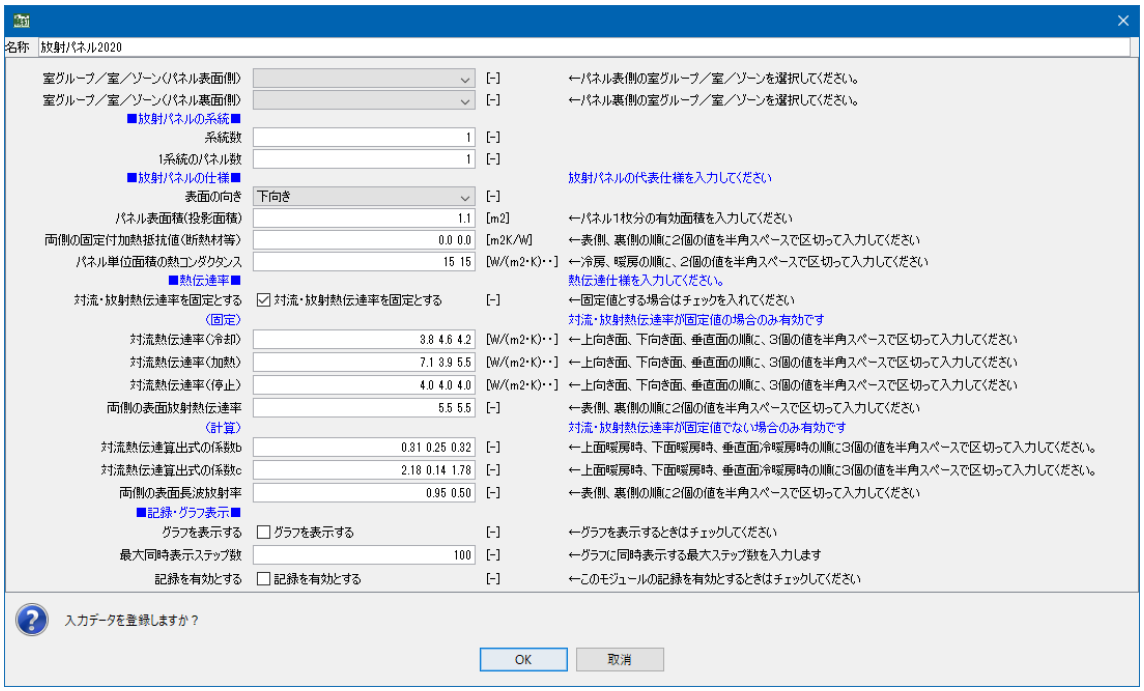
4）外皮・躯体と設備・機器の総合エネルギーシミュレーションツール「BEST」の開発（その 22）建物側プログラムの概要空気調和・衛生工学会大会学術講演論文集, 2008, 郡

「温水式床暖房」（場所：設備 2015／空調機 2015）

モジュール名	放射パネル
クラス	RadPanelModule2019

（1）入力画面

・スペック



（2）モジュールの概要[文 1)]

躯体埋め込み式ではない、熱容量の小さな放射パネルで、かつ両側が空気に接している場合を対象とする。住宅用の床暖房用温水パネルのように床等の部位に接して設けられる場合には、その部位の熱容量が無視できるときには、部位全体を、熱抵抗を有するパネルと見なすことにより適用できるが、部位の熱容量が無視できない場合には適用できない。一方、室内に独立して設けられる放射パネルも対象としている(図 1)。

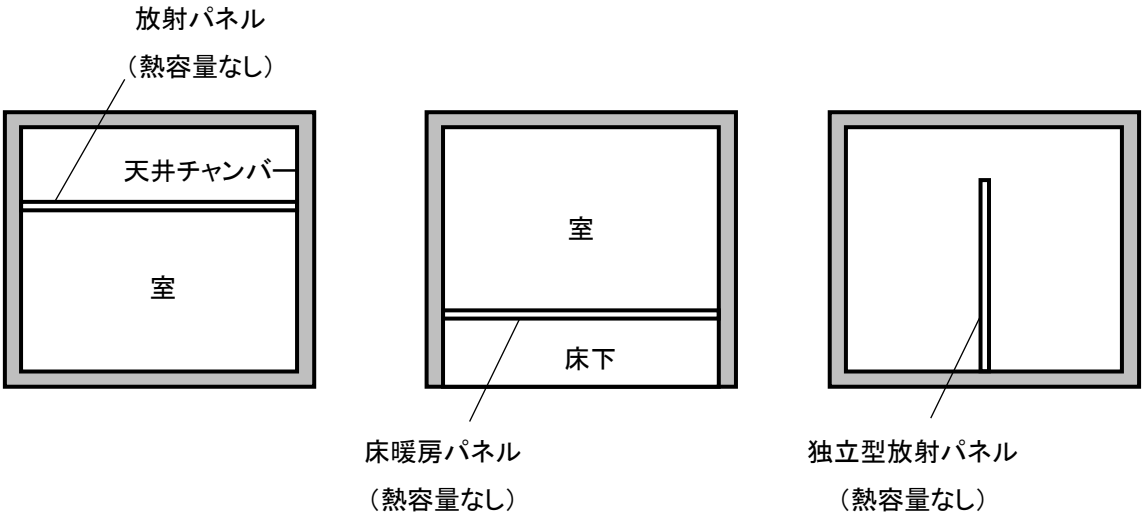


図 1　対象とする放射パネル

放射パネルのモデル化の方法として、配管やパネルの材質・形状から演繹的に伝熱量を計算する方法もあるが[文 2)]、ここで採用した方法は、パネルの放熱特性を示した試験結果から帰納的にマクロな伝熱係数を算出するものである[文 3 等)]。この方法によったのは、パネルの詳細仕様を計算者が知ることは難しく、パネルメーカから入手できる特性データから入力値を決定できる方が実用的と考えたためである。

パネルと隣接空気および他表面との間の伝熱モデルを図 2 に示す。隣接する 2 つのゾーンから室温、パネル用平均表面温度 ASTp を受け取り、また、冷温水入口温度・流量を与条件として、各ゾーンへの対流・放射熱量と冷温水出口水温を算出する。

前処理において、特性データ（実測値）をもとに冷温水からパネルに至る実質的な熱抵抗値 Rw-p を推定した後、その熱抵抗値を用いて時々刻々の計算を行う。なお、パネルの断面方向・平面方向の温度分布、及び流速の変化によるパネル熱コンダクタンスの変化は無視できるものとする。

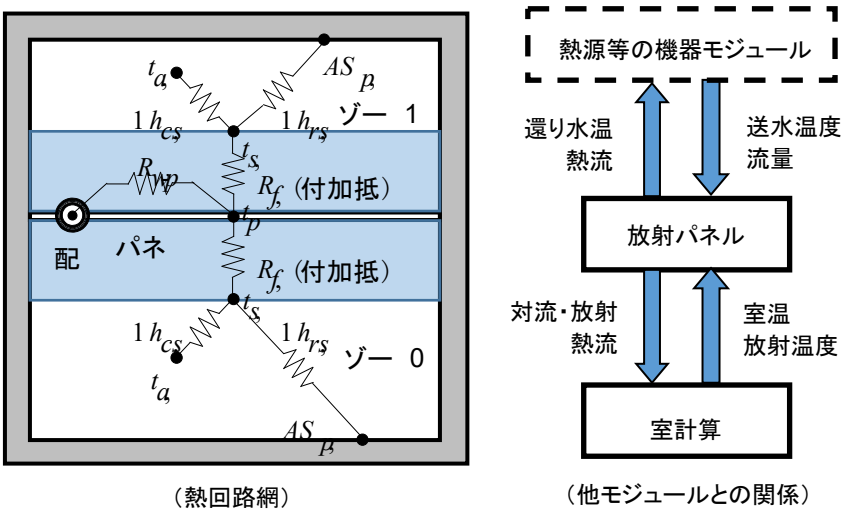


図２　モジュールの概要

（３）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を図３及び次項より示す。

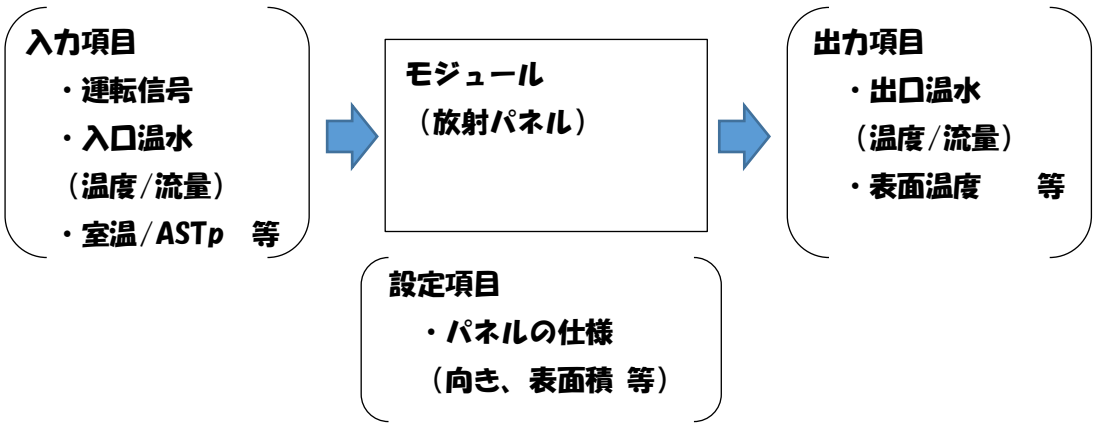


図２　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 2 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	―	―	[-]	―	―	○	複数台設定する場合は変更が必要。
①	■設置するゾーン■								
2	室グループ/室/ゾーン（パネル表面側）	String	$t_{a,0}$,, $AST_{p,0}$,	―	[-]	―	―	◎	室温やパネルの表面から見た室内表面温度の面積加重平均値 ASTp[℃]をシーケンス接続にて読み込む場合は、”外部データから読み込む”にチェックを入れ、これ以外の場合は、“室グループ/室/ゾーン（パネル表面側）及び（パネル裏面側）”にてパネル上下のゾーンを選択する。
3	室グループ/室/ゾーン（パネル裏面側）	String	$t_{a,1}$, $AST_{p,1}$	―	[-]	―	―	◎	
②	■放射パネルの系統■								
4	系統数	Int	N_p	1	[-]	―	0	◎	
5	1 系統のパネル数	Int	N_s	1	[-]	―	0	◎	
③	■放射パネルの仕様■								
6	表面の向き	String（下向き、上向き、垂直面）	$face_{,0}$, $face_{,1}$	下向き	[-]	―	―	◎	
7	パネル表面積(投影面積)	double	A_p	1.1	[m2]	―	0	◎	1 ユニットあたりのパネル表面積
8	両側の固定付加熱抵抗値（断熱材等）	String	$R_{f,0}$, $R_{f,1}$	0.0 0.0	[m2K/W]	―	0	○	断熱材で裏打ちされた放射パネル等に設定。
9	パネル単位面積の熱コンダクタンス	double	$K_{w\text{-}otp,t}$	15	[W/(m2･K)]	―	0	○	試験時(冷房)の熱コンダクタンス（冷房能力__((平均)室温-平均冷水温度）
④	■熱伝達率■								

10	対流・放射熱伝達率を固定とする	Boolean	-	True	[-]	―	-	○	
11	対流熱伝達率（冷却）	String	$h_{c,s}$	3.8 4.6 4.2	[W/(m2･K)・・]	―	0	○	（固定）パネル面の熱伝達率[W/(m2･K)]（冷却、加熱に対する、上向面、下向面、垂直面の計 6 つの値）
12	対流熱伝達率（加熱）	String		7.1 3.9 5.5	[W/(m2･K)・・]	―	0	○	
13	対流熱伝達率（停止）	String		4.0 4.0 4.0	[W/(m2･K)・・]	―	0	○	
14	両側の表面放射熱伝達率	String	$h_{r,s}$	5.5 5.5	[W/(m2･K)・・]	―	0	○	（固定）パネル面の放射熱伝達率（表側、裏側の 2 つの値）
15	対流熱伝達算出式の係数 b	String	B	0.31 0.25 0.32	[-]	―	0	○	（計算）
16	対流熱伝達算出式の係数 c	String	C	0.28 0.14 1.78	[-]	―	0	○	
10	両側の表面長波放射率	String	$\varepsilon_0, \varepsilon_1$	0.95 0.50	[-]	―	0	○	
⑤	■記録・グラフ表示■								
14	グラフを表示する	Boolean	―	False	[-]	―	―	―	グラフを表示するときはチェック
15	最大同時表示ステップ数	int	―	100	[-]	―	―	―	グラフに同時表示する最大ステップ数
16	記録を有効とする	Boolean	―	False	[-]	―	―	○	このモジュールの記録を有効とするときはチェック

（5）シーケンス接続（図 2 における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	－	－	記録	メモリ	出口	記録モジュールへ運転状態等を出力する。
2	運転	L1_swcIn	－	－	制御	On/Off 信号	入口	On/Off
3	空調モード	L1_modIn	mode	－	制御	制御モード	入口	冷房、暖房
4	入口冷温水	L0_watIn	Twi, Gw	℃,g/s	状態	水	入口	熱源から供給される冷温水
5	出口水	L0_watOut	Two, Gw	℃,g/s	状態	水	出口	還り水
6	パネル（表）の表面温度[℃]	L0_valOutObs	$Ts_{,0}$	℃	状態	Double 値	出口	表面温度を制御対象とする場合に使用する。
7	対象室側(表)の室温[℃]	L0_airInObs1	$Ta_{,0}$	℃	状態	空気	入口	外部データから取り込む=True の場合のみ有効
8	対象室側(表)のパネル表面から見た室内表面温度の面積加重平均値 ASTp [℃]	L0_valInObs1	$ASTp_{,0}$	℃	状態	Double 値	入口	
9	隣室側(裏)の室温[℃]	L0_airInObs2	$Ta_{,1}$	℃	状態	空気	入口	
10	隣室側(裏)のパネル表面から見た室内表面温度の面積加重平均値 ASTp [℃]	L0_valInObs2	$ASTp_{,1}$	℃	状態	double 値	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
2	放射パネル_入口水温	温度	℃	入口
3	放射パネル_入口流量	流量	g/s	入口
4	放射パネル_出口水温	温度	℃	出口
5	放射パネル_対流熱_表	熱量	W	My
6	放射パネル_対流熱_裏	熱量	W	My
7	放射パネル_対流熱_合計	熱量	W	My
8	放射パネル_放射熱_表	熱量	W	My
9	放射パネル_放射熱_裏	熱量	W	My
10	放射パネル_放射熱_合計	熱量	W	My
11	放射パネル_顕熱_表	熱量	W	My
12	放射パネル_顕熱_裏	熱量	W	My
13	放射パネル_顕熱_合計	熱量	W	My
14	放射パネル_表側室温 ta	温度	℃	My
15	放射パネル_裏側室温 ta	温度	℃	My
16	放射パネル_表側表面温度 ts	温度	℃	My
17	放射パネル_裏側表面温度 ts	温度	℃	My
18	放射パネル_表側環境温度 OTp	温度	℃	My
19	放射パネル_裏側環境温度 OTp	温度	℃	My
20	放射パネル_表側 ASTp	温度	℃	My
21	放射パネル_裏側 ASTp	温度	℃	My
22	処理熱量	熱量	W	負荷
23	単位面積処理熱量	熱量	W/㎡	負荷

（７）計算方法[文 1)]

１．熱交換モデル(図 3)

放射パネル 1 直列系統あたりの放熱量 q_p は下式で表わされる。

$$q_p = N_s A_p / R_{w-otp} \cdot \Delta t_{w-otp} = c_w \left(t_{w,i} - t_{w,o} \right) \qquad [\mathrm{W}] \qquad \cdots (1)$$

ここで、

$$R_{w-otp} = R_{w-p} + \frac{1}{1/R_0 + 1/R_1} \qquad [(\mathrm{m}^2 \cdot \mathrm{K})/\mathrm{W}] \qquad \cdots (2)$$

$$\Delta t_{w-otp} = \frac{t_{w,i} - t_{w,o}}{\ln\left\{ \left(t_{w,i} - ot_{p,2} \right) / \left(t_{w,o} - ot_{p,2} \right) \right\}} \qquad [\mathrm{K}] \qquad \cdots (3)$$

$$c_w = c_{pw} m_w / N_p \qquad [\mathrm{W/K}] \qquad \cdots (4)$$

$$R_0 = R_{f,0} + \frac{1}{h_{cs,0} + h_{rs,0}} \qquad [(\mathrm{m}^2 \cdot \mathrm{K})/\mathrm{W}] \qquad \cdots (5)$$

$$ot_{p,2} = \frac{1/R_0 \cdot ot_{p,0} + 1/R_1 \cdot ot_{p,1}}{1/R_0 + 1/R_1} \qquad [\mathrm{°C}] \qquad \cdots (6)$$

$$ot_{p,0} = \frac{h_{cs,0} t_{a,0} \cdot + h_{rs,0} AST_{p,0}}{h_{cs,0} + h_{rs,0}} \qquad [\mathrm{°C}] \qquad \cdots (7)$$

また、R_1, $ot_{p,1}$ は、それぞれ式(5), (7)と同様である。

式(1), (2)の R_{w-otp} は、水から 2 ゾーン全体の環境温度 $ot_{p,2}$ に至る熱抵抗であり、R_0, R_1 は、パネルから各ゾーンの環境温度 $ot_{p,0}$, $ot_{p,1}$ に至る熱抵抗である。式(1), (3)の Δt_{w-otp} は、水温と 2 ゾーン全体の環境温度 $ot_{p,2}$ との間の対数平均温度差である。パネルを横断する方向のパネル自体の熱抵抗は無視し、$R_{f,0}$, $R_{f,1}$ によってパネルの付加熱抵抗を考慮する（断熱材で裏打ちされた放射パネル等に対応）。

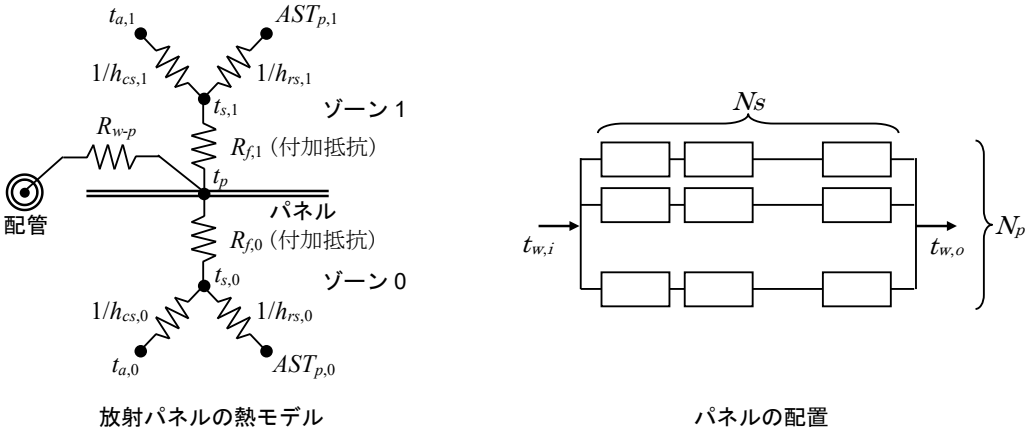


図３　放射パネルの伝熱モデルと配置

２．入出力

放射パネルの計算モデルをモジュール化した場合の入出力を表１に示す。ここで、「パラメータ」とは、パネルの仕様等、時間とともに変化しない固定値であり、一方、「入力」とは一般には時々刻々変化するものである。

また、パラメータ中の試験時の熱コンダクタンス $K_{w\text{-}otp,t}$ とは、図５に示すような性能線図から読み取って設定することを想定している。性能線図の中に、両側空間（居室側と天井裏等）の平均温度を「室温」としている特性がある場合は、そちらの特性を採用する。図４の例では実線の特性を採用し、試験時の「室温」が式(6)の $ot_{p,2}$ に相当することとする。

表１放射パネルモジュールの入出力変数一覧

パラメータ：

A_p：　パネル１枚あたり面積[m^2]

N_p：　パネル並列系統数

N_s：　パネル直列接続数

c_{pw}：　水の比熱[J/(g･K)]

$R_{f,0}$, $R_{f,1}$：　固定付加熱抵抗値[m^2K/W]

e_0, e_1　：　パネル両面の長波放射率[-]

$h_{c,s}$：　パネル面の熱伝達率[W/(m^2･K)]（冷却、加熱に対する、上向面、下向面、垂直面の計６つの値）

$face_{,0}$, $face_{,1}$：両面の向き（'up':上向き、'down':下向き、'vertical':垂直）

$K_{w\text{-}otp,t}$：　試験時(冷房)の熱コンダクタンス（パネル単位面積あたり）[W/(m^2･K)]

$inc_{\_r,0}$, $inc_{\_r,1}$：両側の温度が、性能線図の横軸（室温－水温）の「室温」に含まれるか否かのフラグ

$mode_{,t}$：　試験時の冷暖モード

入力：

$t_{w,i}$：　パネル系統全体の入口水温[℃]

G_w：　パネル系統全体の入口流量[g/s]

$t_{a,0}$, $t_{a,1}$：　室内空気温度（ゾーン 0, 1）[℃]

$AST_{p,0}$, $AST_{p,1}$：パネル用平均表面温度（ゾーン 0, 1）[℃]

$mode$ ：　運転時の冷暖モード（'cooling'：冷房、'heating'：暖房）

出力：

$t_{w,o}$：　パネル系統全体の出口水温[℃]

G_w：　パネル系統全体の出口流量[g/s]

$q_{cs,0}$, $q_{cs,1}$：パネル系統全体からの対流放熱量（ゾーン 0, 1）[W]

$q_{rs,0}$,$q_{rs,1}$：　パネル系統全体からの放射量（ゾーン 0, 1）[W]

$t_{s,0}$, $t_{s,1}$：　パネル表面温度

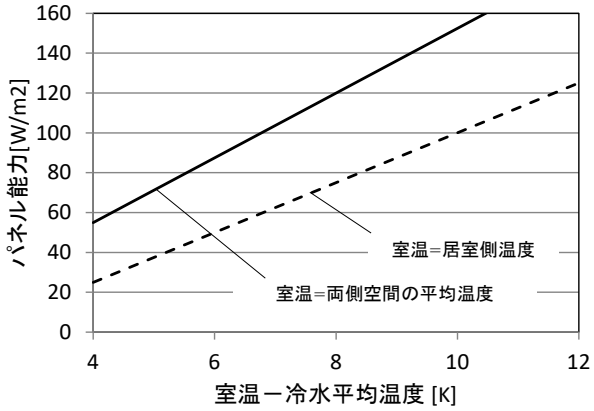


図４　放射パネルの特性データ（例）

対流熱伝達率については、「対流・放射熱伝達率を固定とする」が True の場合は、冷暖モードとパネル面の向きから、熱流方向に応じた固定的な値（パラメータとして指定）を用いることとし、False の場合は、式(A)にて算出する。

$$\alpha_c = C * |T_s - T_a|^B \cdots (A)$$

C：対流熱伝達算出式の係数 c

B：対流熱伝達算出式の係数 b

放射熱伝達率は、「対流・放射熱伝達率を固定とする」が True の場合は、固定的な値（パラメータとして指定）を用いることとし、False の場合は、無限平行 2 平面における放射熱伝達率[W/m^{2}K]と仮定[文 4)]し、式(B)にて算出する。

$$\alpha_r = 5.7 \times \frac{1}{\frac{1}{eps_1} + \frac{1}{eps_2} - 1} \cdots (B)$$

eps_1：床の長波放射率[-]（入力値（例：0.9））

eps_2：相対するもう一方の面の長波放射率[-](0.9)

３．計算手順

(1) 前処理計算

式(2)において $R_{w\text{-}otp} = 1/K_{w\text{-}otp,t}$ とおき、水からパネルに至る等価熱抵抗値 R_{w-p} を算出し記憶する。

$$R_{w-p} = \frac{1}{K_{w-otp,t}} - \frac{1}{1/R_0 + 1/R_1} \quad \cdots (8)$$

R_0, R_1 については、試験時の $h_{cs,0}$, $h_{cs,1}$ を用いて式(5)等から求める。

(2) 時間ループ計算

① 水から環境温度に至る熱抵抗 $R_{w\text{-}otp}$ の算出

式(2)より算出する。R_0, R_1 については、運転時の $h_{cs,0}$, $h_{cs,1}$ を用いて式(5)等から求める。

② パネル 1 直列系統あたり放熱量 q_p の算出

式(1)の第二式＝第三式が成り立つように収束計算を実施し、出口水温 $t_{w,o}$ を求める。求まった $t_{w,o}$ をもとに式(1)より q_p を算出する。

③等価パネル温度 t_p、パネル両側表面温度 $t_{s,0}$, $t_{s,1}$ の算出

式(9), (10)より算出する。$t_{s,1}$ の算出は式(10)と同様である。

$$t_p = ot_{p,2} + \frac{q_p}{N_s A_p} \cdot \frac{1}{1/R_0 + 1/R_1} \qquad \cdots (9)$$

$$t_{s,0} = t_p - \frac{R_{f,0}}{R_0} \cdot \left(t_p - ot_{p,0}\right) \qquad \cdots (10)$$

④ 各ゾーンへの対流・放射熱量の算出

式(11), (12)より各ゾーンへの対流・放射熱量を算出する（$q_{cs,1}$, $q_{rs,1}$ も同様である）。

$$q_{cs,0} = N_p N_s A_p h_{cs,0} \left(t_{s,0} - t_{a,0}\right) \qquad \cdots (11)$$

$$q_{rs,0} = N_p N_s A_p h_{rs,0} \left(t_{s,0} - AST_{p,0}\right) \qquad \cdots (12)$$

文献

1）長井達夫：外皮・躯体と設備・機器の総合エネルギーシミュレーションツール「BEST」の開発 （その 162）空調システム関連の整備状況と放射パネルモジュール, 空気調和・衛生工学会大会学術講演論文集 第 5 巻, pp.37-40, 2015

2）塩谷正樹, 郡公子：スリットを有する天井放射パネルユニットの熱性能に関する研究, 日本建築学会環境系論文集,Vol.78, No.683, pp.31-37, 2013

3）TRNSYS17, Volume 5, Multizone Building modeling with Type56 and TRNBuild, Solar Energy Laboratory, TRANSSOLAR, CSTB, TESS, 2012

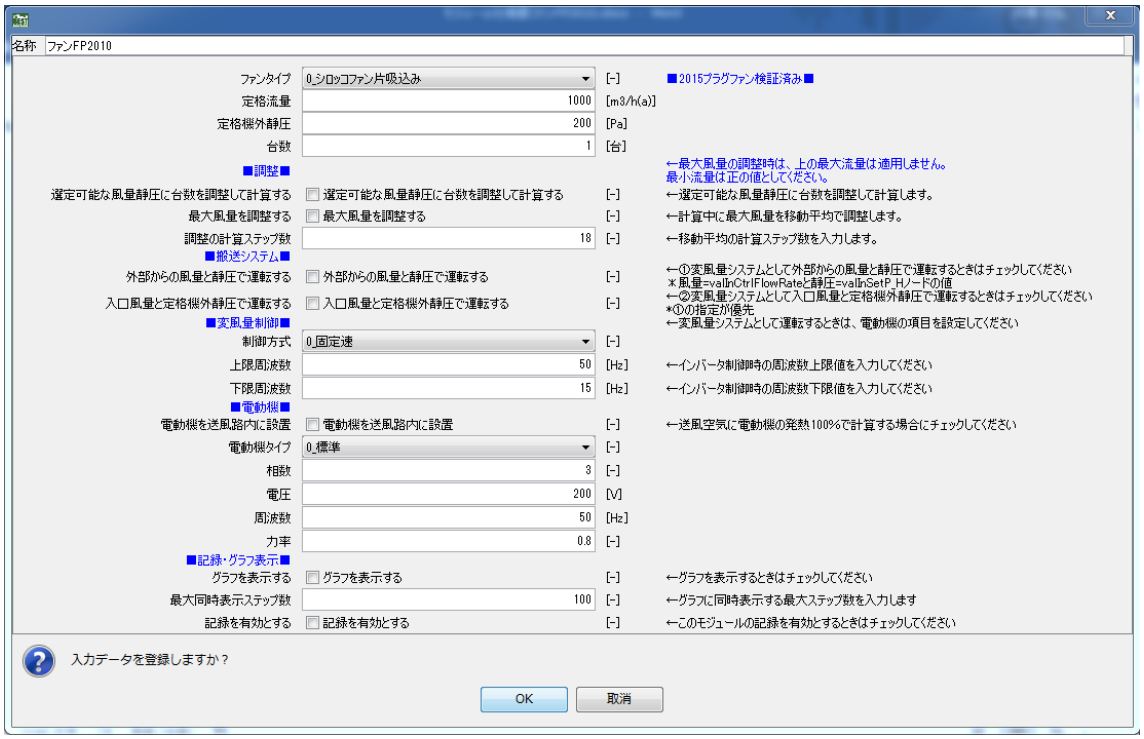
4）建築環境工学 熱環境と空気環境, 朝倉書店, 2009, 宇田川他

「ファン FP2010」（場所：設備 2015／搬送機器 2015）

モジュール名	ファン FP2010
クラス	FanFPModule20101111

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、ファンを再現するモジュールであり、「シロッコファン片吸込み」、「シロッコファン両吸込み」、「リミットロードファン片吸込み」、「リミットロードファン両吸込み」、「プラグファン」の機器特性を使用して、シミュレーションを行うものである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

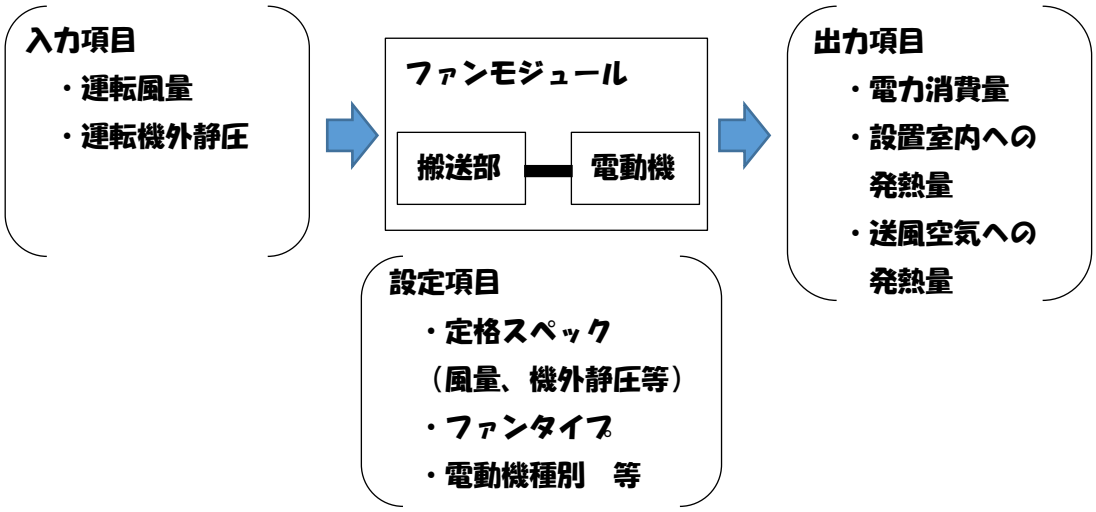


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図1における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー	◎	複数種類設定する場合は変更が必要。
1	ファンタイプ	String 0_シロッコファン片吸込み、 1_シロッコファン両吸込み、 2_リミットロードファン片吸込み、 3_リミットロードファン両吸込み、 4_プラグファン	dFanType	0_シロッコファン片吸込み	[-]	ー	ー	◎	
2	定格流量	double	f_GA_S	1000	[m3/h(a)]	ー	ー	◎	
3	定格機外静圧	double	f_PA_S	200	[Pa]	ー	ー	◎	
4	台数	double	dai	1	[台]	ー	ー	◎	
①	■調整■								
1	選定可能な風量静圧に台数を調整して計算する	boolean	isCalcAdjustFP	FALSE(チェック無し)	[-]	ー	ー	○	選定可能な風量静圧に台数を調整して計算します。
2	最大風量を調整する	boolean	isAdjust	FALSE(チェック無し)	[-]	ー	ー	○	計算中に最大風量を移動平均で調整します。
②	■搬送システム■								
1	外部からの風量と静圧で運転する	boolean	isUseValIn	FALSE(チェック無し)	[-]	ー	ー	○	変風量システムとして外部からの風量と静圧で運転するとき（ファン台数制御モジュール等から接続）はチェックしてください
2	入口風量と定格機外静圧で運転する	boolean	isVAV	FALSE(チェック無し)	[-]	ー	ー	○	変風量システムとして入口風量と定格機外静圧で運転

									するときはチェックしてください
No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
③	■変風量制御■								
1	制御方式	String 0_固定速、1_INV制御	dControlType	0_固定速	[-]		ー	◎	
2	上限周波数	double	dHzMax	50	[Hz]	ー	0	◎	
3	下限周波数	double	dHzMin	15	[Hz]	ー	0	◎	
④	■電動機■								
1	電動機を送風路内に設置	boolean	disMotorHeat	FALSE(チェック無し)	[-]		0	◎	送風空気に電動機の発熱100%で計算する場合にチェックしてください
2	電動機タイプ	String 0_標準、1_高効率、2_IPM	dMotorType	0_標準	[-]		0	◎	
3	相数	int	Phase	3	[-]	3	1	△	今後計算に使用予定
4	電圧	double	Voltage	200	[V]	1000000	0	△	今後計算に使用予定
5	周波数	double	Frequency	50	[Hz]	60	50	△	今後計算に使用予定
6	力率	double	PowerFactor	0.8	[-]		0	△	今後計算に使用予定
⑤	■記録・グラフ表示■								
1	グラフを表示する	boolean	isGVisible	FALSE(チェック無し)	[-]			○	グラフを表示するときはチェック
2	最大同時表示ステップ数	int	maxItemCount	100	[-]			○	グラフに同時表示する最大ステップ数を入力
3	記録を有効とする	boolean	isRecord	FALSE(チェック無し)	[-]			○	このモジュールの記録を有効とするときはチェック

（5）シーケンス接続（図1における入力項目、出力項目）

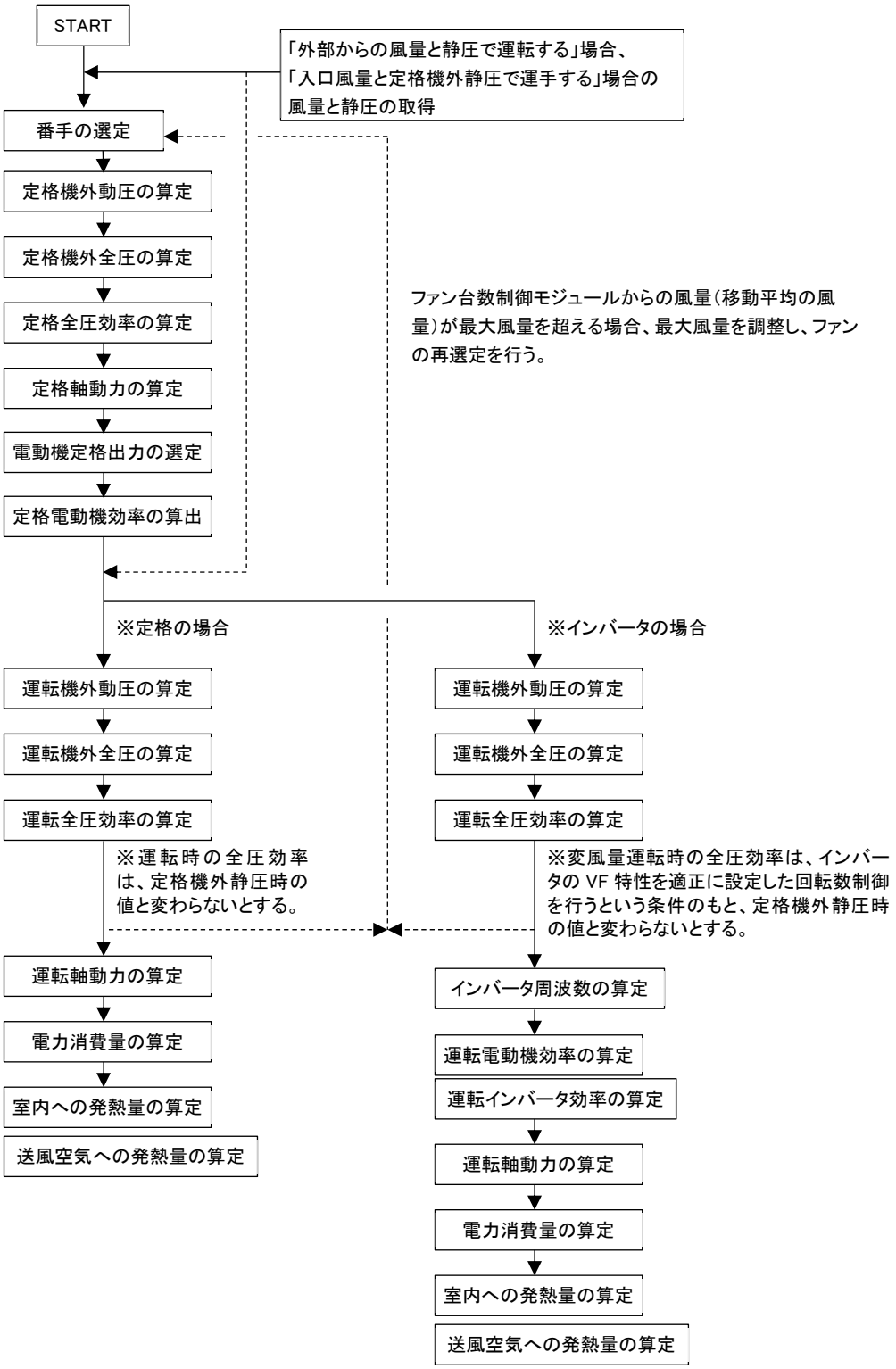
No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	入口空気	L0_airIn	DB_{in}		状態	空気	入口	
2	出口空気	L0_airOut	DB_{out}		状態	空気	出口	
3	電力	L0_eleIn	EP		状態	電気	入口	
4	上位からの質量風量の制御量	L0_valInCtrlFlowRate		g/s	状態	double 値	入口	ファン台数制御モジュールからの信号で容量を制御する場合に用いる。制御モジュールから風量制御信号がある場合のみ有効とする。
5	指定運転全揚程	L0_valInSetP_H		Pa	状態	double 値	入口	ファン台数制御モジュールからの全静圧でを制御する場合に用いる。制御モジュールからの風量制御信号がある場合のみ有効とする。
7	運転状態	L1_swcIn	SWC		制御	On/Off 信号	入口	制御モジュールからの On/Off 信号を受け取る。
8	空調モード	L1_modIn	MODE		制御	制御モード	入口	制御モジュールからの冷房・暖房別の信号を受け取る。
9	記録	L2_recOut	REC		記録	メモリ	出口	空調記録モジュールへ運転状態等を出力する。

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
2	消費電力	電力	W	出口
3	入口空気温度	温度	℃	入口
4	入口空気絶対湿度	絶対湿度	g/g'	入口
5	入口風量	質量流量	g/s	入口
6	出口空気温度	温度	℃	出口
7	出口空気絶対湿度	絶対湿度	g/g'	出口
8	出口風量	質量流量	g/s	出口
9	軸動力	軸動力	W	My
10	周囲空気への発熱量	顕熱	W	My
11	送風空気への発熱量	顕熱	W	My
12	運転質量流量	質量流量	g/s	My
13	運転機外静圧	機外静圧	Pa	My
14	運転流量負荷率	ファン流量負荷率	−	My
15	運転全圧効率	ファン全圧効率	−	My
16	電動機効率	電動機効率	−	My
17	インバータ効率	インバータ効率	−	My
18	調整最大風量	質量流量	g/s	調整
19	調整ステップ平均風量	質量流量	g/s	調整
20	調整 SPEC	−	−	調整

（7）計算フロー・計算内容

モジュール内での計算フローを示す。

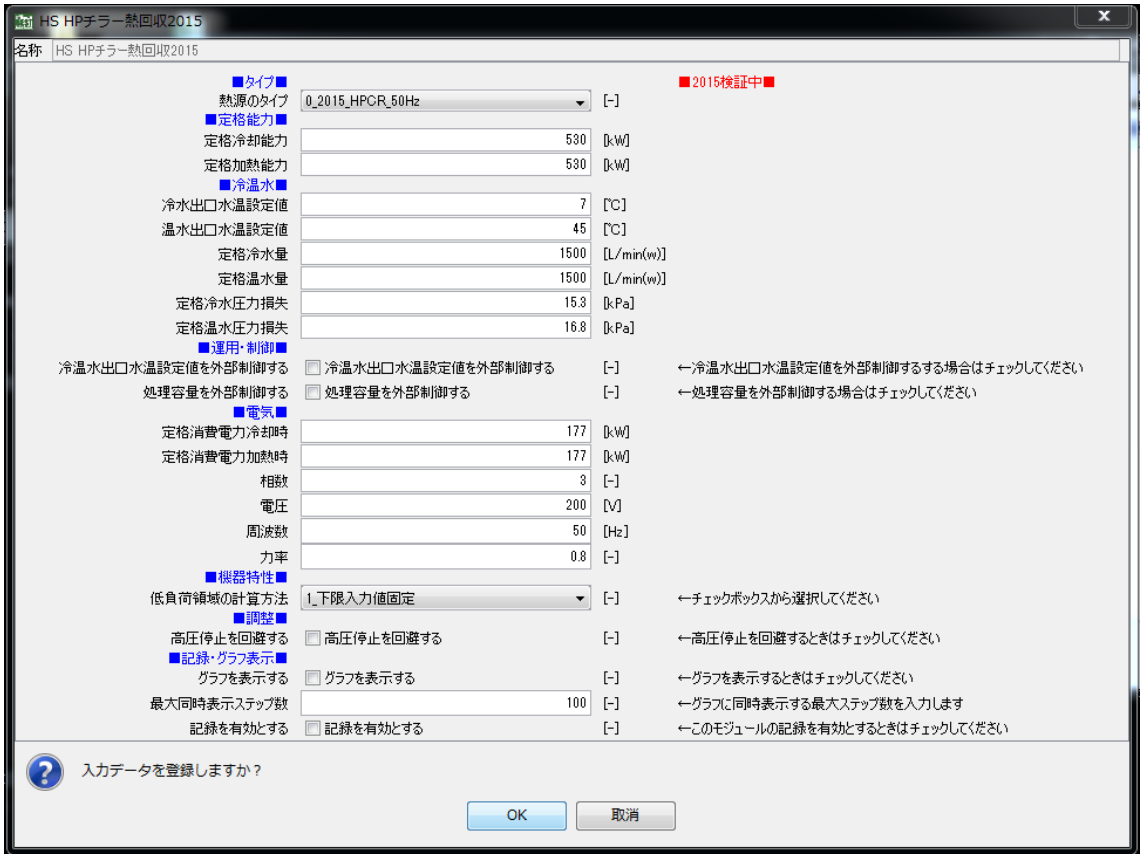


「熱回収式空冷ヒートポンプチラー」（場所：設備 2015／熱源 2015）

モジュール名	熱回収型空冷ヒートポンプチラー
クラス	HeatPumpChillerRecModule201

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、熱回収式空冷ヒートポンプチラーを再現するモジュールであり、本機器の運転モードである「冷却専用運転」、「冷却主体熱回収運転」、「完全熱回収運転」、「加熱主体熱回収運転」、「加熱専用運転」のそれぞれにおいて、マップデータにて整理した機器特性を使用して、シミュレーションを行うものである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

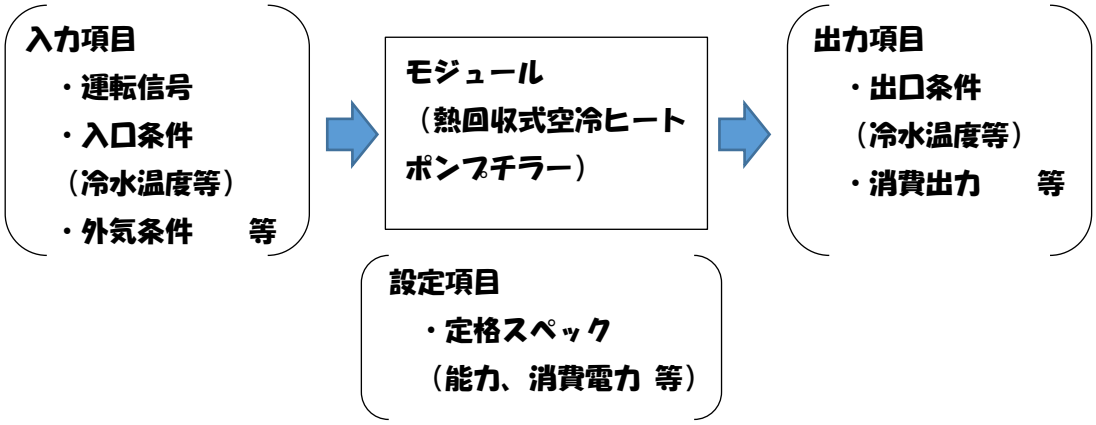


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図1における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー	○	複数台設定する場合は変更が必要。
①	■タイプ■								
1	熱源のタイプ	String (0_2015_HPCR_50Hz 1_2015_HPCR_60Hz)		0_2015_HPCR_50Hz	[-]	ー	ー	◎	
②	■定格能力■								
2	定格冷却能力	double	Q_C_S	530	[kW]	ー	0	◎	
3	定格加熱能力	double	Q_H_S	530	[kW]	ー	0	◎	
③	■冷温水■								
4	冷水出口水温設定値	double	TWout_C_SP	7	[℃]		0	◎	
5	温水出口水温設定値	double	TWout_H_SP	45	[℃]		0	◎	
6	定格冷水量	double	GW_C_S	1500	[L/min(w)]		0	◎	
7	定格温水量	double	GW_H_S	1500	[L/min(w)]		0	◎	
8	定格冷水圧力損失	double	PW_C_S	15.3	[kPa]		0	△	今後計算に使用予定
9	定格温水圧力損失	double	PW_H_S	16.8	[kPa]		0	△	今後計算に使用予定
④	■運用・制御■								
10	冷温水出口水温設定値を外部制御する	boolean	SP_T_watOutCH		[-]			△	別のモジュールで出口温度設定値を変更する場合に変更要
11	処理容量を外部制御する	boolean	CapRate		[-]			△	別のモジュールで処理容量を変更する場合に変更要
⑤	■電気■								
12	定格消費電力冷却時	double	PE_C_S	177	[kW]		0	◎	
13	定格消費電力加熱時	double	PE_H_S	177	[kW]		0	◎	
14	相数	int	Phase	3	[-]		1	△	今後計算に使用予定
15	電圧	double	Voltage	200	[V]		0	△	今後計算に使用予定

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
16	周波数	double	Frequency	50	[Hz]		50	△	今後計算に使用予定
17	力率	double	PowerFactor	0.8	[-]		0	△	今後計算に使用予定
⑥	■機器特性■								
18	低負荷領域の計算方法	String （0_発停運転 1_下限入力値固定 2_下限 COP 値固定 3_下限入力値と中間切片）	CalcTypeLowerRangeLoad	1_下限入力値固定	[-]	1	0	△	論文参照
⑦	■調整■								
19	高圧停止を回避する	boolean	isOPEmanual		[-]			△	今後計算に使用予定
⑧	■記録・グラフ表示■								
1	グラフを表示する	boolean	isGVisible	FALSE(チェック無し)	[-]			○	グラフを表示するときはチェック
2	最大同時表示ステップ数	int	maxItemCount	100	[-]			○	グラフに同時表示する最大ステップ数を入力
3	記録を有効とする	boolean	isRecord	FALSE(チェック無し)	[-]			○	このモジュールの記録を有効とするときはチェック

（5）シーケンス接続（図1における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	チラー入口冷水	LO_watInC	TW_{C_in}		状態	水	入口	
2	チラー出口冷水	LO_watOutC	TW_{C_out}		状態	水	出口	
3	チラー入口温水	LO_watInH	TW_{H_in}		状態	水	入口	
4	チラー出口温水	LO_watOutH	TW_{H_out}		状態	水	出口	
5	入口空気	LO_airIn	DB_{in}		状態	空気	入口	
6	出口空気	LO_airOut	DB_{out}		状態	空気	出口	
7	電力	LO_eleIn	EP		状態	電気	入口	
8	外部からの容量制御	LO_valInCapCtrl	PLR	ー	状態	double 値	入口	熱源制御モジュールからの信号（0～1）で容量を制御する場合に用いる。 熱源制御モジュールから容量制御信号がある場合のみ有効とする。
9	外部からの冷水出口温度の設定値	LO_valInSP_T_watOutC	$TW_{C\_out\_set}$	℃	状態	double 値	入口	熱源制御モジュールからの信号（0～1）で冷水出口温度を制御する場合に用いる。 熱源制御モジュールからの出口温度設定がある場合のみ有効とする。
10	外部からの温水出口温度の設定値	LO_valInSP_T_watOutH	$TW_{H\_out\_set}$	℃	状態	double 値	入口	熱源制御モジュールからの信号（0～1）で温水出口温度を制御する場合に用いる。 熱源制御モジュールからの出口温度設定がある場合のみ有効とする。
11	運転状態	L1_swcIn	SWC		制御	On/Off 信号	入口	熱源制御モジュールからの On/Off 信号を受け取る。
12	空調モード	L1_modIn	MODE		制御	制御モード	入口	熱源制御モジュールからの冷房・暖房別の信号を受け取る。
13	記録	L2_recOut	REC		記録	メモリ	出口	空調記録モジュールへ運転状態等を出力する。

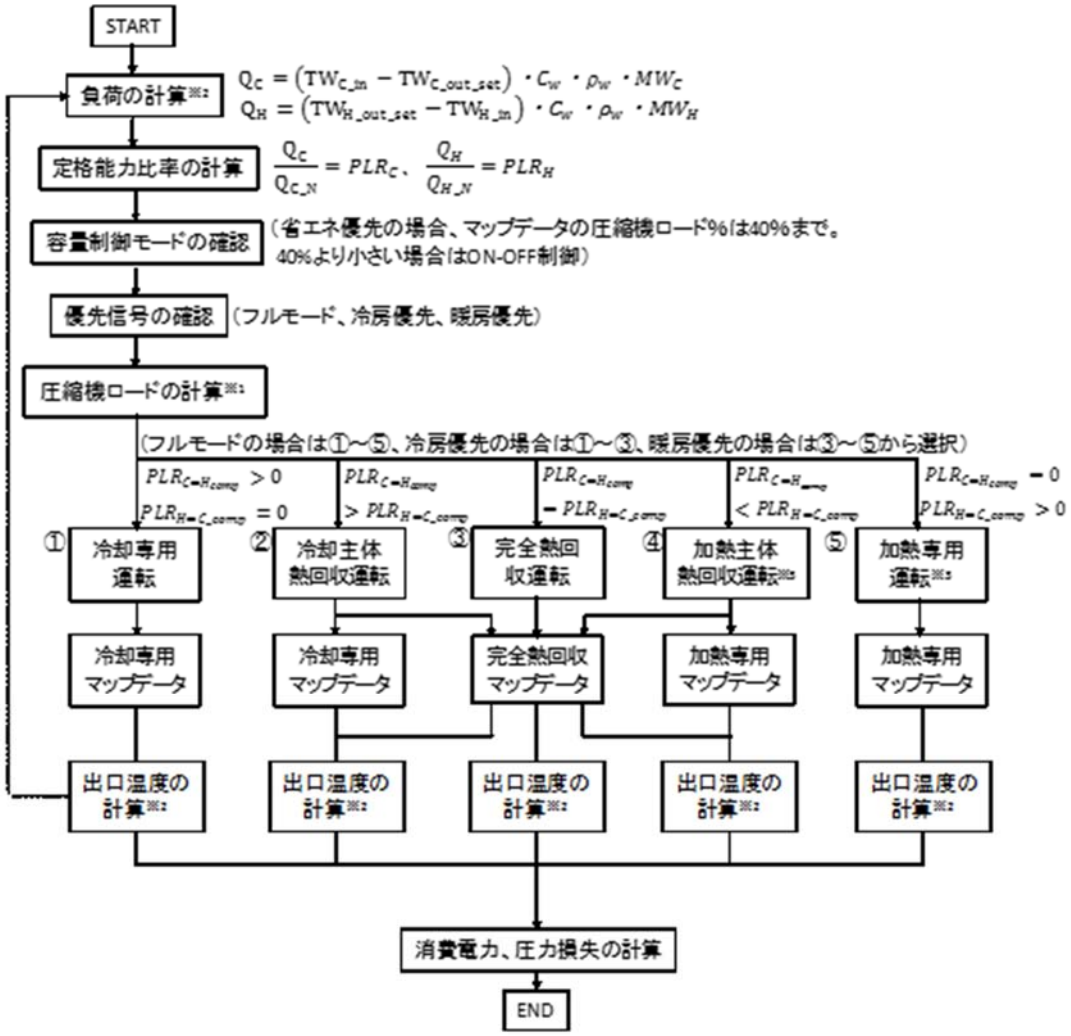
（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
2	入口冷水温度	温度	°C	入口
3	入口冷水流量	質量流量	g/s	入口
4	出口冷水温度	温度	°C	出口
5	出口冷水流量	質量流量	g/s	出口
6	入口温水温度	温度	°C	入口
7	入口温水流量	質量流量	g/s	入口
8	出口温水温度	温度	°C	出口
9	出口温水流量	質量流量	g/s	出口
10	入口空気乾球温度	温度	°C	入口
11	入口空気湿球温度	温度	°C	入口
12	消費電力	電力	W	エネルギー
13	COP	COP	−	My
14	処理能力冷却	処理能力	W	My
15	処理熱量冷却	処理熱量	J	My
16	処理能力加熱	処理能力	W	My
17	処理熱量加熱	処理熱量	J	My
18	累積熱量冷却	処理熱量	J	My
19	累積熱量加熱	処理熱量	J	My
20	負荷率冷却	−	−	My
21	負荷率過熱	−	−	My
22	出口冷水温度設定値	温度	°C	入口
23	出口温水温度設定値	温度	°C	入口
24	外部からの要求処理容量	制御	−	入口
25	冷水側圧力損失	圧力	Pa	My
26	温水側圧力損失	圧力	Pa	My

（7）計算フロー・計算内容

モジュール内での計算フローを示す。

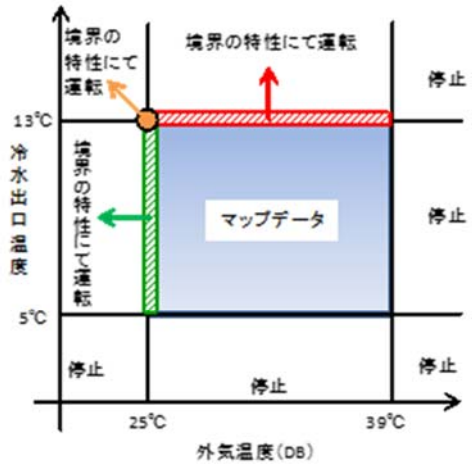
モジュールに入力されたデータ及び設定項目にて設定された定格能力を用いて、負荷や定格能力比率を計算し、運転モードを選択する。運転モードごとに設定されている特性データ（マップ格子点データ）を用いて、処理能力等を算出、出口温度の計算を行い、出口温度が設定温度に満たない場合は、その温度を出口温度として再度負荷や定格能力比率を再計算する。



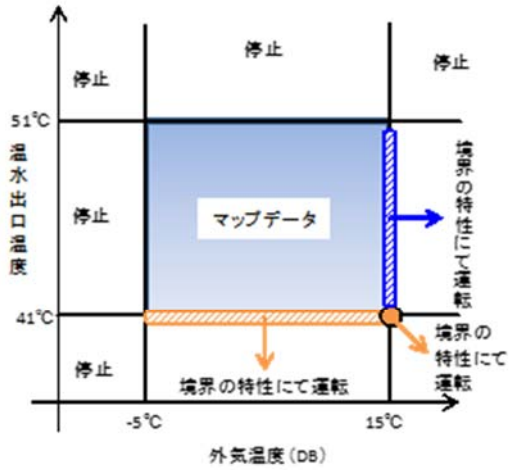
（8）データ範囲と範囲外の取扱い

入力項目・出力項目の範囲と範囲外の取扱いを示す。

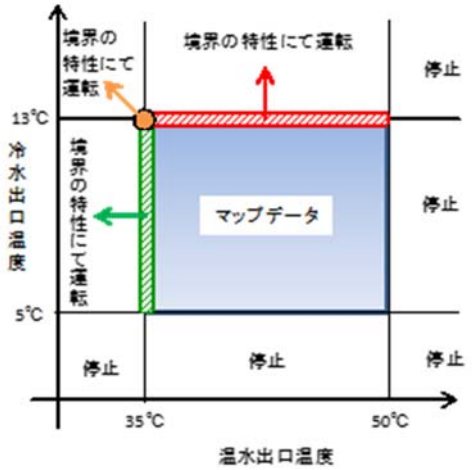
冷却専用運転



加熱専用運転



完全熱回収運転

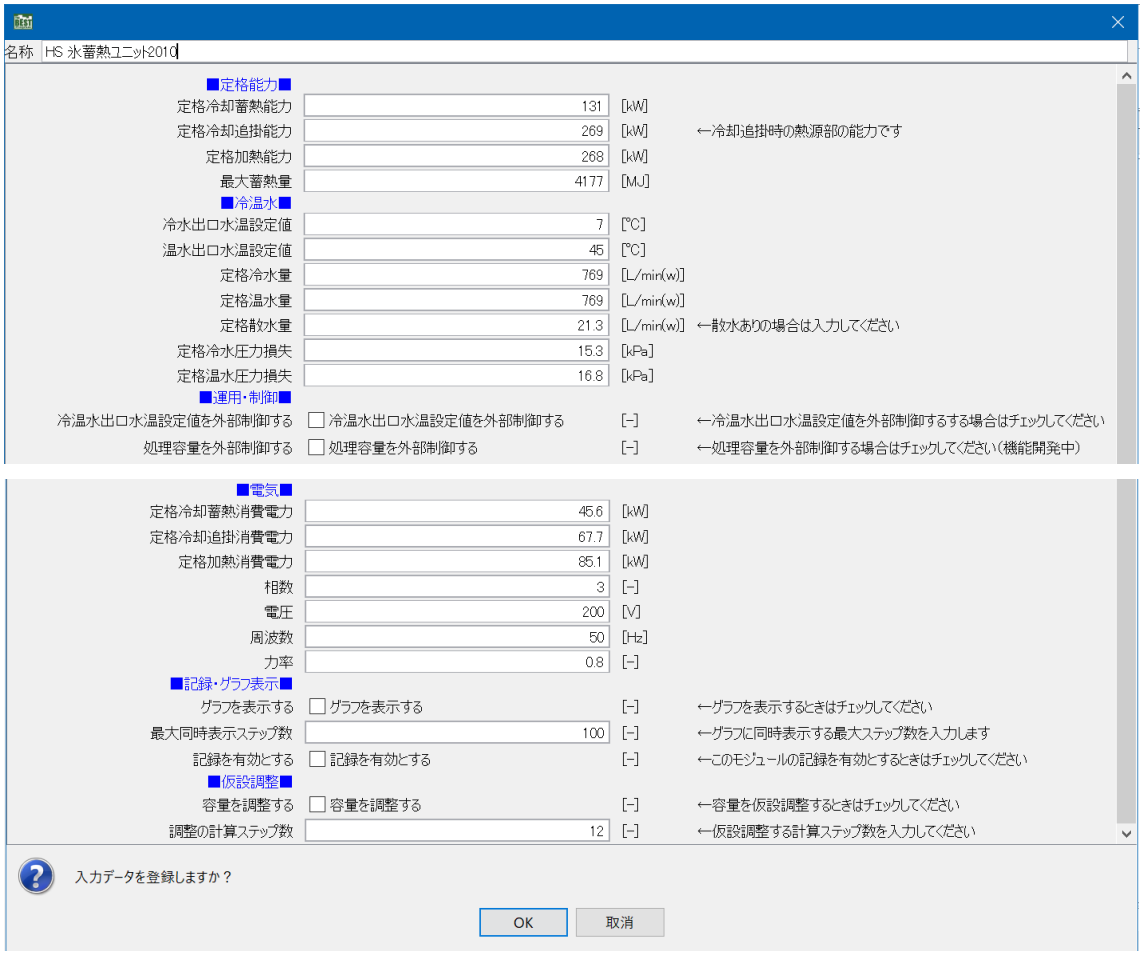


「HS 氷蓄熱ユニット 2010」（場所：設備 2015／熱源 2015／）

モジュール名	HS 氷蓄熱ユニット 2010
クラス	HeatPumpChillerIceSTunitModule20100808

（1）入力画面

・スペック



（2）モジュールの概要

熱源と氷蓄熱槽が一体となったモデルで、他の熱源の入力項目と違う点は、冷却能力は蓄熱時と追掛け時の 2 個の能力、氷蓄熱槽の最大蓄熱量の入力が必要である。

運転は冷房（追掛け運転、ピークシフト、ピークカット、夜間蓄熱)、暖房運転に分類し処理を行っている。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

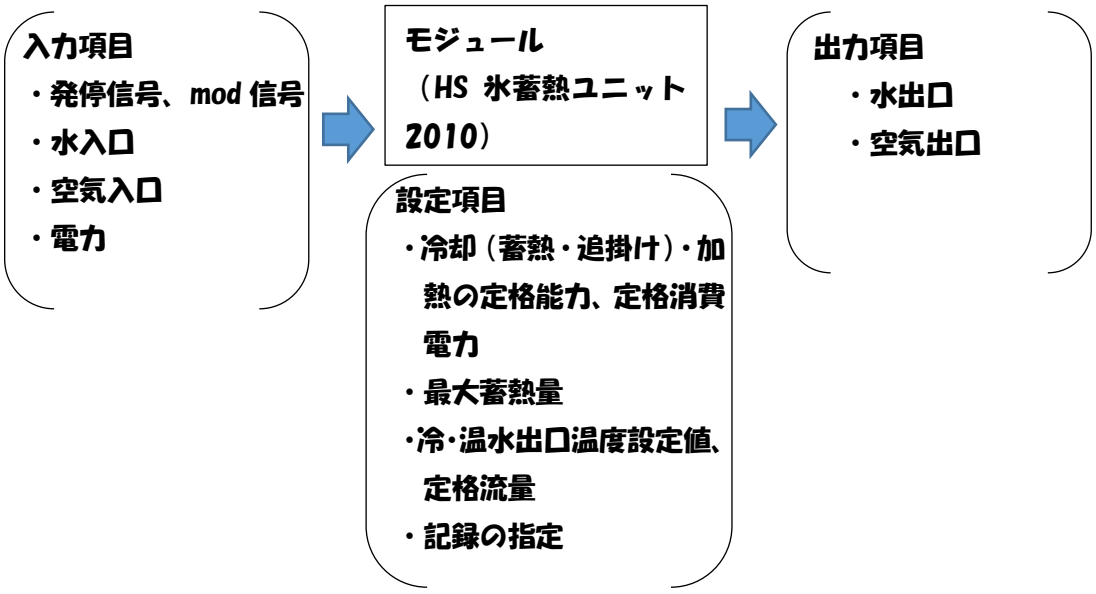


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	■定格能力■					―	―		
2	定格冷却蓄熱能力	double	AHPC_Q_CI	131	[kW]	―	0		
3	定格冷却追掛能力	double	AHPC_Q_C	269	[kW]	―	0		←冷却追掛時の熱源部の能力です
4	定格加熱能力	double	AHPC_Q_H	268	[kW]	―	0		
5	最大蓄熱量	double	AHPC_MaxQQST	4177	[MJ]	―	0		
6	■冷温水■					―	―		
7	冷水出口水温設定値	double	AHPC_TWout_C	7	[℃]	―	0		
8	温水出口水温設定値	double	AHPC_TWout_H	45	[℃]	―	0		
9	定格冷水量	double	AHPC_GW_C	769	[L/min(w)]	―	0		
10	定格温水量	double	AHPC_GW_H	793	[L/min(w)]	―	0		
11	定格散水量	double	AHPC_GW_EV	21.3	[L/min(w)]	―	0		←散水ありの場合は入力してください
12	定格冷水圧力損失	double	AHPC_PW_C	15.3	[kPa]	―	0		
13	定格温水圧力損失	double	AHPC_PW_H	16.8	[kPa]	―	0		
14	■運用・制御■					―	―		
15	冷温水出口水温設定値を外部制御する	boolean	isChangeSP_T_watOutCH	FALSE	[-]	―	―		←冷温水出口水温設定値を外部制御するする場合はチェックしてください チェックがある場合はLO_valInSP_T_watOutCHの値で制御する。
16	処理容量を外部制御する	boolean	isChangeCapRate	FALSE	[-]	―	―		←処理容量を外部制御する場合はチェックしてください（機能開発中） チェックがある場合は LO_valInCapCtrl のあたいで制御する。

17	■電気■					―	―		
18	定格冷却蓄熱消費電力	double	AHPC_PE_CI	45.6	[kW]	―	0		
19	定格冷却追掛消費電力	double	AHPC_PE_C	67.7	[kW]	―	0		
20	定格加熱消費電力	double	AHPC_PE_H	85.1	[kW]	―	0		
21	相数	int	phase	3	[-]	3	1		
22	電圧	double	voltage	200	[V]	―	0		
23	周波数	double	frequency	50	[Hz]	60	50		
24	力率	double	powerFactor	0.8	[-]	1	0		
25	■記録・グラフ表示■					―	―		
26	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
27	最大同時表示ステップ数	int	maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
28	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください
29	■仮設調整■					―	―		
30	容量を調整する	boolean	isAdjust2012	FALSE	[-]	―	―		←容量を仮設調整するときはチェックしてください
31	調整の計算ステップ数	int	numAdjustSteps	12	[-]	―	6		←仮設調整する計算ステップ数を入力してください

（5）シーケンス接続（図１における入力項目、出力項目）

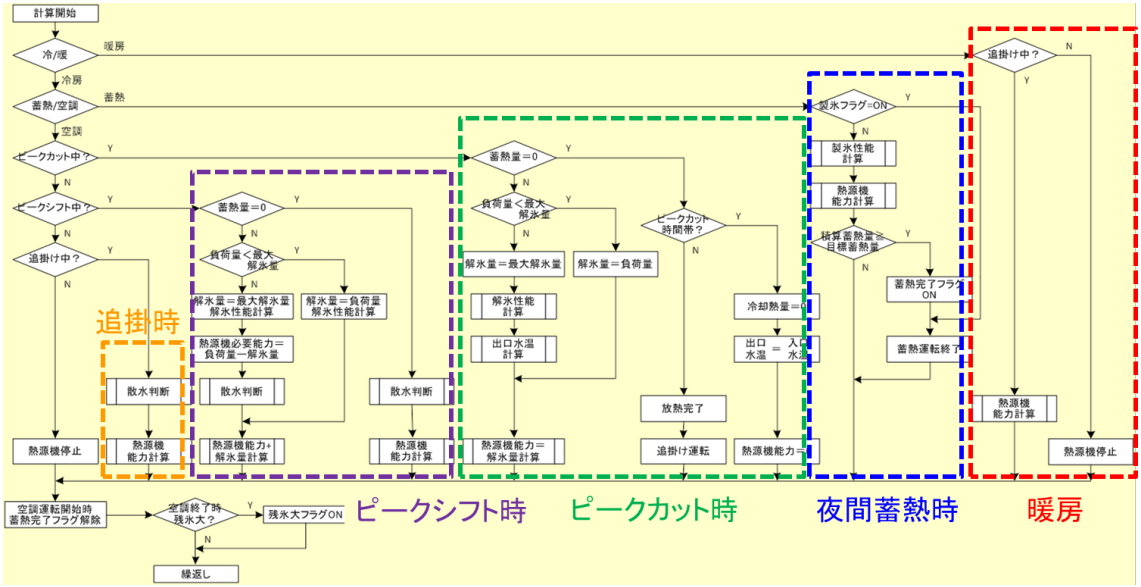
No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	On/Off 信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	運転モード	L1_modIn	modIn	-	制御	制御モード	入口	
4	冷温水入口	L0_watInCH	watInCH	-	状態	水	入口	
5	冷温水出口	L0_watOutCH	watOutCH	-	状態	水	出口	
6	外気入口	L0_airIn	airIn	-	状態	空気	入口	
7	外気出口	L0_airOut	airOut	-	状態	空気	出口	
8	水（散水用）入口	L0_watInCW	watInCW	-	状態	水	入口	散水に使用する水
9	電力入口	L0_eleIn	eleIn	-	状態	電気	入口	
10	蓄熱率出口	L0_valOutSTRate	valOutSTRate	-	値	値	出口	蓄熱率＝計算時刻の蓄熱量／最大蓄熱量
11	運転容量設定値入口	L0_valInCapCtrl	valInCapCtrl	-	値	値	入口	
12	出口水温設定値入口	L0_valInSP_T_watOutCH	valInSP_T_watOutCH	-	値	値	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	AHPCiSTu_Message#-#-	メッセージ		メッセージ
2	AHPCiSTu_入口冷温水温度#℃#温度	入口冷温水温度	℃	入口
3	AHPCiSTu_入口冷温水流量#g/s#質量流量	入口冷温水流量	g/s	入口
4	AHPCiSTu_出口冷温水温度#℃#温度	出口冷温水温度	℃	出口
5	AHPCiSTu_出口冷温水流量#g/s#質量流量	出口冷温水流量	g/s	出口
6	AHPCiSTu_入口 CW 流量#g/s#質量流量	入口 CW 流量	g/s	入口
7	AHPCiSTu_消費電力#W#電力 熱源 空調熱源本体	消費電力	W	エネルギー
8	AHPCiSTu_COP#-#COP	COP	—	My
9	AHPCiSTu_処理能力#W#処理能力	処理能力	W	My
10	AHPCiSTu_処理能力熱源分#W#処理能力熱源分	処理能力（熱源追掛け）	W	My
11	AHPCiSTu_処理能力蓄熱分#W#処理能力蓄熱分	処理能力（放熱）	W	My
12	AHPCiSTu_放熱#W#放熱	放熱（出口空気への放熱）今は＝０	W	My
13	AHPCiSTu_蓄熱量#J#蓄熱量	蓄熱量	J	My
14	AHPCiSTu_出口冷水温度設定値#℃#温度	外部からの出口冷温水設定値	℃	入口
15	AHPCiSTu_外部からの要求処理容量#-#制御	外部からの要求処理容量　今は未使用	—	入口
16	AHPCiSTu_調整冷却能力#W#熱量	自動調整の冷却能力	W	調整
17	AHPCiSTu_調整加熱能力#W#熱量	自動調整の加熱能力	W	調整
18	AHPCiSTu_圧力損失#Pa#圧力	圧力損失	Pa	My

（7）計算フロー・計算内容

・計算フローを下図に示す。



（8）データ範囲と範囲外の取扱い

冷却運転時

外気 DB<13℃の場合 13℃として計算。

外気 DB＞43℃の場合 43℃として計算。

暖房運転時

外気 DB>-5℃の場合、温水出口温度設定値の上限は 55℃

外気 DB＝-15℃の場合、温水出口温度設定値の上限は 45℃

-15℃から-5℃の間の温水出口設定値の上限は比例補間

名称　　：冷却塔 2016（場所：設備 2015／熱源 2015）

モジュール名　：CoolingTowerwithValveModule2016

(1) 入力画面

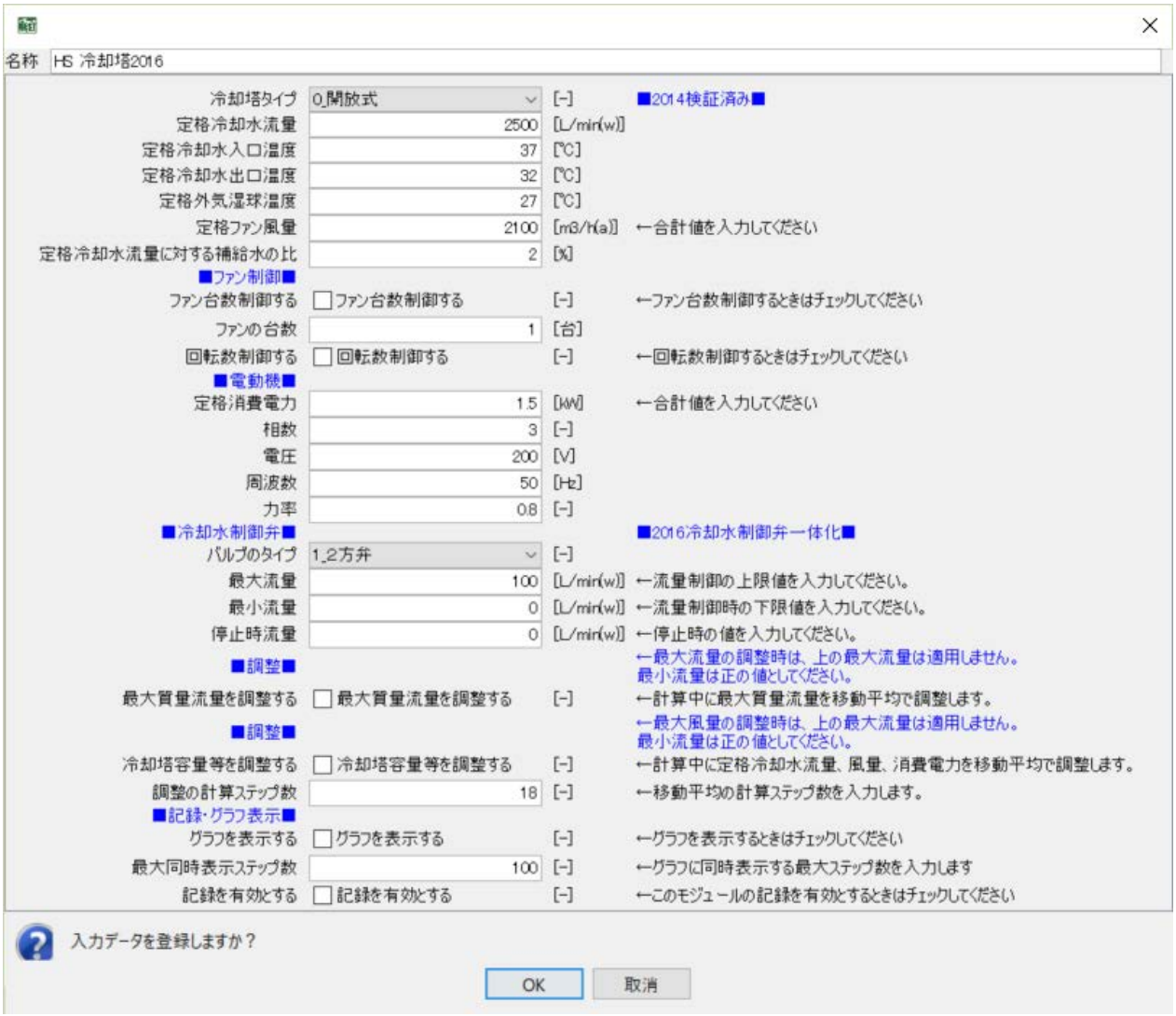


図 1　冷却塔モジュールの入力画面

(2) **モジュールの概要**

本モジュールは、冷却塔を再現するモジュールであり、本モジュールで「開放式」、「密閉式」、のそれぞれにおいて、物理モデルによってにて整理した機器特性を使用して、シミュレーションを行うものである。なお、冷却水温度安定用のバイパス用 2 方弁および 3 方弁、冷却塔ファンの発停および変風量制御についても内包したモジュールである。ただし、3 方弁およびﾌｧﾝ発停用の PID モジュールからの制御信号が必要である。

(3) **モジュールの入出力**

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

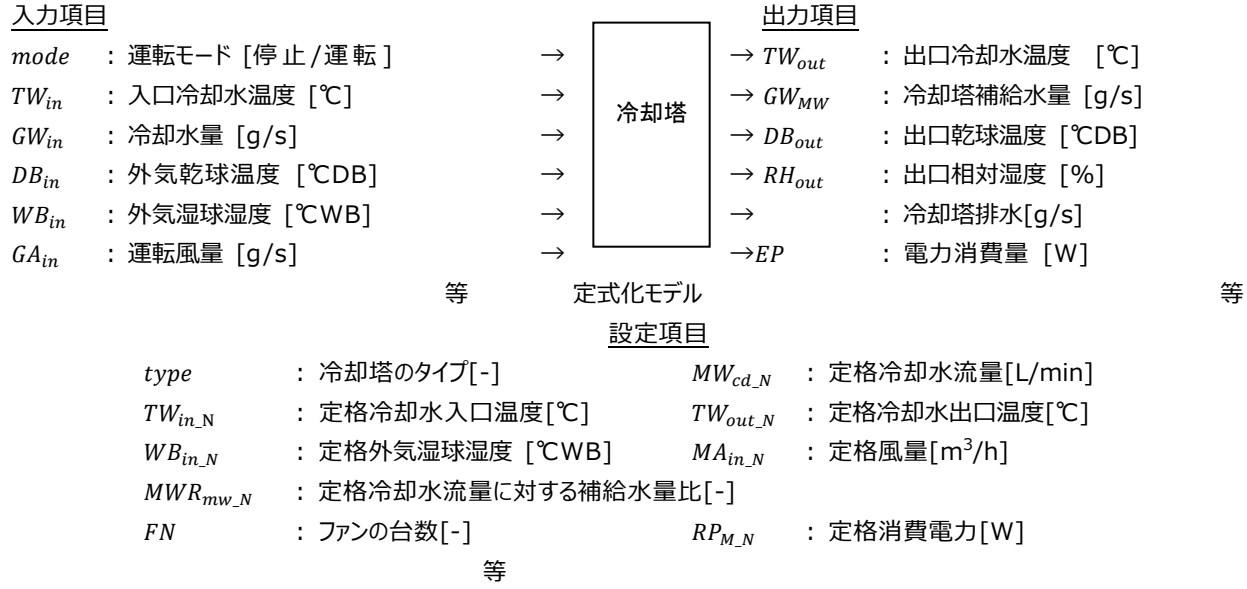


図 2 冷却塔モジュールの計算概要

(4) スペック入力項目（図 2 における設定項目）

表 1 空調機コイルモジュールのスペック入力項目

NO.	項目	型・クラス名	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
1	冷却塔タイプ	String(プルダウン) 0_開放式、1_密閉式		0_開放式	[-]			◎	
2	定格冷却水流量	double	MW_{cd_N}	2500	[L/min(w)]	1000000	0	◎	
3	定格冷却水入口温度	String	TW_{in_N}	37	[℃]		0	◎	
4	定格冷却水出口温度	String	TW_{out_N}	32	[℃]		0	◎	
5	定格外気湿球温度	String	WB_{in_N}	27	[℃]		0	◎	
6	定格ファン風量	String	MA_{in_N}	2100	[m3/h(a)]		0		
7	定格冷却水流量に対する補給水の比	double	MWR_{mw_N}	2	[%]	1000000	0	○	
8	■ファン制御■								
9	ファン台数制御する	boolean		FALSE	[-]			◎	ファン台数制御するときはチェックしてください
10	ファンの台数	String	FN	1	[台]		0	◎	
11	回転数制御する	boolean		FALSE	[-]			◎	回転数制御するときはチェックしてください
12	■電動機■								
13	定格消費電力	double	RP_{M_N}	1.5	[kW]	1000000	0	◎	合計値を入力してください
14	相数	int		3	[-]	3	1	△	
15	電圧	double		200	[V]		0	△	
16	周波数	double		50	[Hz]	60	50	△	
17	力率	double		0.8	[-]	1	0	△	
18	設置空間への熱損失を計算する	boolean		TRUE	[-]			△	計算する場合にチェックする。熱損失を計算する場合は、以下の項目への入力が必要です。
19	熱通過率[W/(m2K)]	double		0	[W/(m2K)]	1000000	0	△	
20	外表面積[m2]	double		0	[m2]	1000000	0	△	
21	保有水量[g]	double		0	[g]	1E+09	0	△	
22	■冷却水制御弁■								
23	バルブのタイプ	String(プルダウン) 0_なし、1_２方弁、2_３方弁		2_３方弁	[-]			◎	
24	最大流量	double		100	[L/min(w)]		0	◎	流量制御の上限値を入力してください。
25	最小流量	double		0	[L/min(w)]		0	○	流量制御時の下限値を入力してください。
26	停止時流量	double		0	[L/min(w)]		0	○	停止時の値を入力してください。

表 2 空調機コイルモジュールのスペック入力項目（続き）

NO.	項目	型・クラス名	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
27	■調整■								
28	最大質量流量を調整する	boolean		FALSE	[-]			△	計算中に最大質量流量を移動平均で調整します。
29	冷却塔容量等を調整する	boolean		FALSE	[-]			△	計算中に定格冷却水流量、風量、消費電力を移動平均で調整します。
30	調整の計算ステップ数	int		18	[-]			△	移動平均の計算ステップ数を入力します。
31	■記録・グラフ表示■								
32	グラフを表示する	boolean		FALSE	[-]			○	グラフを表示するときはチェックしてください
33	最大同時表示ステップ数	int		100	[-]			○	グラフに同時表示する最大ステップ数を入力します
34	記録を有効とする	boolean		FALSE	[-]			○	このモジュールの記録を有効とするときはチェックしてください

(5) シーケンス接続（図 2 における入力項目、出力項目）

表 3 冷却塔モジュールのシーケンス接続項目

No.	項目	接続端子名	記号	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut		記録	メモリ	出口	空調記録モジュールへ運転状態等を出力する。
2	上位モジュールからの運転状態	L1_swcIn		制御	On/Off 信号	入口	熱源制御モジュールから On/Off 信号を受け取る。
3	上位モジュールからの空調制御	L1_modIn		制御	制御モード	入口	熱源制御モジュールから冷房/暖房モード信号を受け取る。
4	冷却塔入口冷却水	L0_watInCD		状態	水	入口	テンプレートは接続準が逆では？ ・現状：熱源→冷却塔→ポンプ→熱源 ・改定案：熱源→ポンプ→冷却塔→熱源
5	冷却塔出口冷却水	L0_watOutCD		状態	水	出口	
6	冷却塔出口冷却水温度	L0_watObsCTout		状態	水	観測	冷却塔ファン PID のための観測用冷却水出口温度
7	冷却塔入口空気	L0_airIn		状態	空気	入口	
8	冷却塔出口空気	L0_airOut		状態	空気	出口	
9	冷却塔入口補給水	L0_watInCW		状態	水	入口	
10	冷却塔出口排水	L0_watOutD		状態	水	出口	
11	電力	L0_eleIn		状態	電気	入口	
12	？	L0_airOutInZone		状態	空気	出入口	
13	バルブ操作値入力	L0_valInValveCtrl		状態	double 値	入口	バイパス制御弁用 PID モジュールからの操作指示値
14	ファン操作値入力	L0_valInFanCtrl		状態	double 値	入口	ファン制御用 PID モジュールからの操作指示値
15	運転時の質量流量	L0_valOutCtrlFlowRate		状態	double 値	出口	

(6) **記録項目**

表 4 冷却塔モジュールの記録項目

No.	項目	出力内容	単位	分類
1	ﾒｯｾｰｼﾞ	ﾒｯｾｰｼﾞ	-	ﾒｯｾｰｼﾞ
2	冷却水入口水温	温度	℃	入口
3	冷却水入口流量	質量流量	g/s	入口
4	冷却水出口水温	温度	℃	出口
5	冷却水出口流量	質量流量	g/s	出口
6	空気入口乾球温度	温度	℃	入口
7	空気入口湿球温度	温度	℃	入口
8	空気入口絶対湿度	湿度	g/g'	入口
9	空気入口流量	質量流量	g/s	入口
10	空気入口ｴﾝﾀﾙﾋﾟｰ	ｴﾝﾀﾙﾋﾟｰ	J/g'	入口
11	空気出口乾球温度	温度	℃	出口
12	空気出口湿球温度	温度	℃	出口
13	空気出口絶対湿度	湿度	g/g'	出口
14	空気出口流量	質量流量	g/s	出口
15	空気出口ｴﾝﾀﾙﾋﾟｰ	ｴﾝﾀﾙﾋﾟｰ	J/g'	出口
16	冷却水冷却熱量	熱量	W	MY
17	ﾌﾞﾛｰ水放熱量	熱量	W	MY
18	空気放熱量	熱量	W	MY
19	消費電力	電力	W	入口
20	補給水流量	質量流量	g/s	入口
21	排水流量	質量流量	g/s	出口
22	制御弁操作量	操作量	-	入口
23	ﾌｧﾝ操作量	操作量	-	入口
24	ntu 塔特性	-	-	MY
25	rn 水空気比	-	-	MY
26	調整冷却水流量	質量流量	g/s	MY
27	調整ｽﾃｯﾌﾟ平均冷却水流量	質量流量	g/s	MY

(7) **計算方法**

〇物理モデルの理論（基礎式）

$C_l = (L/A)(\alpha t_l/\alpha z) = -(G/A')(\alpha h/\alpha x) = -Ka(h_l - h)$ ・・・（1）

ここで、

C_l ： 水の比熱（＝2.4[kJ/(kg･K)]

L ： 冷却水量 ［kg/h］

G ： ファン風量 [kg/h]

A ： 冷却水の流れに直角方向の充てん材断面積（図 1 参照）［㎡］

A' ： 空気の流れに直角方向の充てん材断面積（図 1 参照）［㎡］

t_l ： 冷却水温度 [℃]

h ： 空気の比エンタルピー [kJ/kg]

h_l ： 冷却水温度 t_l と同じ温度の飽和空気の比エンタルピー [kJ/kg]

x ： 空気の流れ方向の座標軸（図 3 参照）

z ： 冷却水の流れ方向の座標軸（3 参照）

Ka ： エンタルピー基準総括容積電熱係数 [kJ/(m3･h･△h)]

また、

$N = L/G$ （水空気比） ・・・(2)

$U/N = KaZd/(L/A) = KaV/L$ （塔特性、N/U） ・・・(3)

ここで、

V ： 体積*[m3]*

(1)式を、$\xi = x/Xd$、$\zeta = z/Zd$ として書き直すと、

$C_l\, \alpha t_l/\alpha\xi = -(1/N)/(\alpha h/\alpha\zeta) = -U/N\,(h_l - h)$ ・・・(4)

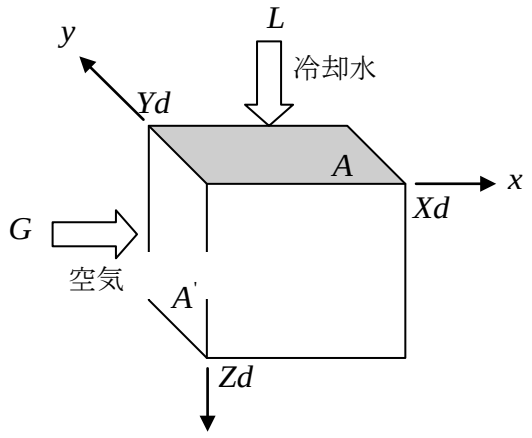


図 直行流形と熱交換部の座標と流れ方向

(4)式を差分法で解くことにより、出口水温及び出口空気エンタルピーを求める。

なお、ファン台数制御時、冷却水循環中にファンが停止した場合は実験値との比較により、開放式は定格風量の 5%の外気が、密閉式は定格風量の 3%の外気が冷却塔に流入することとする。

また、変流量・変風量時の特性は、変流量・変風量時の塔特性は、塔内の水の空気の流動状態が複雑で理論的には求めにくいため、実験結果から式 1.6.3-5 が成り立つものとする。

$Ka = c_1(L/A)^{\alpha}(G/A)^{\beta}$ ・・・(5)

c_1、α、β：定数

式(5)を式(3)に代入すると

$U/N = c_1 Z(L/A)^{\alpha-1}(G/A)^{\beta}$ ・・・(6)

実験結果から求めた各定数は以下の通り。

・$c_1 = 0.05$、$\alpha = 0.2$、$\beta = 0.8$ （開放式）

・$c_1 = 0.03$、$\alpha = 0.1$、$\beta = 0.6$ （密閉式）

○差分法（中心差分法）

冷却塔タイプ

開放式　　FanStopAirInRate(ファン停止時の特性計算上の最低風量比) ＝ 0.05, α ＝ 0.2 , β ＝ 0.2

密閉式　　FanStopAirInRate ＝ 0.03, α ＝ 0.1 , β ＝ 0.6

定格冷却水水量・入口温度・出口温度、外気湿球温度、ファン風量、消費電力、補給水量比、

水空気比の算出

$rn\_def = GW_{in\_N}/GA_{in\_N}$

rn_def;//水空気比[-] N=L/G　waer to air ratio

// N：水空気比　=L/G　=水流量[kg/h]／空気流量[kg(DA)/h]

GW_{in_N} = "定格冷却水流量[g/s]"

GA_{in_N}S = "定格ファン風量[g/s]"

定格時 NTU の算出

this.ntu = this.getNTU(this.def_T_watInCD, this.def_T_watOutCD, this.def_WB_airInOA, this.rn_def);

private double getNTU(double T_watInCD, double T_watOutCD, double WB_airInOA, double N){

$NTU_{CO} = c_l * 1000.* dt /4.* (1/(h_{s1} - h_1) + 1/(h_{s2} - h_2) + 1/(h_{s3} - h_3) + 1/(h_{s4} - h_4))$

*ntu;//*移動単位数（向流型）

// NTU　Number of Transfer Units　＝U/N

// U：交換係数　＝KaV/G

// N：水空気比　=L/G　=水流量[kg/h]／空気流量[kg(DA)/h]

c_l *= 4.18605;*　//水の定圧比熱[J/(g K)]

// V：熱交換部分の容積[m3]

// a：単位容積当たりの接触面積[m2]

// Ka：エンタルピー基準総容積熱伝達係数　あるいは　エンタルピー基準総括体積熱伝達率[kJ/m3hΔh]

TW_{in_N} = *"*定格冷却水入口温度*[℃]"*

TW_{out_N} = *"*定格冷却水出口温度*[℃]"*

$dt = TW_{in\_N} - TW_{out\_N}$

$t_1 = TW_{out\_N} + 0.1 * dt$

$t_2 = TW_{out\_N} + 0.4 * dt$

$t_3 = TW_{out\_N} - 0.4 * dt$

$t_4 = TW_{out\_N} - 0.1 * dt$

$h_{in} = Psychrometrics.FNH(WB_{in}, Psychrometrics.FNxtr(WB_{in}, 100))$

FNxtr(double T, double RH) //　温度、相対湿度から絶対湿度を計算する

$h_1 = h_{in} + c_l * 1000.* N * (t_1 - TW_{out\_N})$

$h_2 = h_{in} + c_l * 1000.* N * (t_2 - TW_{out\_N})$

$h_3 = h_{in} + c_l * 1000.* N * (t_3 - TW_{out\_N})$

$h_4 = h_{in} + c_l * 1000.* N * (t_4 - TW_{out\_N})$

$h_{s1} = Psychrometrics.FNH(t_1, Psychrometrics.FNxtr(t_1, 100))$

$h_{s2} = Psychrometrics.FNH(t_2, Psychrometrics.FNxtr(t_2, 100))$

$h_{s3} = Psychrometrics.FNH(t_3, Psychrometrics.FNxtr(t_3, 100))$

$h_{s4} = Psychrometrics.FNH(t_4, Psychrometrics.FNxtr(t_4, 100))$

FNH(double T, double X) // 乾球温度、絶対湿度からエンタルピーを計算する

定格時 NTU の補正（向流型→直交流型）

$h_{out} = h_{in} + c_l * 1000.* N * dt$

$F = 1. - 0.106 * (1.0 - s)^{3.5}$

$$s = \frac{Psychrometrics.FNH\{TW_{out\_N}, Psychrometrics.FNxtr(TW_{out\_N}, 100)\} - h_{out}}{Psychrometrics.FNH\{TW_{in\_N}, Psychrometrics.FNxtr(TW_{in\_N}, 100)\} - h_{in}}$$

$NTU_{CR} = NTU_{CO}/F$

$rn = GW_{in}/GA_{in}$

//変流量、変風量補正 20121018

$NTU_{VAWV} = (GW_{in}/GW_{in\_N})^{\alpha-1} * (GA_{in}/GA_{in\_N})^{\beta}$

$xNTU = NTU_{VAWV} * NTU_{CR}$

this.calc_airOutANDwatOut(watInCD_, watOutCD_, this.airIn, this.airOut, this.rn, this.ntu * c_ntu);

private void calc_airOutANDwatOut(BestWater watInCD, BestWater watOutCD, BestAir airInOA, BestAir airOutEA, double xN rn, double xNTU $NTU_c * NTU_{CR}$){

$U = xNTU * xN$

int m = 6;

int n = 6;

---step 1---

$t_l(i,1) = TW_{in}$

$h_l(i,1) = Psychrometrics.FNH(t_l(i,1), Psychrometrics.FNxtr(t_l(i,1), 100))$

$h_a(i,1) = TA_{in}$

i=l,2, . ,m+l について，上式(III-34)から$h(i,1)$ を計算する。

$h(i,1) = h_l(i,1) * [exp\{U * (i-1)\} - 1]/exp\{U * (i-1)/m\}$

--- step 2 --- III-35

j= 1, 2, . n について，式(III-35)から$t_l(1,\mathrm{j}+1)$，次いで$h_l(1,\mathrm{j}+1)$を計算する。

$t_l(1,j+1) = t_l(1,\mathrm{j}) + (xk_0 + 2.0 * xk_1 + 2.0 * xk_2 + xk_3)/6$

$h_l(1,j+1) = Psychrometrics.FNH\big(t_l(1,j+1) + xk_2/2\,, Psychrometrics.FNxtr(t_l(1,j+1),100)\big)$

ここで

$R3 = xNTU/(c_l * n)$

$xk_0 = \mathrm{R}_3 * (TW_{in}(1,\mathrm{j}) - HA_{in})$

$hk_0 = Psychrometrics.FNH\big(TW_{in}(1,\mathrm{j}) + xk_0/2\,, Psychrometrics.FNxtr(TW_{in}(1,\mathrm{j}) + xk_0/2\,,100)\big)$

$xk_1 = \mathrm{R}_3 * (hk_0 - HA_{in})$

$hk_1 = Psychrometrics.FNH\big(TW_{in}(1,\mathrm{j}) + xk_1/2\,, Psychrometrics.FNxtr(TW_{in}(1,\mathrm{j}) + xk_1/2\,,100)\big)$

$xk_2 = \mathrm{R}_3 * (hk_1 - HA_{in})$

T = tl[1][j] + xk2 / 2.0

hk2 = Psychrometrics.FNH(T, Psychrometrics.FNxtr(T, 100)) / 1000

$hk_2 = Psychrometrics.FNH\big(TW_{in}(1,\mathrm{j}) + xk_2/2\,, Psychrometrics.FNxtr(TW_{in}(1,\mathrm{j}) + xk_2/2\,,100)\big)$

$xk_3 = R_3 * (hk_2 - HA_{in})$

--- step 3 --- III-33, III-31

i=l,2, …m と j=1, 2, …n について，式(III･33)から$\mathrm{t}_l(i+1,j+1)$，次いで$h_l(i+1,j+1)$，さらに式(III･31)から$h_a(i+1,j+1)$を順番に計算する。

$R_1 = U/\{(2m+U) * R_2\}$

$R_2 = (c_l * xN * n)/m$

$\mathrm{t}_l(i+1,j+1)$

$= t_l(i,j)$

$- [t_l(i,j+1) - t_l(i+1,j) + R_1 * \{2h_l(i,j) + h_l(i,j+1) + h_l(i+1,j) - 2h_a(i,j) - 2h_a(i,j+1)\}]/\{1 + R_1 c_s(i,j)\}$

$h_l(i+1,j+1) = Psychrometrics.FNH\big(\mathrm{t}_l(i+1,j+1), Psychrometrics.FNxtr(\mathrm{t}_l(i+1,j+1),100)\big)$

温度 t,(i, j)における飽和空気の比エンタルピの温度こう配 $(\partial \mathrm{h}_l/\partial h_t)$ を$\mathrm{c}_s[i][j]$とする。

$\mathrm{c}_s[i][j] = \{h_l(i+1,j+1) - h_l(i,j)\}/\{t_l(i+1,j+1) - t(i,j)\}$

$h_a(i+1,j+1) = h(i,j+1) - h(i+1,j) + h(i,j) - R_2$

$* \{t_l(i+1,j+1) + t_l(i+1,j) + t_l(i,j+1) - t_l(i,j)\}$

--- step 4 --- III-36

$TW_{out} = \{t_l(1,n+1)/2 + t_l(m+1,n+1)/2 + \sum_{i=2}^{m} t_l(i,n+1)\}/m$

$HA_{out} = \{h_a(m+1{,}1)/2 + h_a(m+1,n+1)/2 + \sum_{i=2}^{m} h_a(m+1,j)\}/n$

watOutCD.setTemp(t_watOutCD);

double wb_airOut = Psychrometrics.FNDbrh(100, h_airOutEA *1000.);

FNDbrh(double RH, double H) // 相対湿度、エンタルピーから乾球温度を計算する

$WB_{out} = Psychrometrics.FNDbrh(100, HA_{out} * 1000.)$

$DB_{out} = Psychrometrics.FNDbhw(HA_{out} * 1000., WB_{out})$

$X_{out} = Psychrometrics.FNXth(HA_{out} * 1000., WB_{out})$

⑤冷却塔補給水量の計算（機器特性より）

1) 蒸発損失水量の計算

$$MW_{mw\_E} = (TW_{out} - TW_{in}) \times MW_{cd} \times \frac{1}{600}$$

2) 飛散損失水量（キャリーオーバ量）の計算

$MW_{mw\_C} = 0.005 \times MW_{cd}$

3) ブローダウン量の計算

$$MW_{mw\_B} = \frac{MW_{mw\_E}}{(CR_N - 1)} - MW_{mw\_C}$$

CR_N ：濃縮倍率

4) 冷却塔補給水量の計算

$\mathrm{MWR_{mw}} = \mathrm{MW_{mw\_E}} + \mathrm{MW_{mw\_C}} + \mathrm{MW_{mw\_B}}$

参考文献

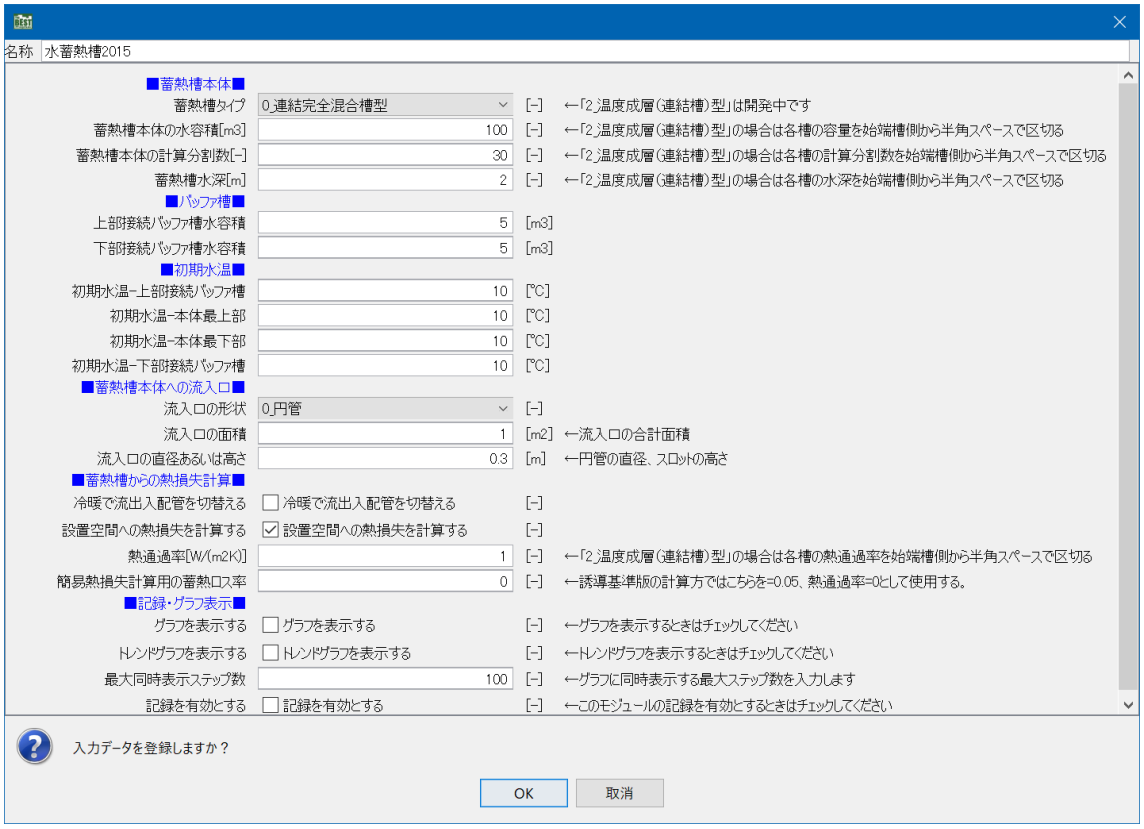
手塚俊一、藤田稔彦：湿り空気線図とその応用(4) III 冷却塔（その 1)、空気調和・衛生工学第 58 巻第 3 号 1984 年 pp59-68

「水蓄熱槽 2015」（場所：設備 2015／熱源 2015／）

モジュール名	水蓄熱槽 2015
クラス	ThermalStratificationStorageTank20101111

（1）入力画面

・スペック



（2）モジュールの概要

水蓄熱槽をモデル化したモジュールで、連結完全混合槽型、温度成層型（単体)、温度成層型（連結）の３タイプの計算が可能である。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	■蓄熱槽本体■					ー	ー		
2	蓄熱槽タイプ	String	stType	0_連結完全混合槽型、1_温度成層（単槽）型、2_温度成層（連結槽）型	[-]	ー	ー		←「2_温度成層（連結槽）型」は開発中です
3	蓄熱槽本体の水容積[m3]	String	stVolume	100	[-]	ー	ー		←「2_温度成層（連結槽）型」の場合は各槽の容量を始端槽側から半角スペースで区切る
4	蓄熱槽本体の計算分割数[-]	String	stNumber	30	[-]	ー	ー		←「2_温度成層（連結槽）型」の場合は各槽の計算分割数を始端槽側から半角スペースで区切る
5	蓄熱槽水深[m]	String	stDepth	2	[-]	ー	ー		←「2_温度成層（連結槽）型」の場合は各槽の水深を始端槽側から半角スペースで区切る
6	■バッファ槽■					ー	ー		
7	上部接続バッファ槽水容積	double	topVolume	5	[m3]	ー	0		
8	下部接続バッファ槽水容積	double	bottomVolume	5	[m3]	ー	0		
9	■初期水温■					ー	ー		
10	初期水温-上部接続バッファ槽	double	tempTopStart	10	[℃]	100	0		
11	初期水温-本体最上部	double	tempSTopStart	10	[℃]	100	0		
12	初期水温-本体最下部	double	tempSBottomStart	10	[℃]	100	0		
13	初期水温-下部接続バッフ	double	tempBottomStart	10	[℃]	100	0		

	ァ槽								
14	■蓄熱槽本体への流入口■					—	—		
15	流入口の形状	String	shapeInlet	0_円管、1_スロット	[-]	—	—		
16	流入口の面積	double	areaInlet	1	[m2]	—	0		←流入口の合計面積
17	流入口の直径あるいは高さ	double	dInlet	0.3	[m]	—	0		←円管の直径、スロットの高さ
18	■蓄熱槽からの熱損失計算■					—	—		
19	冷暖で流出入配管を切替える	boolean	isChangePipe	FALSE	[-]	—	—		
20	設置空間への熱損失を計算する	boolean	isCalcHeatLoss	TRUE	[-]	—	—		
21	熱通過率[W/(m2K)]	double	heatTransferCoefficient	1	[-]	—	—		←「2_温度成層（連結槽）型」の場合は各槽の熱通過率を始端槽側から半角スペースで区切る
22	簡易熱損失計算用の蓄熱ロス率	double	ch1HeatLossRate	0	[-]	—	—		←誘導基準版の計算方ではこちらを=0.05、熱通過率=0 として使用する。
23	■記録・グラフ表示■					—	—		
24	グラフを表示する	boolean	isGVisible	FALSE	[-]	—	—		←グラフを表示するときはチェックしてください
25	トレンドグラフを表示する	boolean	isGVisibleTrend	FALSE	[-]	—	—		←トレンドグラフを表示するときはチェックしてください
26	最大同時表示ステップ数	int	maxItemCount	100	[-]	—	—		←グラフに同時表示する最大ステップ数を入力します
27	記録を有効とする	boolean	isRecord	FALSE	[-]	—	—		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	On/Off 信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	運転モード	L1_modIn	modIn	-	制御	制御モード	入口	
4	冷温水熱源側入口	L0_watInCH1	watInCH1	-	状態	水	入口	
5	冷温水二次側入口	L0_watInCH2	watInCH2	-	状態	水	入口	
6	冷温水熱源側出口	L0_watOutCH1	watOutCH1	-	状態	水	出口	
7	冷温水二次側出口	L0_watOutCH2	watOutCH2	-	状態	水	出口	
8	空気	L0_airIn	airIn	-	状態	空気	入口	蓄熱槽の周囲の空気。熱損失計算に使用。
9	水槽（本体）	L0_wblOutTanks	wblOutTanks	-	状態	水塊	出口	
10	始端槽（Top）	L0_wblOutTankTop	wblOutTankTop	-	状態	水塊	出口	
11	終端槽（Bottom）	L0_wblOutTankBottom	wblOutTankBottom	-	状態	水塊	出口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	TST_Message#-#-	メッセージ	—	メッセージ
2	TST_T_watInCH1#℃#温度	冷温水入口温度　熱源側	℃	入口
3	TST_T_watInCH2#℃#温度	冷温水入口温度　二次側	℃	入口
4	TST_T_watOutCH1#℃#温度	冷温水出口温度　熱源側	℃	出口
5	TST_T_watOutCH2#℃#温度	冷温水出口温度　二次側	℃	出口
6	TST_M_watInCH1#g/s#質量流量	冷温水入口流量　熱源側	g/s	入口
7	TST_M_watInCH2#g/s#質量流量	冷温水入口流量　二次側	g/s	入口
8	TST_M_watOutCH1#g/s#質量流量	冷温水出口流量　熱源側	g/s	出口
9	TST_M_watOutCH2#g/s#質量流量	冷温水出口流量　二次側	g/s	出口
10	TST_Top 温度#℃#温度	始端槽（Top）の水温	℃	My
11	TST_Bottom 温度#℃#温度	終端槽（Bottom）の水温	℃	My
12	TST_＊ 温度#℃#温度	計算の分割槽 No＊の水温	℃	My
13	TST_Top 顕熱蓄熱量(0℃基準)#J#熱量	始端槽（Top）の顕熱蓄熱量（0℃基準）	J	My
14	TST_Bottom 顕熱蓄熱量(0℃基準)#J#熱量	終端槽（Bottom）の顕熱蓄熱量（0℃基準）	J	My
15	TST_＊ 顕熱蓄熱量(0℃基準)#J#熱量	計算の分割槽 No＊の顕熱蓄熱量（0℃基準）	J	My
16	TST_ST 全体（顕熱蓄熱量(0℃基準)）#J#熱量	蓄熱槽全体の顕熱蓄熱量（0℃基準）	J	My
17	TST_1 次側からの熱量#J#熱量	蓄熱槽への１次側からの熱量	J	My
18	TST_2 次側からの熱量#J#熱量	蓄熱槽への２次側からの熱量	J	My
19	TST_1 次側の累積熱量#J#熱量	蓄熱槽への１次側からの累積熱量	J	My
20	TST_2 次側の累積熱量#J#熱量	蓄熱槽への２次側からの累積熱量	J	My
21	TST_1 次側の累積熱量+2 次側の累積熱量#J#熱量	蓄熱槽への１次側からの累積熱量＋２次側からの累積熱量	J	My
22	TST_ST 全体（外部から蓄熱槽への出入り熱量の変化）#J#熱量	外部から蓄熱槽への出入り熱量の変化	J	My
23	TST_Top 次温度#℃#温度	始端槽（Top）の次の計算ステップの水温	℃	My
24	TST_Botoom 次温度#℃#温度	終端槽（Bottom）の次の計算ステップの水温	℃	My
25	TST_ST 全体（ heatLossSum ）#J#熱量	槽からの熱損失合計	J	My
26	TST_ST 全体（補正熱量）#J#熱量	熱収支のチェックによる補正熱量	J	My
27	TST_ST 全体（補正熱量 2）#J#熱量	熱収支のチェック２による補正熱量	J	My
28	TST_ST 全体（顕熱蓄熱量(0℃基準)変化量）#J#熱量	蓄熱槽全体の顕熱蓄熱量（0℃基準）の変化量	J	My
29	TST_ST 全体（平均補正温度）#℃#温度	熱収支のチェックによる槽水温の平均補正温度	℃	My

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

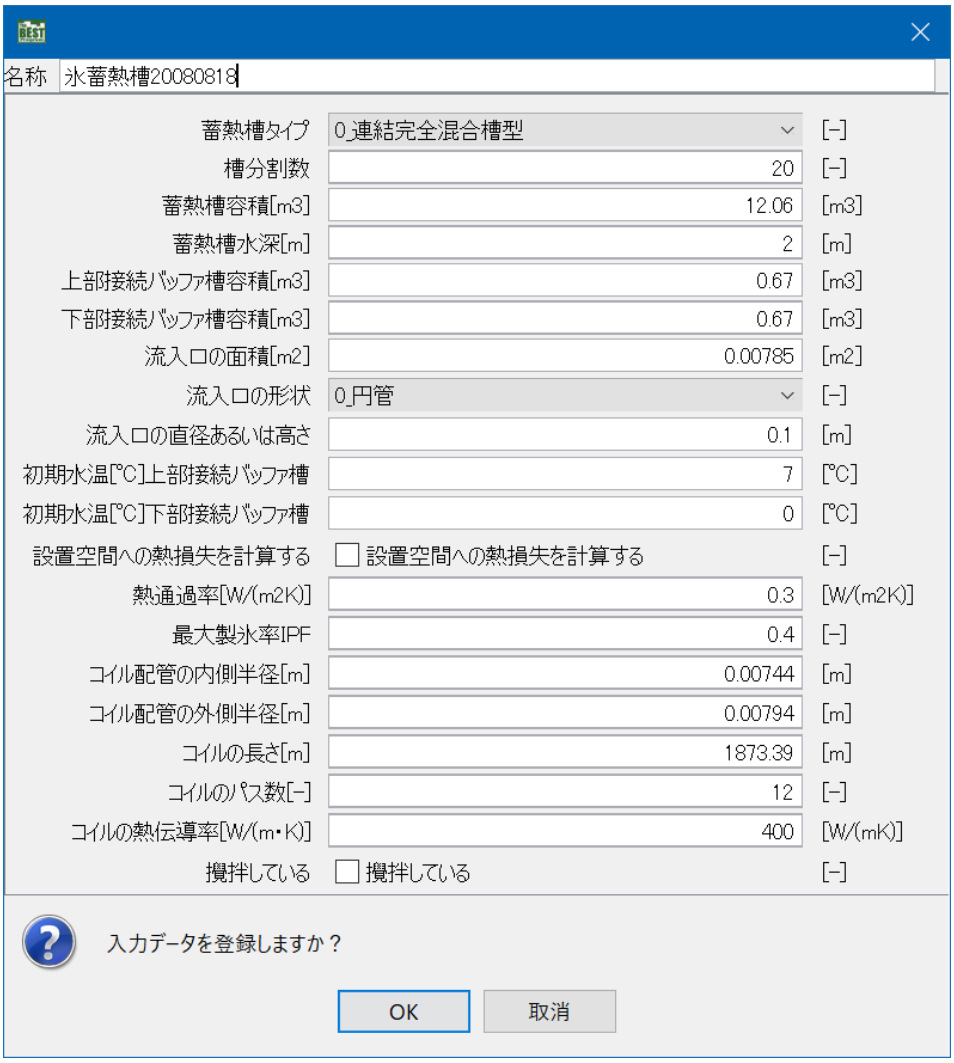
特になし。

「氷蓄熱槽 20080818」（場所：空調・換気設備／蓄熱システム／）

モジュール名	氷蓄熱槽 20080818
クラス	ThermalStratificationStorageTankICE20080818

（1）入力画面

・スペック



（2）モジュールの概要

外融式の氷蓄熱槽のモジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

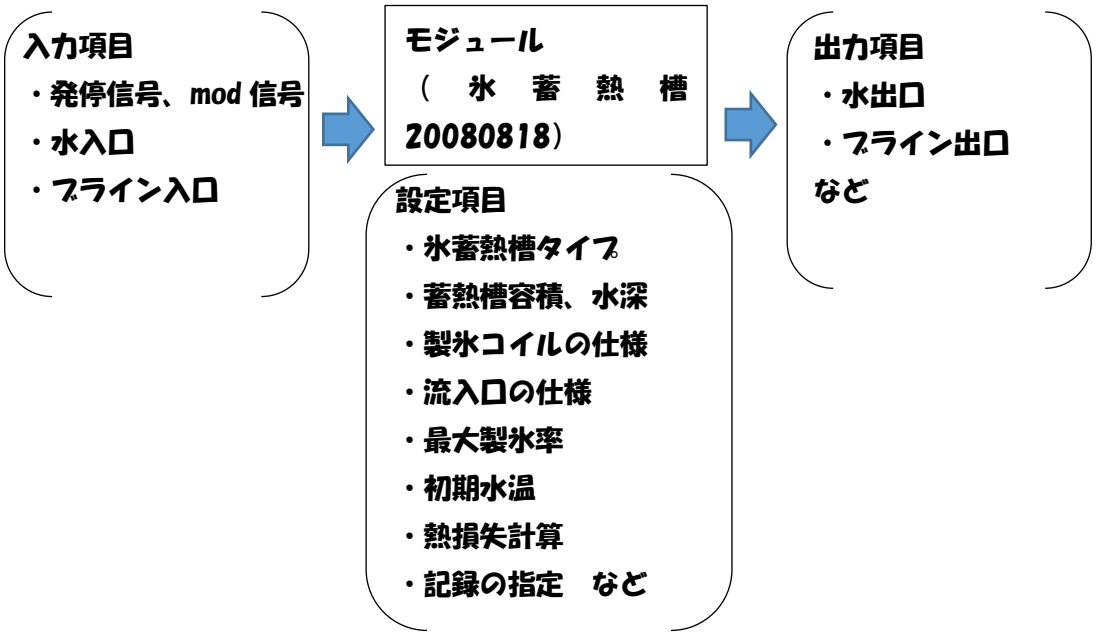


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	蓄熱槽タイプ	String	stType	0_連結完全混合槽型、1_温度成層（単槽）型、2_温度成層(連結槽)型	[-]	ー	ー		
2	槽分割数	double	stNumber	20	[-]	ー	0		
3	蓄熱槽容積[m3]	double	stVolume	12.06	[m3]	ー	0		
4	蓄熱槽水深[m]	double	stDepth	2	[m]	ー	0		
5	上部接続バッファ槽容積[m3]	double	topVolume	0.67	[m3]	ー	0		
6	下部接続バッファ槽容積[m3]	double	bottomVolume	0.67	[m3]	ー	0		
7	流入口の面積[m2]	double	areaInlet	0.00785	[m2]	ー	0		
8	流入口の形状	String	shapeInlet	0_円管、1_スロット	[-]	ー	ー		
9	流入口の直径あるいは高さ	double	dInlet	0.1	[m]	ー	0		
10	初期水温[℃]上部接続バッファ槽	double	tempTopStart	7	[℃]	100	0		
11	初期水温[℃]下部接続バッファ槽	double	tempBottomStart	0	[℃]	100	0		
12	設置空間への熱損失を計算する	boolean	isCalcHeatLoss	TRUE	[-]	ー	ー		
13	熱通過率[W/(m2K)]	double	heatTransferCoefficient	0.3	[W/(m2K)]	ー	0		
14	最大製氷率 IPF	double	maxIPF	0.4	[-]	1	0		
15	コイル配管の内側半径[m]	double	radiusInCoil	0.00744	[m]	ー	0		

16	コイル配管の外側半径[m]	double	radiusOutCoil	0.00794	[m]	―	0		
17	コイルの長さ[m]	double	lengthCoil	1873.39	[m]	―	0		
18	コイルのパス数[-]	double	passCoil	12	[-]	―	1		
19	コイルの熱伝導率[W/(m・K)]	double	thermalConductivityCoil	400	[W/(mK)]	―	0		
20	撹拌している	boolean	isStirUp	FALSE	[-]	―	―		

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	recOut	-	記録	メモリ	出口	
2	On/Off 信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	運転モード	L1_modeIn	modeIn	-	制御	制御モード	入口	
4	ブライン入口	L0_briInCH1	briInCH1	-	状態	ブライン	入口	
5	冷温水入口	L0_watInCH2	watInCH2	-	状態	水	入口	
6	ブライン出口	L0_briOutCH1	briOutCH1	-	状態	ブライン	出口	
7	冷温水出口	L0_watOutCH2	watOutCH2	-	状態	水	出口	
8	空気	L0_airIn	airIn	-	状態	空気	入口	氷蓄熱槽の周囲の空気。熱損失計算にしようする。
9	氷蓄熱部分	L0_wcbOutTanks	wcbOutTanks	-	状態	コイル水塊	出口	
10	始端槽（Top）	L0_wblOutTankTop	wblOutTankTop	-	状態	水塊	出口	
11	終端槽（Bottom）	L0_wblOutTankBottom	wblOutTankBottom	-	状態	水塊	出口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	TSTMessage#-#-	メッセージ	—	メッセージ
2	T_briInCH1#℃#温度	ブライン入口温度	℃	入口
3	T_watInCH2#℃#温度	冷温水入口温度	℃	入口
4	T_briOutCH1#℃#温度	ブライン出口温度	℃	出口
5	T_watOutCH2#℃#温度	冷温水出口温度	℃	出口
6	M_briInCH1#g/s#質量流量	ブライン入口流量	g/s	入口
7	M_watInCH2#g/s#質量流量	冷温水入口流量	g/s	入口
8	M_briOutCH1#g/s#質量流量	ブライン出口流量	g/s	出口
9	M_watOutCH2#g/s#質量流量	冷温水出口流量	g/s	出口
10	ST-Top 温度#℃#温度	始端槽（Top）の水温	℃	My
11	ST-Bottom 温度#℃#温度	終端槽（Bottom）の水温	℃	My
12	ST-＊ 温度#℃#温度	計算の分割槽 No＊の水温	℃	My
13	ST-＊ 等価温度#℃#温度	計算の分割槽 No＊の等価水温(氷の蓄熱分を水温に換算して水温から減じた温度)	℃	My
14	ST-Top 顕熱蓄熱量(0℃基準)#J#熱量	始端槽（Top）の顕熱蓄熱量（0℃基準）	J	My
15	ST-Bottom 顕熱蓄熱量(0℃基準)#J#熱量	終端槽（Bottom）の顕熱蓄熱量（0℃基準）	J	My
16	ST-＊ 顕熱蓄熱量(0℃基準)#J#熱量	計算の分割槽 No＊の顕熱蓄熱量（0℃基準）	J	My
17	ST 全体　顕熱蓄熱量(0℃基準)#J#熱量	氷蓄熱槽全体の顕熱蓄熱量（0℃基準）	J	My
18	ST-＊ 潜熱蓄熱量#J#熱量	計算の分割槽 No＊の潜熱蓄熱量	J	My
19	ST 全体　潜熱蓄熱量#J#熱量	氷蓄熱槽全体の潜熱蓄熱量	J	My
20	ST 全体　1 次側からの熱量#J#熱量	氷蓄熱槽への１次側からの熱量	J	My
21	ST 全体　2 次側からの熱量#J#熱量	氷蓄熱槽への２次側からの熱量	J	My
22	ST 全体　1 次側の累積熱量#J#熱量	氷蓄熱槽への１次側からの累積熱量	J	My
23	ST 全体　2 次側の累積熱量#J#熱量	氷蓄熱槽への２次側からの累積熱量	J	My
24	ST 全体　1 次側の累積熱量+2 次側の累積熱量#J#熱量	氷蓄熱槽への１次側からの累積熱量＋２次側からの累積熱量	J	My
25	ST 全体　外部から蓄熱槽への出入り熱量の変化#J#熱量	外部から氷蓄熱槽への出入り熱量の変化	J	My

（7）計算フロー・計算内容

このモジュールの製氷・融解についての基本アルゴリズムは次の文献を参考とした。

中原信夫、山羽基、氷蓄熱層の熱特性に関する研究　第 3 報　アイスオンコイル型氷蓄熱層のシミュレーションモデルと蓄熱槽効率推定表の作成、1994 年 10 月　空気調和・衛生工学会論文集 No56.PP13-24

また、流速と保有水量とから計算時間間隔を自己分割するアルゴリズムについては次のプログラムに関する文献を参考とした。

財団法人　ヒートポンプ・蓄熱センター

TESEP-W　水蓄熱槽最適プログラム　マニュアル（理論編）　平成 12 年 7 月

（8）データ範囲と範囲外の取扱い

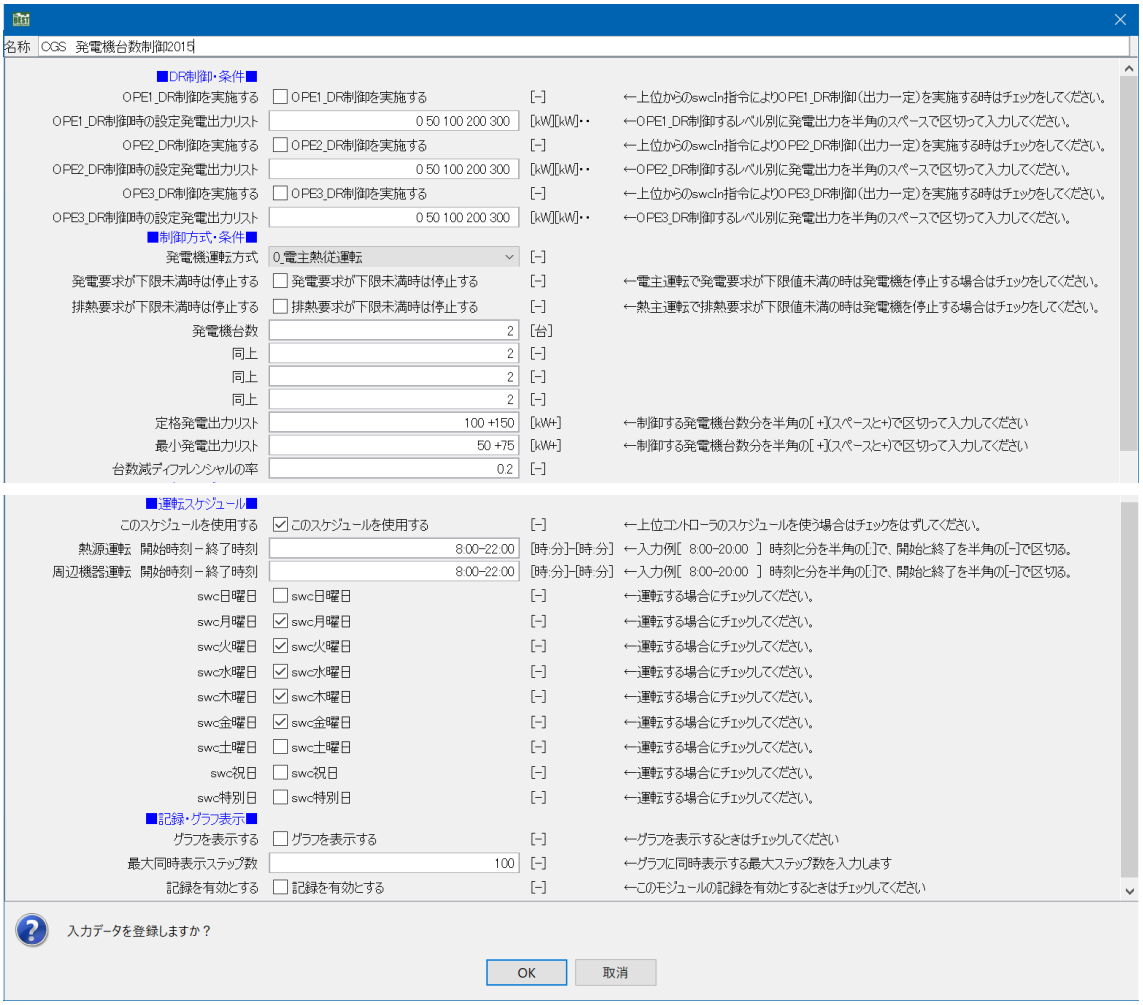
特になし。

「CGS　発電機台数制御 2015」（場所：設備 2015／コージェネ 2015／）

モジュール名	CGS　発電機台数制御 2015
クラス	ControlnUnitsGenOperatingModule20100101

（1）入力画面

・スペック



（2）モジュールの概要

CGS の複数台の発電機に対して、要求発電量あるいは要求排熱量に応じた発電機の運転台数制御を行う。

運転台数の制御方式として、電主熱従運転、熱主電従運転、発電出力一定運転の３通りから指定できる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

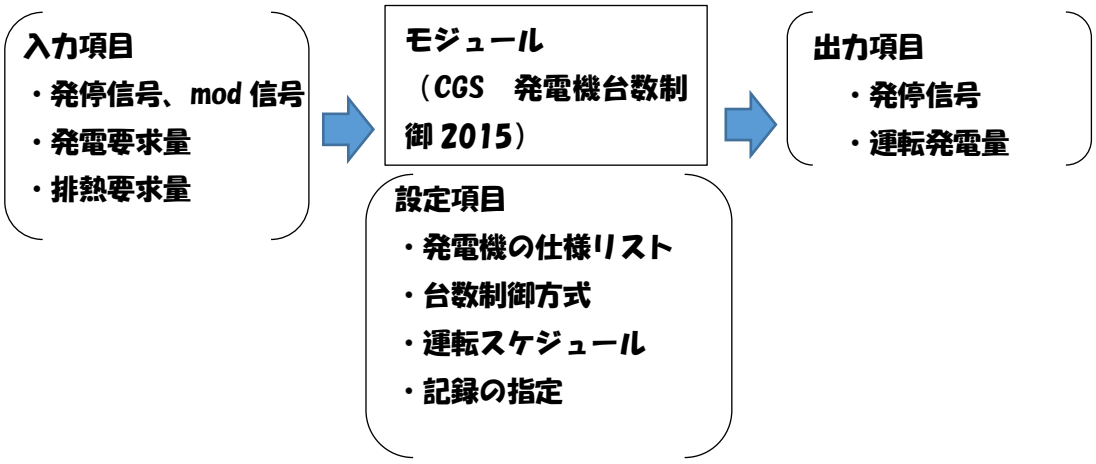


図 1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図1における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	■DR制御・条件■					―	―		
2	OPE1_DR制御を実施する	boolean	isDRCtrl[0]	FALSE	[-]	―	―		←上位からのswcIn指令によりOPE1_DR制御（出力一定）を実施する時はチェックをしてください。
3	OPE1_DR 制御時の設定発電出力リスト	String	drGenPower[0]	0 50 100 200 300	[kW][kW]・・	―	―		←OPE1_DR制御するレベル別に発電出力を半角のスペースで区切って入力してください。
4	OPE2_DR制御を実施する	boolean	isDRCtrl[1]	FALSE	[-]	―	―		←上位からのswcIn指令によりOPE2_DR制御（出力一定）を実施する時はチェックをしてください。
5	OPE2_DR 制御時の設定発電出力リスト	String	drGenPower[1]	0 50 100 200 300	[kW][kW]・・	―	―		←OPE2_DR制御するレベル別に発電出力を半角のスペースで区切って入力してください。
6	OPE3_DR制御を実施する	boolean	isDRCtrl[2]	FALSE	[-]	―	―		←上位からのswcIn指令によりOPE3_DR制御（出力一定）を実施する時はチェックをしてください。
7	OPE3_DR 制御時の設定発電出力リスト	String	drGenPower[2]	0 50 100 200 300	[kW][kW]・・	―	―		←OPE3_DR制御するレベル別に発電出力を半角のスペースで区切って入力してください。
8	■制御方式・条件■					―	―		
9	発電機運転方式	String	controlType	0_電主熱従運転、1_熱主電従運転、2_発電出力一定運転	[-]	―	―		
10	発電要求が下限未満時は停止する	boolean	isStopLowerDemandGen	FALSE	[-]	―	―		←電主運転で発電要求が下限値未満の時は発電機を停止する場合はチェックをしてください。
11	排熱要求が下限未満時は停止する	boolean	isStopLowerDemandHea	FALSE	[-]	―	―		←熱主運転で排熱要求が下限値未満の時は発電機を停止する場合はチェックをしてください。
12	発電機台数	int	numberOfGen	2	[台]	―	0		
13	同上	int	numberOfGen	2	[-]	―	0		
14	同上	int	numberOfGen	2	[-]	―	0		

15	同上	int	numberOfGen	2	[-]	ー	0		
16	定格発電出力リスト	String	maxGenPower	100 +150	[kW+]	ー	ー		←制御する発電機台数分を半角の[+](スペースと+)で区切って入力してください
17	最小発電出力リスト	String	minGenPower	50 +75	[kW+]	ー	ー		←制御する発電機台数分を半角の[+](スペースと+)で区切って入力してください
18	台数減ディファレンシャルの率	double	changeGenPower	0.2	[-]	1	0		
19	//制御タイプ	String	controlType	0_熱量、1_電力	[-]	ー	ー		
20	■運転スケジュール■					ー	ー		
21	このスケジュールを使用する	boolean	isUseThisOpe	TRUE	[-]	ー	ー		←上位コントローラのスケジュールを使う場合はチェックをはずしてください。
22	熱源運転　開始時刻ー終了時刻	String	HSOpe_START_END	8:00-22:00	[時:分]-[時:分]	ー	ー		←入力例[8:00-20:00] 時刻と分を半角の[:]で、開始と終了を半角の[-]で区切る。
23	周辺機器運転　開始時刻ー終了時刻	String	SUOpe_START_END	8:00-22:00	[時:分]-[時:分]	ー	ー		←入力例[8:00-20:00] 時刻と分を半角の[:]で、開始と終了を半角の[-]で区切る。
24	//冷房　開始月日ー終了月日	String		5/1-11/30	[月/日]-[月/日]	ー	ー		←入力例[5/1-11/30] 月と日を半角の[/]で、開始と終了を半角の[-]で区切る。
25	//暖房　開始月日ー終了月日	String		12/1-4/30	[月/日]-[月/日]	ー	ー		←入力例[12/1-4/30] 月と日を半角の[/]で、開始と終了を半角の[-]で区切る。
26	swc 日曜日	boolean	isCalcSun	FALSE	[-]	ー	ー		←運転する場合にチェックしてください。
27	swc 月曜日	boolean	isCalcMon	TRUE	[-]	ー	ー		←運転する場合にチェックしてください。
28	swc 火曜日	boolean	isCalcTue	TRUE	[-]	ー	ー		←運転する場合にチェックしてください。
29	swc 水曜日	boolean	isCalcWed	TRUE	[-]	ー	ー		←運転する場合にチェックしてください。
30	swc 木曜日	boolean	isCalcThu	TRUE	[-]	ー	ー		←運転する場合にチェックしてください。
31	swc 金曜日	boolean	isCalcFri	TRUE	[-]	ー	ー		←運転する場合にチェックしてください。
32	swc 土曜日	boolean	isCalcSat	FALSE	[-]	ー	ー		←運転する場合にチェックしてください。
33	swc 祝日	boolean	isCalcHo	FALSE	[-]	ー	ー		←運転する場合にチェックしてください。

			l						
34	swc 特別日	boolean	isCalcSpc	FALSE	[-]	ー	ー		←運転する場合にチェックしてください。
35	■記録・グラフ表示■					ー	ー		
36	グラフを表示する	boolean	isGVisible	FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
37	最大同時表示ステップ数	int	maxItemCount	100	[-]	ー	ー		←グラフに同時表示する最大ステップ数を入力します
38	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号入口	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	運転モード入口	L1_modIn	modIn	-	制御	制御モード	入口	
4	発停信号出口	L1_swcOut	swcOut	-	制御	OnOff 信号	出口	
5	運転モード出口	L1_modOut	modOut	-	制御	制御モード	出口	
6	発電機[*]への発停信号出口	L1_swcOutGen[]	swcOutGen[]	-	制御	OnOff 信号	出口	発電機台数分のノード数
7	発電機[*]への運転モード出口	L1_modOutGen[]	modOutGen[]	-	制御	制御モード	出口	発電機台数分のノード数
8	発電デマンド入口	L0_valInDemandele	valInDemandele	-	値	値	入口	
9	発電機[*]への発電デマンド出口	L0_valOutDemandele[]	valOutDemandele[]	-	値	値	出口	発電機台数分のノード数
10	排熱制御量入口	L0_valInCtrlHE	valInCtrlHE	-	値	値	入口	
11	排熱温水循環ポンプへの発停信号出口	L1_swcOutPumpHE[]	swcOutPumpHE[]	-	制御	OnOff 信号	出口	発電機台数分のノード数

（6）記録項目

No.	項目	出力内容	単位	分類
1	ConnUnitsGenMessage#-#-	メッセージ	—	メッセージ
2	ConnUnitsGen_swcIn#-#-	上位からの発停信号	—	入口
3	ConnUnitsGen_modeIn#-#-	上位からの運転モード信号	—	入口
4	ConnUnitsGen_swcOut#-#-	下位への発停信号	—	出口
5	ConnUnitsGen_valInDemandELE#-#-	発電電力デマンド	—	入口
6	ConnUnitsGen_valInCtrlHE#-#-	要求排熱量比	—	入口
7	ConnUnitsGen_swcOutGen[*]#-#-	発電機[*]への発停信号	—	出口
8	ConnUnitsGen_modOutGen[*]#-#-	発電機[*]への運転モード信号	—	出口
9	ConnUnitsGen_valOutDemandele[*]#W#-	発電機[*]への要求発電量	W	出口
10	ConnUnitsGen_swcOutPumpHE[*]#-#-	排熱循環ポンプ[*]への発停信号	—	出口

（7）計算フロー・計算内容

.

（8）データ範囲と範囲外の取扱い

参考ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.Map;

//import jp.or.ibec.best.DO.BestAir;
//import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.building.manager.ScheduleManager;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;


/**
 * @author HIROSHI NINOMIYA /20080909
 * ControlnUnitsModule 発電機n台の台数制御コントローラです
 *   ControlHSModuleへ、ON-OFFスケジュールを制御します
 *
 *   swc
 *   mod
 *
 * 20080610 祝日の判定　ー1～ー7を追加
 *
 * 記録の有効無効判断を追加
 * 20090101 グラフ表示ステップ数追加
 * 20100101 熱主電従運転制御を追加
 * 20150312 DR制御実施する　対応
 */
public class ControlnUnitsGenOperatingModule20100101
                      extends AbstractBestModule implements IBestMetaModule{

   private final String moduleName = "(ControlnUnitsGenOperatingModule20100101) ";

   //接続ノード
   private final String S_NODE_valInDemandele  = "L0_valInDemandele";//電力デマンド//20100101
   private final String S_NODE_valInCtrlHE     = "L0_valInCtrlHE";//排熱制御量　//20100101//L0_valInCtrlEH

   //デマンドの出力
   private String[] S_NODE_valOutDemandele;
   //
   //出口　接続名称は　L0_eleOut[0],L0_eleOut[1]・・・・とする
   //

   //制御ノード　　swc**　運転状態(0:停止、1:運転)
   private final String C_NODE_swcIn ="L1_swcIn";//上位からのswc信号
   private final String C_NODE_modIn ="L1_modIn";//上位からのmode信号
   private final String C_NODE_swcOut="L1_swcOut";//下位制御へのswc信号
   private final String C_NODE_modOut="L1_modOut";//下位制御へのmod信号
   //
   private String[] C_NODE_swcOutGen;//発電機へのswc信号
   private String[] C_NODE_modOutGen;//発電機へのmod信号

   private String[] C_NODE_swcOutPumpHE;//循環ポンプ専用 = "L1_swcOutPumpHE";//循環ポンプ専用

   //記録ノード
   private final String R_NODE = "L2_recOut";

   private final String RECORD_Message   = "ConnUnitsGenMessage#-#-";
   private final String RECORD_swcIn     = "ConnUnitsGen_swcIn#-#-";
   private final String RECORD_modIn     = "ConnUnitsGen_modeIn#-#-";
   private final String RECORD_swcOut    = "ConnUnitsGen_swcOut#-#-";
   private final String RECORD_valInDemandELE = "ConnUnitsGen_valInDemandELE#-#-";
```

```
private final String RECORD_valInCtrlHE    = "ConnUnitsGen_valInCtrlHE#-#-";
private  String[] RECORD_swcOutGen;//= "ConnUnitsGen_swcOutGen#-#-";
private  String[] RECORD_modOutGen;//= "ConnUnitsGen_modOutGen#-#-";
private  String[] RECORD_valOutDemandele;//= "ConnUnitsGen_valOutDemandele#W#-";
private  String[] RECORD_swcOutPumpHE;

//外部定義項目
//仕様
private final String SPEC_name      = "名称";         //外部定義項目

private final String SPEC_isOPE1_DRCtrl = "isOPE1_DR制御実施";
private final String SPEC_isOPE2_DRCtrl = "isOPE2_DR制御実施";
private final String SPEC_isOPE3_DRCtrl = "isOPE3_DR制御実施";
private final String SPEC_OPE1_DRGenPowerList = "OPE1_DR設定発電出力リスト[kW]";
private final String SPEC_OPE2_DRGenPowerList = "OPE2_DR設定発電出力リスト[kW]";
private final String SPEC_OPE3_DRGenPowerList = "OPE3_DR設定発電出力リスト[kW]";

private final String SPEC_isUseThisOpe = "このスケジュールを使用する";

private final String SPEC_ControlType = "発電機運転方式";//20100101

private final String SPEC_isStopLowerDemandGen = "発電要求が下限未満時は停止する";//20121015
private final String SPEC_isStopLowerDemandHea = "排熱要求が下限未満時は停止する";//20121015

private final String SPEC_NumberOfGen = "[]発電機台数";
private final String SPEC_MinGenPower = "最小発電出力[kW+]";
private final String SPEC_MaxGenPower = "定格発電出力[kW+]";
//流量で制御  No1を優先運転とする
private final String SPEC_Differential = "台数減ディファレンシャルの率[-]";//[0-1]

private final String SPEC_GenOpe_START_END = "発電機運転  開始時刻ー終了時刻";
private final String SPEC_SUOpe_START_END  = "周辺機器運転  開始時刻ー終了時刻";

//private final String SPEC_HSOpe_START = "熱源運転開始時刻";
//private final String SPEC_HSOpe_END   = "熱源運転終了時刻";
//private final String SPEC_SUOpe_START = "周辺機器運転開始時刻";
//private final String SPEC_SUOpe_END   = "周辺機器運転終了時刻";
//private final String SPEC_COOL_START  = "冷房開始月日";
//private final String SPEC_COOL_END    = "冷房終了月日";
//private final String SPEC_HEAT_START  = "暖房開始月日";
//private final String SPEC_HEAT_END    = "暖房終了月日";

//曜日スケジュール
private final String SPEC_isCalcSun    = "swc日曜日";
private final String SPEC_isCalcMon    = "swc月曜日";
private final String SPEC_isCalcTue    = "swc火曜日";
private final String SPEC_isCalcWed    = "swc水曜日";
private final String SPEC_isCalcThu    = "swc木曜日";
private final String SPEC_isCalcFri    = "swc金曜日";
private final String SPEC_isCalcSat    = "swc土曜日";
private final String SPEC_isCalcHol    = "swc祝日";
private final String SPEC_isCalcSpc    = "swc特別日";
//
private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
private final String SPEC_maxItemCount = "最大同時表示ステップ数";//最大同時表示ステップ数
private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする


private StringBuffer message = new StringBuffer();    //メッセージ

private BestValue valInDemandele  = null;//電力デマンド20100101
private BestValue valInCtrlHE = null;//排熱制御量//20100101

private BestValue[] valOutDemandele;
private double[] maxGenPower;
private double[] minGenPower;

private double sumMaxGenPower;//20100101
private double sumMinGenPower;//20100101
private double changeGenPower;//20100101
```

```
//仕様など
private String  name = null;  //機器名称
private int swcIn;
private int modIn;
private int swcOut;
private int modOut;

private int[] swcOutGen;
private int[] modOutGen;

private int[] swcOutPumpHE;

private boolean[] isDRCtrl = {false, false, false};//"isRD制御実施";
private double[][] drGenPower = new double[3][];//"RD設定発電出力リスト[kW]";


private boolean isStopLowerDemandGen;// = "発電要求が下限未満時は停止する";//20121015
private boolean isStopLowerDemandHea;// = "排熱要求が下限未満時は停止する";//20121015
private boolean isUseThisOpe;// = "このスケジュールを使用する";
private int numberOfGen;

private double differential; //"台数減ディファレンシャル";  //[g/s]
private String  controlType;// = "制御タイプ";  //流量、熱量
private String maxGenPowers = null;
private String minGenPowers = null;
//
private boolean isGVisible = false;//このグラフを表示する=true
private int  maxItemCount = 100;//最大同時表示ステップ数
private boolean isRecord   = false;//記録を有効とする=true

private String HSOpe_START_END   = null;//"熱源運転　開始時刻ー終了時刻";
private String SUOpe_START_END   = null;//"周辺機器運転　開始時刻ー終了時刻";

private String HSOpe_START = null;
private String HSOpe_END = null;
private String SUOpe_START = null;
private String SUOpe_END = null;

private boolean isCalcSun = true;
private boolean isCalcMon = true;
private boolean isCalcTue = true;
private boolean isCalcWed = true;
private boolean isCalcThu = true;
private boolean isCalcFri = true;
private boolean isCalcSat = true;
private boolean isCalcHol = true;
private boolean isCalcSpc = true;

//内部変数
private int hsOpeStartHour;
private int hsOpeStartMinute;
private int hsOpeEndHour;
private int hsOpeEndMinute;
private int suOpeStartHour;
private int suOpeStartMinute;
private int suOpeEndHour;
private int suOpeEndMinute;

private int modMy;

private int swcMy;
private int swcSUMy;

private boolean isDemandEle = true;//電主熱従運転//20100101

//グラフ表示など
private GraphJFrameC_nUnitsOperating20090101 gC_nUnitsOperating = null;

@Override
public void setProfile(BestSpecs spec) {
   if( spec == null ){
      return;
```

```
}
Map<String, String> map = spec.getSpec();
if( map == null ){
   return;
}

//名称を取得
this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

//
this.isUseThisOpe = spec.getSpecValue( this.SPEC_isUseThisOpe, true );

//numberOfGenを取得
if (null != map.get(this.SPEC_NumberOfGen)) {
   this.numberOfGen
   = Integer.parseInt( (String)map.get( this.SPEC_NumberOfGen ));
   //出口ｎ系統のBestWaterの初期化
   this.valOutDemandele = new BestValue[this.numberOfGen];
   this.S_NODE_valOutDemandele = new String[this.numberOfGen];
   this.maxGenPower = new double[this.numberOfGen];
   this.minGenPower = new double[this.numberOfGen];
   this.RECORD_swcOutGen = new String[this.numberOfGen];
   this.RECORD_modOutGen = new String[this.numberOfGen];
   this.RECORD_valOutDemandele = new String[this.numberOfGen];
   this.RECORD_swcOutPumpHE = new String[this.numberOfGen];
   this.swcOutGen = new int[this.numberOfGen];
   this.modOutGen = new int[this.numberOfGen];
   this.swcOutPumpHE = new int[this.numberOfGen];
   this.C_NODE_swcOutGen = new String[this.numberOfGen];
   this.C_NODE_modOutGen = new String[this.numberOfGen];
   this.C_NODE_swcOutPumpHE = new String[this.numberOfGen];

   for( int i=0; i<this.numberOfGen; i++ ){
      //this.eleOut[i] = new BestElectricity();
      //this.eleOut[i] = new BestElectricity( 0, 0, 200, 0, 3, 50);
      this.S_NODE_valOutDemandele[i] = new String( "LO_valOutDemandele[" + i + "]" );
      this.maxGenPower[i] = 0;
      this.minGenPower[i] = 0;
      //
      this.C_NODE_swcOutGen[i] = "L1_swcOutGen["+i+"]";
      this.C_NODE_modOutGen[i] = "L1_modOutGen["+i+"]";
      this.C_NODE_swcOutPumpHE[i] = "L1_swcOutPumpHE["+i+"]";
      this.RECORD_swcOutGen[i] = "ConnUnitsGen_swcOutGen["+i+"]#-#-";
      this.RECORD_modOutGen[i] = "ConnUnitsGen_modOutGen["+i+"]#-#-";
      this.RECORD_valOutDemandele[i] = "ConnUnitsGen_valOutDemandele["+i+"]#W#-";
      this.RECORD_swcOutPumpHE[i] = "ConnUnitsGen_swcOutPumpHE["+i+"]#-#-";
   }
}
//SPEC_Differential = "台数減ディファレンシャル";  //[g/s]
this.differential = spec.getSpecValue( this.SPEC_Differential, 0.2 );

//SPEC_ControlType = "発電機運転方式";//20100101
this.controlType = spec.getSpecValue( this.SPEC_ControlType, "0_電主熱従運転" );

//controlType
//0_電主熱従運転
//1_熱種電従運転
//2_発電出力一定運転
//

//20100101
if( this.controlType.equals("0_電主熱従運転")){
   this.isDemandEle = true;
}else{
   this.isDemandEle = false;
}

//private final String SPEC_isOPE1_DRCtrl = "isOPE1_DR制御実施";
this.isDRCtrl[0] = spec.getSpecValue( this.SPEC_isOPE1_DRCtrl, false );
//private final String SPEC_isOPE2_DRCtrl = "isOPE2_DR制御実施";
this.isDRCtrl[1] = spec.getSpecValue( this.SPEC_isOPE2_DRCtrl, false );
//private final String SPEC_isOPE3_DRCtrl = "isOPE3_DR制御実施";
```

```
this.isDRCtrl[2] = spec.getSpecValue( this.SPEC_isOPE3_DRCtrl, false );

//private final String SPEC_RDGenPowerList = "DR設定発電出力リスト[kW]";
String str = spec.getSpecValue( this.SPEC_OPE1_DRGenPowerList, "100" );
str = str.replace( "_", " " );
String[] strs = null;
strs = str.split( "[ /]" );
this.drGenPower[0] = new double[strs.length];
for( int i=0; i<strs.length; i++ ){
   this.drGenPower[0][i] = Double.parseDouble( strs[i] ) * 1000.;
   //System.out.println( "drGenPower[0]["+i+"]="+this.drGenPower[0][i] );
   if( this.isRecord ){
      this.message.append("()drGenPower[0]["+i+"]="+this.drGenPower[0][i] );
   }
}

str = spec.getSpecValue( this.SPEC_OPE2_DRGenPowerList, "100" );
str = str.replace( "_", " " );
strs = null;
strs = str.split( "[ /]" );
this.drGenPower[1] = new double[strs.length];
for( int i=0; i<strs.length; i++ ){
   this.drGenPower[1][i] = Double.parseDouble( strs[i] ) * 1000.;
   //System.out.println( "drGenPower[1]["+i+"]="+this.drGenPower[1][i] );
   if( this.isRecord ){
      this.message.append("()drGenPower[1]["+i+"]="+this.drGenPower[1][i] );
   }
}

str = spec.getSpecValue( this.SPEC_OPE3_DRGenPowerList, "100" );
str = str.replace( "_", " " );
strs = null;
strs = str.split( "[ /]" );
this.drGenPower[2] = new double[strs.length];
for( int i=0; i<strs.length; i++ ){
   this.drGenPower[2][i] = Double.parseDouble( strs[i] ) * 1000.;
   //System.out.println( "drGenPower[2]["+i+"]="+this.drGenPower[2][i] );
   if( this.isRecord ){
      this.message.append("()drGenPower[2]["+i+"]="+this.drGenPower[2][i] );
   }
}

//isStopLowerDemandを取得
this.isStopLowerDemandGen = spec.getSpecValue( this.SPEC_isStopLowerDemandGen, false );

//isStopLowerDemandを取得
this.isStopLowerDemandHea = spec.getSpecValue( this.SPEC_isStopLowerDemandHea, false );

//
this.maxGenPowers = spec.getSpecValue( this.SPEC_MaxGenPower, "100" );
this.maxGenPowers = this.maxGenPowers.replace( "_", " +" );

//System.out.println( "ConGen nUnits maxGenPowers = "+this.maxGenPowers );

//
this.minGenPowers = spec.getSpecValue( this.SPEC_MinGenPower, "100" );
this.minGenPowers = this.minGenPowers.replace( "_", " +" );

//"熱源運転　開始時刻ー終了時刻";
this.HSOpe_START_END = spec.getSpecValue( this.SPEC_GenOpe_START_END, "8:00-20:00" );

//"周辺機器運転　開始時刻ー終了時刻";
this.SUOpe_START_END = spec.getSpecValue( this.SPEC_SUOpe_START_END, "8:00-20:00" );

this.isCalcSun = spec.getSpecValue( this.SPEC_isCalcSun, false );

this.isCalcMon = spec.getSpecValue( this.SPEC_isCalcMon, false );

this.isCalcTue = spec.getSpecValue( this.SPEC_isCalcTue, false );

this.isCalcWed = spec.getSpecValue( this.SPEC_isCalcWed, false );
```

```
        this.isCalcThu = spec.getSpecValue( this.SPEC_isCalcThu, false );

        this.isCalcFri = spec.getSpecValue( this.SPEC_isCalcFri, false );

        this.isCalcSat = spec.getSpecValue( this.SPEC_isCalcSat, false );

        this.isCalcHol = spec.getSpecValue( this.SPEC_isCalcHol, false );

        this.isCalcSpc = spec.getSpecValue( this.SPEC_isCalcSpc, false );

        //isGVisibleを取得
        this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

        // 最大同時表示ステップ数
        this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

    }


    public void initialize(IBestStateMessage stateNodes,
          IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
          IBestDomainSchedule schedule) {
        super.sm = stateNodes   ;     //状態ノードを取得
        super.cm = commandNodes ;     //制御ノードを取得
        super.rm = recordNodes  ;     //記録ノードを取得

        //接続ノード　入口チェック
        //接続ノード　valOutDemandele
        for( int i=0; i<this.numberOfGen; i++ ){
            this.valOutDemandele[i] = BestValue.bindnode( super.transferMapState, super.sm,
this.S_NODE_valOutDemandele[i] );
        }

        //private BestValue valInDemandele  = null;//電力デマンド20100101
        this.valInDemandele = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInDemandele );

        //S_NODE_valInCtrlHE = "LO_valInCtrlHE";//排熱制御量  //20100101
        this.valInCtrlHE = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInCtrlHE );

        this.sumMaxGenPower = 0;//20100101
        this.sumMinGenPower = 0;//20100101

        for( int i=0; i<this.numberOfGen; i++ ){
            if( this.maxGenPowers.split( " +", this.numberOfGen  )[i] != null ){
                this.maxGenPower[i] = Double.parseDouble( this.maxGenPowers.split( " +" )[i] );
                this.maxGenPower[i] *= 1000.;//kW -> W
                //System.out.println( "conNunits max ["+i+"]="+this.maxGenPower[i]);

                this.sumMaxGenPower += this.maxGenPower[i];//20100101
            }else{
                this.maxGenPower[i] = 0;
            }
            //
            if( this.minGenPowers.split( " +", this.numberOfGen )[i] != null ){
                this.minGenPower[i] = Double.parseDouble( this.minGenPowers.split( " +" )[i] );
                this.minGenPower[i] *= 1000.;//kW -> W

                this.sumMinGenPower += this.minGenPower[i];//20100101
            }else{
                this.minGenPower[i] = 0;
            }
        }

        //this.changeGenPower = this.sumMaxGenPower - this.sumMinGenPower;//20100101
        this.changeGenPower = this.sumMaxGenPower - this.minGenPower[0];//20121015


        super.cm.setCommand(super.getConnectionNode(this.C_NODE_swcIn), 0 );
        super.cm.setCommand(super.getConnectionNode(this.C_NODE_modIn), 0 );
```

```
      for( int i=0; i<this.numberOfGen; i++ ){
         super.cm.setCommand(super.getConnectionNode( this.C_NODE_swcOutGen[i] ), Airswc.OFF );
         super.cm.setCommand(super.getConnectionNode( this.C_NODE_modOutGen[i] ), Airmod.E_CHARGE );
         super.cm.setCommand(super.getConnectionNode( this.C_NODE_swcOutPumpHE[i] ), Airswc.OFF );
      }

      this.HSOpe_START = this.HSOpe_START_END.split( "-" )[0];
      this.HSOpe_END   = this.HSOpe_START_END.split( "-" )[1];
      this.SUOpe_START = this.SUOpe_START_END.split( "-" )[0];
      this.SUOpe_END   = this.SUOpe_START_END.split( "-" )[1];

      hsOpeStartHour   = Integer.parseInt( this.HSOpe_START.split( ":" )[0] );
      hsOpeStartMinute = Integer.parseInt( this.HSOpe_START.split( ":" )[1] );
      hsOpeEndHour     = Integer.parseInt( this.HSOpe_END.split( ":" )[0] );
      hsOpeEndMinute   = Integer.parseInt( this.HSOpe_END.split( ":" )[1] );

      suOpeStartHour   = Integer.parseInt( this.SUOpe_START.split( ":" )[0] );
      suOpeStartMinute = Integer.parseInt( this.SUOpe_START.split( ":" )[1] );
      suOpeEndHour     = Integer.parseInt( this.SUOpe_END.split( ":" )[0] );
      suOpeEndMinute   = Integer.parseInt( this.SUOpe_END.split( ":" )[1] );

      //グラフ表示の準備
      if( this.isGVisible ){
         gC_nUnitsOperating = new GraphJFrameC_nUnitsOperating20090101( this.name, this.maxItemCount,  0, 0,
this.numberOfGen );
      }
   }

   private int[] dateTime; //0:年、1:月、2:日、3:年間通算日、4:曜日、5:時、6:分、7:秒
   private int dayOfWeek;


   public void outputs(){
      if(super.sm == null || super.cm == null ){
         if( this.isRecord ){
            this.message.append( "(E) sm cm rm がない");
         }
         return;
      }

      this.swcIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn) );

      this.valInDemandele = (BestValue) super.sm.getState( super.getConnectionNode( this.S_NODE_valInDemandele ));

      int numOPE = Airswc.getMinOnOPEnum( this.swcIn );
      //System.out.println("gen numOpe="+numOPE );
      if(numOPE>0 && numOPE<4  && this.isDRCtrl[ numOPE-1 ] ){
         //DR制御をする場合
         int nDRL = Airswc.getMaxOnDRLevel( this.swcIn );
         nDRL = ( nDRL < this.drGenPower[ numOPE-1 ].length )? nDRL : this.drGenPower[ numOPE-1 ].length;
         if( nDRL > 0 && !(this.drGenPower[ numOPE-1 ][nDRL-1]<0 )){
            //DR制御レベル別の発電出力をセット
            this.valInDemandele.setValue( this.drGenPower[ numOPE-1 ][nDRL-1] );
            if( this.isRecord ){
               this.message.append( "(C)DR発令制御レベル["+nDRL+"]発電出力"+this.drGenPower[ numOPE-1 ][nDRL-1]+"[W]にセ
ット");
            }
         }else{
            if( this.controlType.equals( "0_電主熱従運転" ) ){//0_電主熱従運転 //20100101
               //何もしない
            }else if( this.controlType.equals( "1_熱主電従運転" ) ){//1_熱主電従運転 //20100101
               this.valInCtrlHE = (BestValue) super.sm.getState( super.getConnectionNode( this.S_NODE_valInCtrlHE ));
               this.valInDemandele.setValue( this.minGenPower[0] + this.changeGenPower * Math.min( 1.0,
this.valInCtrlHE.getValue() ));//20121015 -->minGenPower[0]
            }else if( this.controlType.equals( "2_発電出力一定運転" ) ){//20121015
               this.valInDemandele.setValue( this.sumMaxGenPower );
            }
            if( this.isRecord ){
               this.message.append( "(C)DR発令制御レベル["+nDRL+"]発電出力"+this.controlType );
            }

         }
```

```
      }else{

         if( this.controlType.equals( "0_電主熱従運転" ) ){//0_電主熱従運転 //20100101
            //何もしない
         }else if( this.controlType.equals( "1_熱主電従運転" ) ){//1_熱主電従運転 //20100101
            this.valInCtrlHE = (BestValue) super.sm.getState( super.getConnectionNode( this.S_NODE_valInCtrlHE ));
            //this.valInDemandele.setValue( this.minGenPower[0] + this.changeGenPower * Math.min( 1.0,
this.valInCtrlHE.getValue() ));//20121015 -->minGenPower[0]
            this.valInDemandele.setValue( this.sumMaxGenPower * Math.min( 1.0, this.valInCtrlHE.getValue() ));//20121015
-->minGenPower[0]
         }else if( this.controlType.equals( "2_発電出力一定運転" ) ){//20121015
            this.valInDemandele.setValue( this.sumMaxGenPower );
         }
         if( this.isRecord ){
            this.message.append( "(C)"+ this.controlType );
         }
      }

      //System.out.println( "(controlnUnitsGemOpe) valInCtrlHE, eleDemand = " + this.valInCtrlHE.getValue() + " / "+
this.valInDemandele.getValue() );

      //20121101
      this.modIn  = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn) );

      if( this.isUseThisOpe ){
         //このスケジュールを使用する
         this.dateTime = BestTimeManager.getDateWeatherTime();
         //dayOfWeek = HolidayManager.getDayOfWeek(dateTime[3]);  20080116
         this.dayOfWeek = ScheduleManager.getHolidayManager().getDayOfWeek(); //20080116


         if( !checkDayOfWeek( this.dayOfWeek ) ){
            //曜日チェック  運転OFF
            this.swcMy = Airswc.onOFF( this.swcMy );
            this.swcSUMy = Airswc.onOFF( this.swcSUMy );
         }else{
            //曜日チェック  運転ON
            //運転時間帯チェック
            //acOpe START END / Hour Minute
            if( Airswc.checkSWCOffOn( this.hsOpeStartHour, this.hsOpeStartMinute,
                  this.hsOpeEndHour,   this.hsOpeEndMinute, this.dateTime ) ){
               this.swcMy = Airswc.onON( this.swcMy );
            }else{
               this.swcMy = Airswc.onOFF( this.swcMy );
            }

            //suOpe START END / Hour Minute
            if( Airswc.checkSWCOffOn( this.suOpeStartHour, this.suOpeStartMinute,
                  this.suOpeEndHour,   this.suOpeEndMinute, this.dateTime ) ){
               this.swcSUMy = Airswc.onON( this.swcSUMy );
            }else{
               this.swcSUMy = Airswc.onOFF( this.swcSUMy );
            }
         }

      }else{
         //上位系のスケジュールを使用する
         this.swcIn
         = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ) );
         this.modIn
         = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn) );

         this.swcMy   = this.swcIn;
         this.swcSUMy = this.swcIn;
         this.modMy   = this.modIn;
      }

      //this.message.append(
      //    "(C)swcIn="+swcIn+"/swcMy="+swcMy+"/modeIn="+modIn+"/modeMy="+modMy );

      //System.out.println( this.name+"(C)swcIn="+swcIn+"/swcMy="+swcMy+"/modeIn="+modIn+"/modeMy="+modMy );
```

```
      if( Airswc.isREMOTE( this.swcMy ) ){
         if( this.isRecord ){
            this.message.append( "(C)遠方運転指令" );
         }
         //すべて動かす
         this.swcOut = Airswc.onON( this.swcOut );
         for( int i=0; i<this.numberOfGen; i++ ){
            this.swcOutGen[i] = Airswc.onON( this.swcOutGen[i] );
            this.swcOutPumpHE[i] = Airswc.onON( this.swcOutPumpHE[i] );
         }

      }else if( Airswc.isON( this.swcMy ) ){
         if( this.isRecord ){
            this.message.append( "(C)運転指令" );
         }
         this.swcOut = Airswc.onON( this.swcOut );
         //台数制御のコントロールを有効とするとき
         double sumGenPower = 0;

         //発電、排熱要求が下限未満の時は停止する20121015
         if( ( this.isStopLowerDemandHea && this.valInCtrlHE.getValue() <= 0 )
               || ( this.isStopLowerDemandGen && this.valInDemandele.getValue() < this.minGenPower[0] ) ){
            for( int i=0; i<this.numberOfGen; i++ ){
               this.swcOutGen[i] = Airswc.onOFF( this.swcOutGen[i] );
               this.valOutDemandele[i].setValue( 0 );
               this.swcOutPumpHE[i] = Airswc.onOFF( this.swcOutPumpHE[i] );
            }

            if( this.isRecord ){
               this.message.append( "(C)排熱要求下限未満で停止 valInCtrlHE=" +
AirFormat.df_2( this.valInCtrlHE.getValue() ) +" valInDemandele="+AirFormat.df_2( this.valInDemandele.getValue()));
            }

         }else{

            for( int i=0; i<this.numberOfGen; i++ ){

               sumGenPower += this.minGenPower[i];

               if( sumGenPower <= this.valInDemandele.getValue() ){
                  this.swcOutGen[i] = Airswc.onON( this.swcOutGen[i] );
                  this.valOutDemandele[i].setValue( this.minGenPower[i] );
                  this.swcOutPumpHE[i] = Airswc.onON( this.swcOutPumpHE[i] );
               }else{
                  //if( sumGenPower - this.minGenPower[i] < this.valInDemandele.getValue() ){
                  // this.swcOutGen[i] = Airswc.onON( this.swcOutGen[i] );
                  // this.valOutDemandele[i].setValue( this.minGenPower[i] );
                  // this.swcOutPumpHE[i] = Airswc.onON( this.swcOutPumpHE[i] );
                  if( this.isStopLowerDemandGen || this.isStopLowerDemandHea ){
                     //下限未満で停止するときi番目は運転できない
                     this.swcOutGen[i] = Airswc.onOFF( this.swcOutGen[i] );
                     this.valOutDemandele[i].setValue( 0. );
                     this.swcOutPumpHE[i] = Airswc.onOFF( this.swcOutPumpHE[i] );
                     if( i > 1 ){//20140416
                        if( sumGenPower - this.minGenPower[i] -this.minGenPower[i-1] + this.maxGenPower[i-1] >
this.valInDemandele.getValue() ){
                           //
                           this.valOutDemandele[i-1].setValue( this.valInDemandele.getValue() - ( sumGenPower -
this.minGenPower[i] -this.minGenPower[i-1] ));
                        }else{
                           this.valOutDemandele[i-1].setValue( this.maxGenPower[i-1] );
                        }
                     }

                  }else if( sumGenPower - this.minGenPower[i] < this.valInDemandele.getValue() ){
                     this.swcOutGen[i] = Airswc.onON( this.swcOutGen[i] );
                     this.valOutDemandele[i].setValue( this.minGenPower[i] );
                     this.swcOutPumpHE[i] = Airswc.onON( this.swcOutPumpHE[i] );
                  }else{
                     this.swcOutGen[i] = Airswc.onOFF( this.swcOutGen[i] );
```

```
                    this.valOutDemandele[i].setValue( 0 );
                    this.swcOutPumpHE[i] = Airswc.onOFF( this.swcOutPumpHE[i] );
                }
            }
        }
        if( sumGenPower < this.valInDemandele.getValue() ){
            for( int i=0; i<this.numberOfGen; i++ ){
                sumGenPower += ( this.maxGenPower[i] - this.minGenPower[i] );
                if( sumGenPower < this.valInDemandele.getValue() ){
                    this.swcOutGen[i] = Airswc.onON( this.swcOutGen[i] );
                    this.valOutDemandele[i].setValue( this.maxGenPower[i] );
                    this.swcOutPumpHE[i] = Airswc.onON( this.swcOutPumpHE[i] );
                }else{
                    if( sumGenPower - this.maxGenPower[i] + this.minGenPower[i] < this.valInDemandele.getValue() ){
                        this.swcOutGen[i] = Airswc.onON( this.swcOutGen[i] );
                        this.valOutDemandele[i].setValue( this.maxGenPower[i] - sumGenPower +
this.valInDemandele.getValue() );
                        this.swcOutPumpHE[i] = Airswc.onON( this.swcOutPumpHE[i] );
                    }else{
                    }
                }
            }
        }

        //循環ポンプ[0]は運転する
        this.swcOutPumpHE[0] = Airswc.onON( this.swcOutPumpHE[0] );

        //System.out.println( "nUnits Demand ="+this.valInDemandele.getValue() +"
/Gen1="+this.valOutDemandele[0].getValue()+" /Gen2="+this.valOutDemandele[1].getValue() );

    }else{
        if( this.isRecord ){
            this.message.append( "(C)停止指令" );
        }
        this.swcOut = Airswc.onOFF( this.swcOut );
        //すべて止める
        for( int i=0; i<this.numberOfGen; i++ ){
            this.swcOutGen[i] = Airswc.onOFF( this.swcOutGen[i] );
            this.valOutDemandele[i].setValue( 0 );
            this.swcOutPumpHE[i] = Airswc.onOFF( this.swcOutPumpHE[i] );
        }
    }

    //20121101
    this.modOut = this.modIn;

    //
    for( int i=0; i<this.numberOfGen; i++ ){
        super.sm.setState( super.getConnectionNode( this.S_NODE_valOutDemandele[i] ), this.valOutDemandele[i] );
        super.cm.setCommand( super.getConnectionNode(this.C_NODE_swcOutGen[i]), this.swcOutGen[i] );
        this.modOutGen[i] = this.modIn;
        super.cm.setCommand( super.getConnectionNode(this.C_NODE_modOutGen[i]), this.modOutGen[i] );
        super.cm.setCommand( super.getConnectionNode(this.C_NODE_swcOutPumpHE[i]), this.swcOutPumpHE[i] );
    }

    super.cm.setCommand( super.getConnectionNode(this.C_NODE_swcOut), this.swcOut );
    super.cm.setCommand( super.getConnectionNode(this.C_NODE_modOut), this.modOut );

    //

    //グラフデータ追加
    if( this.isGVisible ){

  ///**********************************************************************************************************************
*///
        gC_nUnitsOperating.addData( BestTimeManager.getDateWeatherTime(),
              this.valInDemandele.getValue(), this.valOutDemandele  );
    }

    //記録ノード
    if( this.isRecord  && super.rm != null ){
```

```
      this.record();
   }

   message.setLength(0);
}

/**
 * 記録
 */
private void record(){
   if( CheckPrintModule.isPrintMessage ){
      super.rm.setRecord(super.getConnectionNode(this.R_NODE),
         this.RECORD_Message, this.name, this.message.toString() );
   }

   if( CheckPrintModule.isPrintStateIn ){
      super.rm.setRecord(super.getConnectionNode(this.R_NODE),
            this.RECORD_swcIn, this.name, this.swcIn );
      super.rm.setRecord(super.getConnectionNode(this.R_NODE),
            this.RECORD_modIn, this.name, this.modIn );
      super.rm.setRecord(super.getConnectionNode(this.R_NODE),
            this.RECORD_valInDemandELE, this.name, AirFormat.df_4( this.valInDemandele.getValue()) );
      super.rm.setRecord(super.getConnectionNode(this.R_NODE),
            this.RECORD_valInCtrlHE, this.name, AirFormat.df_4( this.valInCtrlHE.getValue()) );
   }

   if( CheckPrintModule.isPrintStateOut ){
      super.rm.setRecord(super.getConnectionNode(this.R_NODE),
            this.RECORD_swcOut, this.name, this.swcOut );
      for( int i=0; i<this.numberOfGen; i++ ){
         super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_swcOutGen[i], this.name, this.swcOutGen[i] );
         super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_modOutGen[i], this.name, this.modOutGen[i] );
         super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_valOutDemandele[i], this.name, this.valOutDemandele[i].getValue() );//20090618
         super.rm.setRecord(super.getConnectionNode(this.R_NODE),
               this.RECORD_swcOutPumpHE[i], this.name, this.swcOutPumpHE[i] );
      }
   }

   message.setLength( 0 );

}

public void update() {

}


private boolean checkDayOfWeek( int dayOfWeek ){
   switch( dayOfWeek ){
   case -1:
   case -2:
   case -3:
   case -4:
   case -5:
   case -6:
   case -7:
      if( this.isCalcHol ) return true;
      break;
   case 0://20121130
      if( this.isCalcSpc ) return true;
      break;
   case 1:
      if( this.isCalcSun ) return true;
      break;
   case 2:
      if( this.isCalcMon ) return true;
      break;
   case 3:
      if( this.isCalcTue ) return true;
```

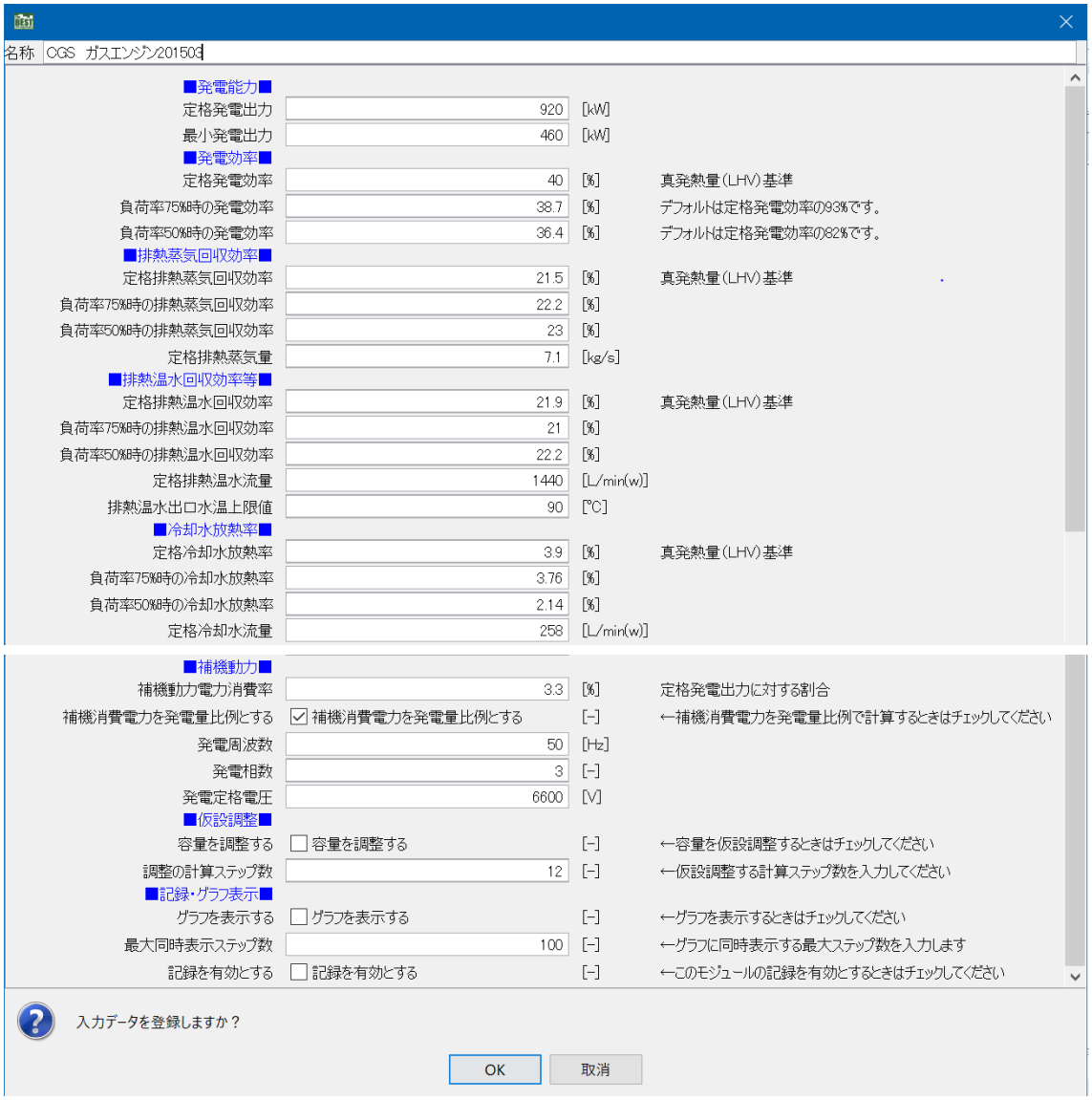
```
            break;
        case 4:
            if( this.isCalcWed ) return true;
            break;
        case 5:
            if( this.isCalcThu ) return true;
            break;
        case 6:
            if( this.isCalcFri ) return true;
            break;
        case 7:
            if( this.isCalcSat ) return true;
            break;
        default :
            return false;
        }
        return false;
    }


    public Object viewInternal(TestCommand cmd) {
        java.util.List<Object> result = new java.util.ArrayList<Object>();
        result.add(super.sm);
        result.add(super.cm);

        result.add(this.differential);
        result.add(this.sumMinGenPower);
        result.add(this.modMy);
        result.add( this.isDemandEle );

        return result;
    }


}
```

「CGS　ガスエンジン蒸気温水」（場所：設備 2015／コージェネ 2015／）

モジュール名	CGS　ガスエンジン蒸気温水
クラス	GasEngine201503

（1）入力画面

・スペック



（2）モジュールの概要

CGS で使用する発電機の一つで、ガスを燃料とする。

このガスエンジンモジュールは、発電時の余剰熱は排熱温水と排熱蒸気および燃焼ガスと冷却水から放熱する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

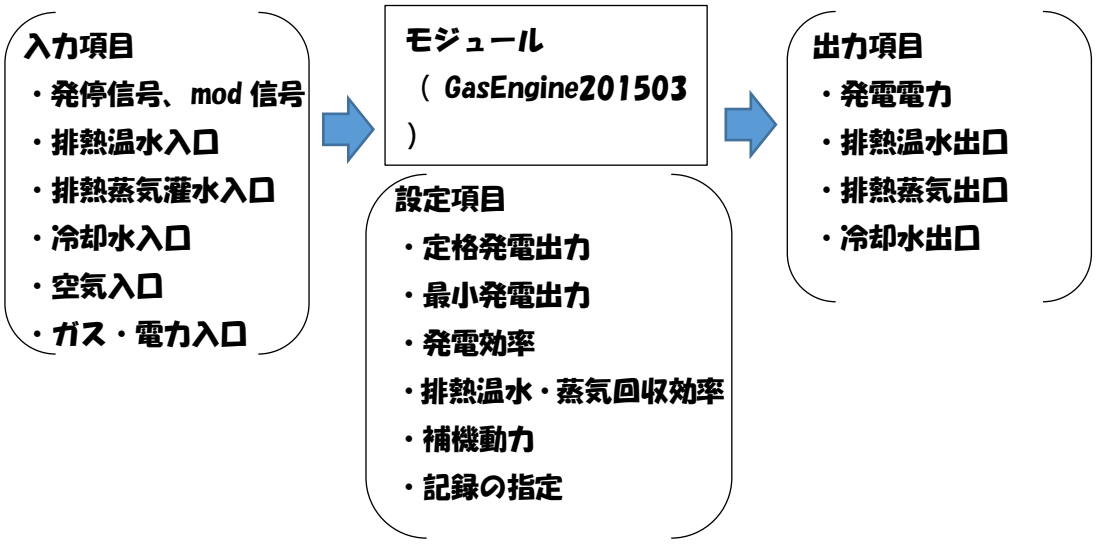


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図1における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	Name	ー	[-]	ー	ー		
1	■発電能力■					ー	ー		
2	定格発電出力	double	n_genePowerW	920	[kW]	ー	ー		
3	最小発電出力	double	minGenePowerW	460	[kW]	ー	ー		
4	■発電効率■					ー	ー		
5	定格発電効率	double	n_geneEffi	40	[%]	ー	ー		
6	負荷率 75%時の発電効率	double	p75geneEffi	38.7	[%]	ー	ー		デフォルトは定格発電効率の 93%です。
7	負荷率 50%時の発電効率	double	p50geneEffi	36.4	[%]	ー	ー		デフォルトは定格発電効率の 82%です。
8	■排熱蒸気回収効率■					ー	ー		
9	定格排熱蒸気回収効率	double	n_SrecvEffi	21.5	[%]	ー	ー		
10	負荷率 75%時の排熱蒸気回収効率	double	p75SrecvEffi	22.2	[%]	ー	ー		デフォルトは定格排熱回収効率の　%です。
11	負荷率 50%時の排熱蒸気回収効率	double	p50SrecvEffi	23	[%]	ー	ー		デフォルトは定格排熱回収効率の　%です。
12	定格排熱蒸気量	double		7.1	[kg/s]	ー	ー		
13	■排熱温水回収効率等■					ー	ー		
14	定格排熱温水回収効率	double	n_HrecvEffi	21.9	[%]	ー	ー		
15	負荷率 75%時の排熱温水回収効率	double	p75HrecvEffi	21	[%]	ー	ー		デフォルトは定格排熱回収効率の　%です。
16	負荷率 50%時の排熱温水回収効率	double	p50HrecvEffi	22.2	[%]	ー	ー		デフォルトは定格排熱回収効率の　%です。
17	定格排熱温水流量	double	n_m_watH	1440	[L/min(w)]	ー	ー		
18	排熱温水出口水温上限値	double	tMax_watOut	90	[℃]	ー	ー		

19	■冷却水放熱率■					―	―		
20	定格冷却水放熱率	double	n_DrecvEffi	3.9	[%]	―	―		
21	負荷率 75%時の冷却水放熱率	double	p75DrecvEffi	3.76	[%]	―	―		デフォルトは定格排熱回収効率の　%です。
22	負荷率 50%時の冷却水放熱率	double	p50DrecvEffi	2.14	[%]	―	―		デフォルトは定格排熱回収効率の　%です。
23	定格冷却水流量	double	n_m_watCD	258	[L/min(w)]	―	―		
24	■補機動力■					―	―		
25	補機動力電力消費率	double	n_subEleRate	3.3	[%]	―	―		
26	補機消費電力を発電量比例とする	boolean	isHokiHirei	TRUE	[-]	―	―		←補機消費電力を発電量比例で計算するときはチェックしてください
27	発電周波数	double	gen_frequency	50	[Hz]	―	―		
28	発電相数	int	gen_phase	3	[-]	―	―		
29	発電定格電圧	double	gen_voltage	6600	[V]	―	―		
30	■仮設調整■					―	―		
31	容量を調整する	boolean	isAdjust2012	FALSE	[-]	―	―		←容量を仮設調整するときはチェックしてください
32	調整の計算ステップ数	int	numAdjustSteps	12	[-]	―	1		←仮設調整する計算ステップ数を入力してください
33	■記録・グラフ表示■					―	―		
34	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
35	最大同時表示ステップ数	int	maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
36	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	
2	発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	発電要求値	L0_valInDemandPower	valInDemandPower	-	値	値	入口	
4	排温水出口	L0_watOut	watOut	-	状態	水	出口	
5	排温水入口	L0_watIn	watIn	-	状態	水	入口	
6	排熱蒸気出口	L0_steOut	steOut	-	状態	蒸気	出口	
7	排熱蒸気灌水入口	L0_watInS	watInS	-	状態	水	入口	
8	冷却水出口	L0_watOutCD	watOutCD	-	状態	水	出口	
9	冷却水入口	L0_watInCD	watInCD	-	状態	水	入口	
10	発電電力出口	L0_eleOut	eleOut	-	状態	電力	出口	
11	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	
12	ガス入口	L0_gasIn	gasIn	-	状態	ガス	入口	
13	空気入口	L0_airIn	airIn	-	状態	空気	入口	
14	空気出口	L0_airOut	airOut	-	状態	空気	出口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	GenGE_Message#-#-	メッセージ	—	メッセージ
2	GenGE_補機動力電力消費量#W#電力消費	補機動力の電力消費量	W	My
3	GenGE_発電量#W#発電電力	発電量	W	My
4	GenGE_発電効率#-#発電効率	発電効率	—	My
5	GenGE_燃料消費量#W#燃料消費	燃料消費量	W	My
6	GenGE_排熱温水回収量#W#排熱回収	排熱温水の回収量	W	My
7	GenGE_排熱温水回収効率#-#効率	排熱温水の回収効率	—	My
8	GenGE_排熱蒸気回収量#W#排熱回収	排熱蒸気の回収量	W	My
9	GenGE_排熱蒸気回収効率#-#効率	排熱蒸気の回収効率	—	My
10	GenGE_放熱量#W#放熱	放熱量	W	My
11	GenGE_放熱率#-#効率	放熱率	—	My
12	GenGE_冷却水出口温度#℃#温度	冷却水の出口温度	℃	出口
13	GenGE_冷却水入口温度#℃#温度	冷却水の入り口温度	℃	入口
14	GenGE_冷却水流量#g/s#質量流量	冷却水の流量	g/s	出口
15	GenGE_総合効率#-#効率	総合効率	—	My
16	GenGE_発電負荷率#-#負荷率	発電負荷率	—	My
17	GenGE_排熱温水出口温度#℃#温度	排熱温水の出口温度	℃	出口
18	GenGE_排熱温水入口温度#℃#温度	排熱温水の入り口温度	℃	入口
19	GenGE_排熱温水流量#g/s#質量流量	排熱温水の流量	g/s	出口
20	GenGE_発電要求量#W#電力	発電要求量	W	入口
21	GenGE_調整発電容量#W#電力	自動調整の発電容量	W	調整

（7）計算フロー・計算内容

.

（8）データ範囲と範囲外の取扱い

参考ソース

```
package jp.or.ibec.best.domain.sample.cogen;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestGas;
import jp.or.ibec.best.DO.BestSteam;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.cogen.AbstractPowerGenerator;
import jp.or.ibec.best.domain.sample.air.AirFormat;
import jp.or.ibec.best.domain.sample.air.Airswc;
import jp.or.ibec.best.domain.sample.air.CheckPrintModule;
import jp.or.ibec.best.domain.sample.air.GraphJFrameGasEngine20080909;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * ガス発電機
 * @author HIROSHI NINOMIYA /20081010 /20090101
 *
 * 蒸気・温水発生型ガスエンジン//201503
 *
 */
public class GasEngine201503 extends AbstractPowerGenerator {

   private final String moduleName = "(GasEngine201503) ";

   // 状態ノード
   private final String S_NODE_valInDemandPower = "L0_valInDemandPower";      // 要求電力
   private final String S_NODE_watInS  = "L0_watInS"; // 排熱蒸気入口
   private final String S_NODE_steOut  = "L0_steOut"; // 排熱蒸気出口
   private final String S_NODE_watIn   = "L0_watIn";  // 排熱温水入口
   private final String S_NODE_watOut  = "L0_watOut"; // 排熱温水出口
   private final String S_NODE_watInCD = "L0_watInCD";   // 冷却水入口
   private final String S_NODE_watOutCD= "L0_watOutCD";// 冷却水出口
   private final String S_NODE_airIn   = "L0_airIn";  // 給気
   private final String S_NODE_airOut  = "L0_airOut"; // 排気
   private final String S_NODE_eleIn   = "L0_eleIn";  // 消費電力
   private final String S_NODE_eleOut  = "L0_eleOut"; // 発電電力
   private final String S_NODE_gasIn   = "L0_gasIn";  // 燃料

// private Double valInDemmandPower = null;
   private BestWater watInS = null;
   private BestSteam steOut = null;
   private BestWater watIn  = null;
   private BestWater watOut = null;
   private BestWater watInCD  = null;
   private BestWater watOutCD = null;
   private BestAir airIn  = null;
   private BestAir airOut = null;
   private BestElectricity eleIn  = null;
   private BestElectricity eleOut = null;
   private BestGas gasIn = null;

   private BestValue valInDemandPower = null;
   // 制御ノード
   private final String C_NODE = "L1_swcIn";// 運転制御
```

```
// 記録ノード
private final String RO_NODE = "L2_recOut";

// 外部定義項目のキー
private final String SPEC_name   = "名称";         //外部定義項目

private final String SPEC_n_genePowerW  = "定格発電出力[W]";
private final String SPEC_minGenePowerW = "最小発電出力[W]";

private final String SPEC_n_geneEffi    = "定格発電効率[-]";
private final String SPEC_p75geneEffi   = "負荷率75%時の発電効率[-]";
private final String SPEC_p50geneEffi   = "負荷率50%時の発電効率[-]";

private final String SPEC_n_SrecvEffi    = "定格排熱蒸気回収効率[-]";
private final String SPEC_p75SrecvEffi   = "負荷率75%時の排熱蒸気回収効率[-]";
private final String SPEC_p50SrecvEffi   = "負荷率50%時の排熱蒸気回収効率[-]";

private final String SPEC_n_HrecvEffi    = "定格排熱温水回収効率[-]";
private final String SPEC_p75HrecvEffi   = "負荷率75%時の排熱温水回収効率[-]";
private final String SPEC_p50HrecvEffi   = "負荷率50%時の排熱温水回収効率[-]";
private final String SPEC_tMax_watOut    = "排熱温水出口水温上限値[℃]";

private final String SPEC_n_DrecvEffi    = "定格冷却水放熱率[-]";
private final String SPEC_p75DrecvEffi   = "負荷率75%時の冷却水放熱[-]";
private final String SPEC_p50DrecvEffi   = "負荷率50%時の冷却水放熱[-]";

private final String SPEC_n_m_watH   = "定格排熱温水流量[g/s]";
private final String SPEC_n_m_watCD  = "定格冷却水流量[g/s]";

private final String SPEC_n_subEleRate  = "補機動力電力消費率[-]";
private final String SPEC_isHokiHire = "is補機発電量比例";        //
private final String SPEC_gen_frequency = "発電周波数[Hz]";
private final String SPEC_gen_phase     = "発電相数[-]";
private final String SPEC_gen_voltage   = "発電定格電圧[V]";
//private final String SPEC_ITEM_13 = "モジュール識別名";
//
private final String SPEC_isGVisible = "グラフを表示する";
private final String SPEC_maxItemCount = "最大同時表示ステップ数";
private final String SPEC_isRecord = "記録を有効とする";

private final String SPEC_isAdjust2012   = "容量を調整する";
private final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";

//記録項目
private final String RECORD_message  = "GenGE_Message#-#-" ;
private final String RECORD_subEleW = "GenGE_補機動力電力消費量#W#電力消費";
private final String RECORD_demandEleW = "GenGE_発電量#W#発電電力";
private final String RECORD_geneEffi = "GenGE_発電効率#-#発電効率";

private final String RECORD_fuelInputW = "GenGE_燃料消費量#W#燃料消費";

private final String RECORD_qhrecv     = "GenGE_排熱温水回収量#W#排熱回収";
private final String RECORD_qhrecvEffi = "GenGE_排熱温水回収効率#-#効率";

private final String RECORD_qSrecv     = "GenGE_排熱蒸気回収量#W#排熱回収";
private final String RECORD_qSrecvEffi = "GenGE_排熱蒸気回収効率#-#効率";

private final String RECORD_qDrecv     = "GenGE_放熱量#W#放熱";
private final String RECORD_qDrecvEffi = "GenGE_放熱率#-#効率";
private final String RECORD_t_watOutCD = "GenGE_冷却水出口温度#℃#温度";
private final String RECORD_t_watInCD  = "GenGE_冷却水入口温度#℃#温度";
private final String RECORD_m_watOutCD = "GenGE_冷却水流量#g/s#質量流量";

private final String RECORD_allEffi = "GenGE_総合効率#-#効率";

private final String RECORD_geneLoadFactor = "GenGE_発電負荷率#-#負荷率";
private final String RECORD_t_watOut = "GenGE_排熱温水出口温度#℃#温度";
private final String RECORD_t_watIn  = "GenGE_排熱温水入口温度#℃#温度";
private final String RECORD_m_watOut = "GenGE_排熱温水流量#g/s#質量流量";
//private final String DEBUG_ID7 = "運転状態";              // [boolean]

private final String RECORD_in_GenDemand  = "GenGE_発電要求量#W#電力";
```

```
private final String RECORD_out_GenAdjust = "GenGE_調整発電容量#W#電力";

// 外部定義項目
//private double MaxGenerate;             // 最大発電量     [kW]
private double minGenePowerW; // 最小発電出力   [W]
private double n_genePowerW;  // 定格発電出力   [W]

private double n_geneEffi;    // 定格発電効率
private double p75geneEffi;   // 負荷率75%時の発電効率
private double p50geneEffi;   // 負荷率50%時の発電効率

private double n_SrecvEffi;   // 定格排熱蒸気回収効率
private double p75SrecvEffi;  // 負荷率75%時の排熱蒸気回収効率
private double p50SrecvEffi;  // 負荷率50%時の排熱蒸気回収効率

private double n_HrecvEffi;   // 定格排熱温水回収効率
private double p75HrecvEffi;  // 負荷率75%時の排熱温水回収効率
private double p50HrecvEffi;  // 負荷率50%時の排熱温水回収効率
private double tMax_watOut;   //

private double n_DrecvEffi;   // 定格冷却水放熱率
private double p75DrecvEffi;  // 負荷率75%時の冷却水放熱率
private double p50DrecvEffi;  // 負荷率50%時の冷却水放熱率

private double n_m_watH;      // 定格排熱温水流量     [g/sec]
private double n_m_watCD;  // 定格冷却水流量     [g/sec]
private double n_subEleRate;  // 補機動力電力消費率
private boolean isHokiHirei = false;      // isHokiHirei補機は発電量に比例で計算する=true
private double n_subEleW;  // 運転時補機動力電力消費量   [W]
//private BestGeneratedPower generated;         // 発電
//private BestElectricity genPower;          // 発電量
//private BestFuel fuel;                     // 消費燃料
//private BestGas usedGass;                  // ガス消費量
private String moduleName2;                  // モジュール識別名
//
private int gen_phase;           //相数[-]
private double gen_voltage;      //電圧[V]
private double gen_frequency;    //周波数[Hz]
//private double gen_powerFactor;   //力率[-]

private boolean isGVisible = false;
private int  maxItemCount = 100;//最大同時表示ステップ数
private boolean isRecord   = false;

//内部変数
private StringBuffer message = new StringBuffer();

private String  name = null;  //機器名称

//private boolean operating;           // 運転状態
//

private double subEleW; // 補機動力電力消費量
private double demandEleW; // 発電量[W]
private double geneEffi;   // 発電効率[-]
private double fuelInputW; // 燃料消費量
private double qhrecvW; // 排熱温水回収量
private double qsrecvW; // 排熱蒸気回収量
private double qdrecvW; // 放熱量
private double qhrecvEffi; // 排熱温水回収効率
private double qsrecvEffi; // 排熱蒸気回収効率
private double qdrecvEffi; // 冷却水放熱率
private double allEffi; // 総合効率
private double geneLoadFactor;   // 発電負荷率
private double exhoustExitTemp;  // 排熱出口温度
//
private double copGEN;//COP発電[-]
private double copREC;//COP有効熱[-]
private double q;//発熱量[W]
private double qex;//放熱量[W]

private int swcIn;
```

```
//グラフ表示など
private GraphJFrameGasEngine20080909 gGE = null;


//仮設調整モード
//仮設調整モード2012
private boolean isAdjust2012  = false;//true="台数を調整する";//
private int numAdjustSteps;//"調整の計算ステップ数";

private double out_Gen_Adjust;//= "調整発電能力";

private LinkedList<Double> rGenList = null;//必要発電容量
private double max_rGenAdjust = 1.;
private double ave_rGenAdjust = 0;
private double out_rGenAdjust = 1.;//= "調整率冷房";

private double minGenePowerW_1;              // 最小発電出力   [W]
private double n_genePowerW_1;         // 定格発電出力   [W]
//private double GenerateEfficiencyRate;     // 定格発電効率
//private double a75;                      // 負荷率75%時の発電効率
//private double a50;                      // 負荷率50%時の発電効率
//private double StateExhoustRecoverRate;    // 定格排熱回収効率
//private double reco75;      // 負荷率75%時の排熱回収効率
//private double reco50;      // 負荷率50%時の排熱回収効率
private double n_m_watOut_1;           // 定格排熱温水流量     [g/sec]
//private double CoverActivationElecRate;    // 補機動力電力消費率
private double n_subEleW_1;   // 運転時補機動力電力消費量   [W]


@Override
public void setProfile(BestSpecs spec) {
   //specを受け取れていなかったら何もしない
   if(spec == null){
      return;
   }
   //getSpec()の戻り値が取得できていない場合何もしない
   Map<String, String> map = spec.getSpec();
   if (map == null){
      return;
   }

   //名称を取得
   this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

   //isRecordを取得
   this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

   // 定格発電出力を取得する
   this.n_genePowerW = spec.getSpecValue( this.SPEC_n_genePowerW, 350000 );

   // 最小発電出力を取得する
   this.minGenePowerW = spec.getSpecValue( this.SPEC_minGenePowerW, (this.n_genePowerW*0.5) );


   // 定格発電効率を取得する
   this.n_geneEffi = spec.getSpecValue( this.SPEC_n_geneEffi, 0.405 );

   // 負荷率75%時の発電効率を取得する
   this.p75geneEffi = spec.getSpecValue( this.SPEC_p75geneEffi, 0.391 );

   // 負荷率50%時の発電効率を取得する
   this.p50geneEffi = spec.getSpecValue( this.SPEC_p50geneEffi, 0.35 );


   // 定格排熱蒸気回収効率を取得する
   this.n_SrecvEffi = spec.getSpecValue( this.SPEC_n_SrecvEffi, 0.345 );

   // 負荷率75%時の排熱蒸気回収効率を取得する
   this.p75SrecvEffi = spec.getSpecValue( this.SPEC_p75SrecvEffi, 0.359 );

   // 負荷率50%時の排熱蒸気回収効率を取得する
```

```
this.p50SrecvEffi = spec.getSpecValue( this.SPEC_p50SrecvEffi, 0.4 );


// 定格排熱温水回収効率を取得する
this.n_HrecvEffi = spec.getSpecValue( this.SPEC_n_HrecvEffi, 0.345 );

// 負荷率75%時の排熱温水回収効率を取得する
this.p75HrecvEffi = spec.getSpecValue( this.SPEC_p75HrecvEffi, 0.359 );

// 負荷率50%時の排熱温水回収効率を取得する
this.p50HrecvEffi = spec.getSpecValue( this.SPEC_p50HrecvEffi, 0.4 );

//private final String SPEC_tMax_watOut    = "排熱温水出口水温上限値[℃]";
this.tMax_watOut = spec.getSpecValue( this.SPEC_tMax_watOut, 90 );

// 定格冷却水放熱率を取得する
this.n_DrecvEffi = spec.getSpecValue( this.SPEC_n_DrecvEffi, 0.345 );

// 負荷率75%時の冷却水放熱率を取得する
this.p75DrecvEffi = spec.getSpecValue( this.SPEC_p75DrecvEffi, 0.359 );

// 負荷率50%時の冷却水放熱率を取得する
this.p50DrecvEffi = spec.getSpecValue( this.SPEC_p50DrecvEffi, 0.4 );

// 出口流量を取得する
this.n_m_watH = spec.getSpecValue( this.SPEC_n_m_watH, 0. );

// 出口流量を取得する
this.n_m_watCD = spec.getSpecValue( this.SPEC_n_m_watCD, 0. );

// 補機動力電力消費率を取得する
this.n_subEleRate = spec.getSpecValue( this.SPEC_n_subEleRate, 0.05 );

// SPEC_isHokiHire
this.isHokiHirei = spec.getSpecValue( this.SPEC_isHokiHire, false );

// 発電諸元を取得する
this.gen_frequency = spec.getSpecValue( this.SPEC_gen_frequency, 50.);
this.gen_phase = spec.getSpecValue( this.SPEC_gen_phase, 3 );
this.gen_voltage = spec.getSpecValue( this.SPEC_gen_voltage, 200. );

//isGVisibleを取得
this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

// 最大同時表示ステップ数
this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

// 運転時補機動力電力消費量 = 定格発電出力 * 補機動力電力消費率
this.n_subEleW = this.n_genePowerW * this.n_subEleRate;


//仮設調整
//isAdjust2012 = "台数を調整する";//
this.isAdjust2012 = spec.getSpecValue( this.SPEC_isAdjust2012, false );

// 調整の計算ステップ数
this.numAdjustSteps = spec.getSpecValue( this.SPEC_NumAdjustSteps, 12 );

this.rGenList = new LinkedList<Double>();
for( int i=0; i<this.numAdjustSteps; i++ ){
   this.rGenList.add( 0. );
}

//
this.minGenePowerW_1 = this.minGenePowerW;              // 最小発電出力    [W]
this.n_genePowerW_1 = this.n_genePowerW;        // 定格発電出力    [W]
this.n_m_watOut_1 = this.n_m_watH;           // 定格排熱温水流量      [g/sec]
this.n_subEleW_1 = this.n_subEleW;  // 運転時補機動力電力消費量    [W]

if( this.isAdjust2012 ){
   if( this.n_genePowerW == 0 ){
      this.n_genePowerW_1 = 0.1;//[kW]発電機容量調整の初期値
```

```
            this.minGenePowerW_1 = 0.05;
            this.n_m_watOut_1 = 0.00204;
            this.n_subEleW_1 = this.n_genePowerW_1 * this.n_subEleRate;
            this.n_genePowerW = 0.1;//[kW]発電機容量調整の初期値
        }else{
            this.n_genePowerW = 0.1;//[kW]発電機容量調整の初期値
        }

        this.select_Gen();

    }

    this.initdata201503();
}


public void initialize(IBestStateMessage stateNodes,
        IBestControlMessage commandNodes,
        IBestRecordMessage recordNodes,
        IBestDomainSchedule schedule) {

    //状態ノードを受け取る
    super.sm = stateNodes;
    //制御ノードを受け取る
    super.cm = commandNodes;
    //記録ノードを受け取る
    super.rm = recordNodes;

    //ノードを受け取れてない時は何もしません 2007/08/21tuika
    if (null == super.sm) {
        return;
    }


    //valInDemmand
    this.valInDemandPower = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInDemandPower);

    //温水
    this.steOut = BestSteam.bindnode( super.transferMapState, super.sm, this.S_NODE_steOut);

    this.steOut.setPressure( 7845000.+BestSteam.SAP );
    //温水
    this.watInS = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInS );

    //温水
    this.watOut = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOut);
    //温水
    this.watIn = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watIn );

    //温水
    this.watOutCD = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutCD);
    //温水
    this.watInCD  = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInCD );

    //airOut
    this.airOut = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airOut );
    //airIn
    this.airIn = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airIn );

    //電力
    this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

    this.eleOut =  BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleOut );
    this.eleOut.setFrequency( this.gen_frequency);
    this.eleOut.setPhase( this.gen_phase );
    this.eleOut.setVoltage( this.gen_voltage );

    //燃料ガス
    this.gasIn = BestGas.bindnode( super.transferMapState, super.sm, this.S_NODE_gasIn );
```

```
        //グラフ表示の準備
        if( this.isGVisible ){
            gGE = new GraphJFrameGasEngine20080909( this.name, this.maxItemCount,  this.n_genePowerW * 1.5, this.n_m_watH *
1.5, this.n_genePowerW / this.n_geneEffi * 1.5 );
        }
    }


    private void adjustSpecs(){

    }

    public void outputs() {
        //接続が存在しない場合は何もしない
        if (super.sm == null || super.cm == null ) {
        //if (super.sm == null && super.cm == null && super.rm == null) {
            //System.out.println("GE no operation");
            if( this.isRecord ) {
                this.message.append( "(E)sm  cm接続なし");
            }
            return;
        }

        //入口側状態値
        this.watInS= (BestWater)super.sm.getState(super.getConnectionNode( this.S_NODE_watInS ));
        this.watIn = (BestWater)super.sm.getState(super.getConnectionNode( this.S_NODE_watIn ));
        this.airIn = (BestAir)super.sm.getState( super.getConnectionNode( this.S_NODE_airIn ));

        // 発電目標
        this.valInDemandPower = (BestValue)super.sm.getState(super.getConnectionNode(this.S_NODE_valInDemandPower));

        //System.out.println( "GASEngine demandPower="+this.demandPower );

        //
        if( this.isAdjust2012 ){
            //
            this.rGenList.removeLast();

            this.rGenList.addFirst( this.valInDemandPower.getValue() );
            //

            double sumGen = 0;
            for( int i=0; i<this.numAdjustSteps; i++ ){
                sumGen += this.rGenList.get( i );
            }
            if( this.isRecord ) {
                this.message.append( "(C)sumGen=" + AirFormat.df_0( sumGen ));
            }

            this.ave_rGenAdjust = sumGen / this.numAdjustSteps;
            if( this.isRecord ) {
                this.message.append( "(C)ave_rGenAdjust=" + AirFormat.df_0( this.ave_rGenAdjust ));
            }
            //
            if( this.ave_rGenAdjust > this.max_rGenAdjust ){
                if( this.isRecord ) {
                    this.message.append( "(C)最大発電容量="+ AirFormat.df_0( this.max_rGenAdjust )+"-->" );
                }

                this.max_rGenAdjust = this.ave_rGenAdjust;
                if( this.isRecord ) {
                    this.message.append( AirFormat.df_0( this.max_rGenAdjust ) );
                }

                this.n_genePowerW = this.max_rGenAdjust;

                this.out_Gen_Adjust = this.n_genePowerW;
                //発電機の再選定
                this.select_Gen();

            }
```

```
         //循環水は定格水量が流れているとする
         this.watIn.setFlowRate( this.n_m_watH );

      }


      //外部制御信号を取得
      this.swcIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE));

      if( true ){

         this.calc201503();

      }else{

         this.calc_0();
      }

      // 出力ノードに設定
      // 排熱出口
      super.sm.setState(super.getConnectionNode(this.S_NODE_watOut), this.watOut );
      // 発電量
      super.sm.setState(super.getConnectionNode(this.S_NODE_eleOut), this.eleOut );
      //消費電力
      super.sm.setState(super.getConnectionNode(this.S_NODE_eleIn ), this.eleIn  );
      // 燃料消費量
      super.sm.setState(super.getConnectionNode(this.S_NODE_gasIn), this.gasIn );
      // 蒸気
      super.sm.setState(super.getConnectionNode(this.S_NODE_steOut), this.steOut );

      // 9) 記録ノードに計算結果を設定します

      //グラフデータ追加
      if( this.isGVisible ){
         gGE.addData( BestTimeManager.getDateWeatherTime(),
               this.watIn, this.watOut,
               this.airIn, this.airOut,
               this.eleOut, this.gasIn,
               this.copGEN, this.q, this.qex, this.copREC );
      }

      //記録ノード
      if( this.isRecord  && super.rm != null ){
         this.record();
      }

      //
      message.setLength(0);
   }


   private void select_Gen(){
      //発電機の仕様を再選定調整する。
      double rate = this.n_genePowerW / this.n_genePowerW_1;

      this.minGenePowerW = this.minGenePowerW_1 * rate;              // 最小発電出力   [W]
      //this.StateGenereteAmount = this.StateGenereteAmount_1 * rate;         // 定格発電出力   [W]
      this.n_m_watH = this.n_m_watOut_1 * rate;          // 定格排熱温水流量     [g/sec]
      this.n_subEleW = this.n_subEleW_1 * rate; // 運転時補機動力電力消費量   [W]
   }

   /**
    * 記録
    */
   private void record(){
      //System.out.println("GE set Record");
      if( CheckPrintModule.isPrintMessage ){
         super.rm.setRecord(super.getConnectionNode( this.RO_NODE), this.RECORD_message, this.name,
this.message.toString());
      }

      //if( CheckPrintModule.isPrintEnergy ){
```

```
      //}

      //if( CheckPrintModule.isPrintLoad ){
      //}

      if( CheckPrintModule.isPrintStateOut ){
         // 排熱温水出口温度
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_t_watOut, this.name,
AirFormat.df_2( this.watOut.getTemp()));
         // 排熱温水流量
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_m_watOut, this.name,
AirFormat.df_2( this.watOut.getFlowRate()));
         // 冷却水出口温度
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_t_watOutCD, this.name,
AirFormat.df_2( this.watOutCD.getTemp()));
         // 冷却水流量
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_m_watOutCD, this.name,
AirFormat.df_2( this.watOutCD.getFlowRate()));
      }

      if( CheckPrintModule.isPrintStateMy ){
         // 補機動力電力消費量
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_subEleW, this.name,
AirFormat.df_2( this.subEleW));
         // 発電量
         //super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_demandEleW, this.name,
AirFormat.df_0( this.demandEleW));
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_demandEleW, this.name,
AirFormat.df_0( this.eleOut.getActivePower()));
         // 発電効率
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_geneEffi, this.name,
AirFormat.df_4( this.geneEffi));
         // 燃料消費量
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_fuelInputW, this.name,
AirFormat.df_2( this.fuelInputW));
         // 排熱温水回収量
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_qhrecv, this.name,
AirFormat.df_4( this.qhrecvW));
         // 排熱温水回収効率
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_qhrecvEffi, this.name,
AirFormat.df_4( this.qhrecvEffi));
         // 排熱蒸気回収量
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_qSrecv, this.name,
AirFormat.df_4( this.qsrecvW));
         // 排熱蒸気回収効率
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_qSrecvEffi, this.name,
AirFormat.df_4( this.qsrecvEffi));
         // 放熱量
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_qDrecv, this.name,
AirFormat.df_4( this.qdrecvW));
         // 放熱率
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_qDrecvEffi, this.name,
AirFormat.df_4( this.qdrecvEffi));
         // 負荷率
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_geneLoadFactor, this.name,
AirFormat.df_4( this.geneLoadFactor));
         // 運転状態
         //super.rm.setRecord(super.getConnectionNode(this.R1_NODE), this.DEBUG_ID7, this.name, this.operating);
         //private final String RECORD_allEffi = "GenGE_総合効率#-#効率";
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_allEffi, this.name,
AirFormat.df_4( this.allEffi));
      }

      if( CheckPrintModule.isPrintStateIn ){
         // 排熱温水戻り温度
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_t_watIn, this.name,
AirFormat.df_2( this.watIn.getTemp()));
         // 冷却水入口温度
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_t_watInCD, this.name,
AirFormat.df_2( this.watInCD.getTemp()));
         // 発電要求量
         super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_in_GenDemand, this.name,
```

```
AirFormat.df_0( this.valInDemandPower.getValue() ));//発電要求量[W]
      }

      if( CheckPrintModule.isPrintAdjust ){
         //
         if( this.isAdjust2012 ){
            //記録ノードに発電調整容量を設定
            super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_out_GenAdjust, this.name,
AirFormat.df_0( this.out_Gen_Adjust ));//発電機調整容量[W]
         }
      }

   }


   public void update() {

   }

   public Object viewInternal(TestCommand cmd) {
      List<Object>  result = new ArrayList<Object>();
      result.add(super.sm);                  // 状態ノード
      result.add(super.cm);                  // 制御ノード
      result.add(super.rm);                  // 記録ノード

      result.add(this.minGenePowerW);  // 最小発電出力
      result.add(this.n_genePowerW);   // 定格発電出力
      result.add(this.p75geneEffi); // 負荷率75%時の発電効率   [%]
      result.add(this.p50geneEffi); // 負荷率50%時の発電効率   [%]
      result.add(this.n_HrecvEffi); // 定格排熱回収効率
      result.add(this.p75HrecvEffi);   // 負荷率75%時の排熱回収効率[%]
      result.add(this.p50HrecvEffi);   // 負荷率50%時の排熱回収効率[%]
      result.add(this.n_m_watH); // 出口流量
      result.add(this.n_subEleRate);   // 補機動力電力消費率   [%]
      result.add(this.eleOut.getActivePower()); // 発電電力

      result.add(this.watOut);      // 排熱出口

      result.add(this.n_geneEffi);  // 定格発電効率
      return result;
   }


   private void calc_0(){


      if ( Airswc.isOFF( this.swcIn ) || this.watIn.getFlowRate() == 0 ) {
         // 停止
         //System.out.println( this.name + "(C)停止OFForMwatIn=0");
         if( this.isRecord ) {
            this.message.append( "(C)停止OFForMwatIn=0");
         }
         this.calc_Stop();

      }else if ( this.watIn.getTemp() > 95. ) {
         // 停止
         //System.out.println( this.name + "(C)入口温度>95℃→停止");
         if( this.isRecord ) {
            this.message.append( "(C)入口温度>95℃→停止");
         }
         this.calc_Stop();

      }else if ( this.valInDemandPower.getValue() < this.minGenePowerW ) {
         // 停止
         //System.out.println( this.name + "(C)入口温度>95℃→停止");
         if( this.isRecord ) {
            this.message.append( "(C)デマンド電力 < 最小発電電力→停止");
         }
         this.calc_Stop();

      }else if ( Airswc.isON( this.swcIn ) ) {
         // 運転
```

```
            //System.out.println( this.name + "(C)運転");
            if( this.isRecord ) {
               this.message.append( "(C)運転");
            }
            this.calc_gene();

         } else {
            // その他
            System.out.println( this.name + "(C)その他");
            if( this.isRecord ) {
               this.message.append( "(C)その他");
            }
            this.demandEleW =0;
            this.geneEffi = 0.0;
            this.qhrecvEffi = 0.0;
            this.fuelInputW = 0.0;
            this.qhrecvW = 0.0;
            this.subEleW = 0.0;
            this.watOut.copyAllState( this.watIn );
            this.eleIn.setActivePower( 0.);
            this.eleIn.setReactivePower( 0. );
            this.eleOut.setActivePower( 0. );
            this.eleOut.setReactivePower( 0. );
            this.gasIn.setWatt( 0. );
         }

         //copGEN copREC
         if(  this.gasIn.getWatt() + this.eleIn.getActivePower() != 0 ){
            this.copGEN = this.eleOut.getActivePower() / ( this.gasIn.getWatt() + this.eleIn.getActivePower() );
            this.copREC = this.qhrecvEffi / ( this.gasIn.getWatt() + this.eleIn.getActivePower() );
         }else{
            this.copGEN = 0;
            this.copREC = 0;
         }

         //q
         this.q = this.qhrecvW;
         //qex
         this.qex = this.fuelInputW - this.qhrecvW;


         //チェック
         if( this.watOut.getTemp() - this.watIn.getTemp() > 100 ){
            if( this.isRecord ) {
               this.message.append( "■■出口温度異常■■TwatIn=" + this.watIn.getTemp() + " / TOut=" +
this.watOut.getTemp() );
            }
            //System.out.println( "TwatIn=" + this.watIn.getTemp() + " / TOut=" + this.watOut.getTemp() );
            //System.out.println( "■■GasEngine  出口温度異常■■");
         }


      }

      private void calc_gene(){
         // 2) 発電量を求めます
         //上限チェック
         this.demandEleW = this.valInDemandPower.getValue() > this.n_genePowerW ? this.n_genePowerW :
this.valInDemandPower.getValue();
         //下限チェック (追加20081023nino) 20090226訂正
         //ガスエンジンは下限チェックを行わない→止める
         //this.demandPower = this.valInDemandPower.getValue() < this.MinGenerate ? this.MinGenerate : this.demandPower;

         //System.out.println ( "DemmandPower = " + this.demandPower + "  /swcIn = "+ this.swcIn + "   /T_watIn =
"+this.watIn.getTemp() );

         // 3) 発電効率を求めます
         // 負荷率 = （発電量 / 定格発電出力 ）
         this.geneLoadFactor = this.demandEleW / this.n_genePowerW;

         //発電の負荷率は最大１とする
         //if( this.loadFactor > 1 ){
```

```
      // this.loadFactor = 1;
      //}

      // 発電効率 = (8 * 定格効率 - 16 * a75 + 8 * a50) * x^2 + ( -10 * 定格効率 + 24 * a75 -14 * a50 ) * x + (3 * 定格効
率 - 8 * a75 + 6 * a50 )
      this.geneEffi =
            (8.0 * this.n_geneEffi - 16.0 * p75geneEffi +  8.0 * p50geneEffi) * Math.pow(this.geneLoadFactor, 2.0)
         + (-10.0 * this.n_geneEffi + 24.0 * p75geneEffi - 14.0 * p50geneEffi) * this.geneLoadFactor
         + (  3.0 * this.n_geneEffi -  8.0 * p75geneEffi +  6.0 * p50geneEffi);
      // 4) 排熱回収効率を求めます
      // 排熱回収効率 = 定格発電効率 + 定格排熱回収効率 - 発電効率
      //this.exhoustRecoverRate = this.GenerateEfficiencyRate
      //            + this.StateExhoustRecoverRate
      //            - this.generateRate;
      this.qhrecvEffi =
           (  8.0 * this.n_HrecvEffi - 16.0 * p75HrecvEffi +  8.0 * p50HrecvEffi) * Math.pow(this.geneLoadFactor, 2.0)
         + (-10.0 * this.n_HrecvEffi + 24.0 * p75HrecvEffi - 14.0 * p50HrecvEffi) * this.geneLoadFactor
         + (  3.0 * this.n_HrecvEffi -  8.0 * p75HrecvEffi +  6.0 * p50HrecvEffi);

      // 5) 燃料消費量を求めます
      // 燃料消費量 = (発電量 / 発電効率 ) * (45 / 40.63)
      if (this.geneEffi != 0.0){
         this.fuelInputW = (this.demandEleW / this.geneEffi ) * (45.0 / 40.63);
      }else{
         this.fuelInputW = 0;
      }

      // 6) 排熱回収量を求めます
      // 排熱回収量 = (燃料消費量 * 排熱回収効率 ) * (40.63 / 45)
      if (this.fuelInputW != 0.0 && this.qhrecvEffi != 0.0)
         this.qhrecvW = (this.fuelInputW * this.qhrecvEffi ) * (40.63 / 45.0);

      // 7) 排熱出口温度を求めます
      double c = 4.186;          // 水の比熱 (8/15モジュール仕様書により4.186)

      // 排熱入口温度は既知なのでノードのインスタンスを確認します
      //this.watIn = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watIn));
      //if (null == this.watIn) {
      // this.watIn = new BestWater();
         //watIn.setTemp(80.0);
      //}
      //this.exhoustExitTemp = 0.0;
      if (this.qhrecvW != 0.0)
         //排熱出口温度 = 排熱入口温度 + 排熱回収量 / c * G
         this.exhoustExitTemp = watIn.getTemp() + ( this.qhrecvW / (c * this.watIn.getFlowRate()));

      // 8) 補機動力の電力消費量を求めます
      this.subEleW = this.n_subEleW;

      // 8) 出口熱媒体に諸元を設定します
      this.eleIn.setActivePower( this.subEleW );
      this.eleIn.setReactivePower( this.subEleW );
      this.eleOut.setActivePower( this.demandEleW );
      this.eleOut.setReactivePower( this.demandEleW );

      this.gasIn.setWatt( this.fuelInputW );

      this.watOut.setTemp(this.exhoustExitTemp);
      this.watOut.setFlowRate(this.watIn.getFlowRate() );

   }

   private void calc_Stop(){
      //停止時の処理
      this.demandEleW = 0.;
      this.geneLoadFactor = 0;
      this.geneEffi = 0.0;
      this.qhrecvEffi = 0.0;
      this.fuelInputW = 0.0;
      this.qhrecvW = 0.0;
      this.subEleW = 0.0;
      this.watOut.copyAllState( this.watIn );
```

```
    this.eleIn.setActivePower( 0.);
    this.eleIn.setReactivePower( 0. );
    this.eleOut.setActivePower( 0. );
    this.eleOut.setReactivePower( 0. );
    this.gasIn.setWatt( 0. );

}

boolean isEta   = false;
boolean isSigma = false;
boolean isDelta = false;
boolean isOmega = false;

double aEta = 0;
double bEta = 0;
double cEta = 0;

double aSigma = 0;
double bSigma = 0;
double cSigma = 0;

double aDelta = 0;
double bDelta = 0;
double cDelta = 0;

double aOmega = 0;
double bOmega = 0;
double cOmega = 0;


private void initdata201503(){
    //蒸気温水

    //（１）定位発熱量　qf_LHV　(kJ/Nm3)
    //  FType=13Aの時：　qf_LHV=46000(kJ/Nm3)
    //  その他ガス　未対応
    //（２）定格熱消費量　QF_N(kW)
    //  QF_N=(VF_N/3600)*qf_LHV
    //（３）定格発電効率　η100(-)
    //  η100=Wel_N/QF_N
    //（４）定格蒸気熱効率　σ100
    //  σ100=QS_N/QF_N
    //（５）定格温水熱効率　ω100
    //  ω100=Qh_N/QF_N
    //（６）定格冷却水放熱率　δ100
    //  δ100=Qd_N/QF_N
    //（７）最小発電負荷率
    //  Rel_min=Welmin/Wel_N
    //（８）補機動力の定格発電能力比
    //  RelAux=WAux_N/Wel_N

    //部分負荷特性
    //１燃料消費量の換算
    //  DC1=0のとき（部分負荷特性不明時）
    //  QF75=0.7911*(VF_N/3600)*qf_LHV  [kW]
    //  QF50=0.5838*(VF_N/3600)*qf_LHV  [kW]
    //
    //  DC1=1のとき（部分負荷特性入力時）
    //  QF75=(VF75/3600)*qf_LHV  [kW]
    //  QF50=(VF50/3600)*qf_LHV  [kW]
    //

    //発電効率特性
    //  DC1=0のとき（部分負荷特性不明時）
    //  Aη=-0.3173
    //  Bη=0.7632
    //  Cη=0.55415
    //
    //  DC1=1のとき（部分負荷特性入力時）
    //  η75=0.75*Wel_N/QF75
    //  Rη75=η75/η100
    //  η50=0.5*Wel_N/QF50
```

```
//  Rη50=η50/η100
//  Aη=-16*Rη75+8*Rη50+8
//  Bη=24*Rη75-14*Rη50-10
//  Cη=-8*Rη75+6*Rη50+3
//
if( isEta ){
   this.aEta = -16. * this.p75geneEffi + 8. * this.p50geneEffi + 8.;
   this.bEta =  24. * this.p75geneEffi -14. * this.p50geneEffi - 10.;
   this.cEta =  -8. * this.p75geneEffi + 6. * this.p50geneEffi + 3.;

}else{
   this.aEta = -0.3173;
   this.bEta = 0.7632;
   this.cEta = 0.55415;

}


//最大蒸気量特性
//  DC1=0のとき（部分負荷特性不明時）
//  Aσ=-0.0690
//  Bσ=0.03717
//  Cσ=1.0379
//
//  DC1=1のとき（部分負荷特性入力時）
//  σ75=QS75/QF75
//  Rσ75=σ75/σ100
//  σ50=QS50/QF50
//  Rσ50=σ50/σ100
//  Aσ=-16*Rσ75+8*Rσ50+8
//  Bσ=24*Rσ75-14*Rσ50-10
//  Cσ=-8*Rσ75+6*Rσ50+3
//
if( isSigma ){
   this.aSigma = -16. * this.p75SrecvEffi / this.n_SrecvEffi + 8. * this.p50SrecvEffi / this.n_SrecvEffi + 8.;
   this.bSigma =  24. * this.p75SrecvEffi / this.n_SrecvEffi -14. * this.p50SrecvEffi / this.n_SrecvEffi - 10.;
   this.cSigma =  -8. * this.p75SrecvEffi / this.n_SrecvEffi + 6. * this.p50SrecvEffi / this.n_SrecvEffi + 3.;

}else{
   this.aSigma = -0.0690;
   this.bSigma = 0.03717;
   this.cSigma = 1.0379;

}



//温水加熱量特性
//  DC1=0のとき（部分負荷特性不明時）
//  Aω=0.2616
//  Bω=-0.6266
//  Cω=1.36495
//
//  DC1=1のとき（部分負荷特性入力時）
//  ω75=Qh75/QF75
//  Rω75=ω75/ω100
//  ω50=Qh50/QF50
//  Rω50=ω50/ω100
//  Aω=-16*Rω75+8*Rω50+8
//  Bω=24*Rω75-14*Rω50-10
//  Cω=-8*Rω75+6*Rω50+3
//
if( isOmega ){
   this.aOmega = -16. * this.p75HrecvEffi / this.n_HrecvEffi + 8. * this.p50HrecvEffi / this.n_HrecvEffi + 8.;
   this.bOmega =  24. * this.p75HrecvEffi / this.n_HrecvEffi -14. * this.p50HrecvEffi / this.n_HrecvEffi - 10.;
   this.cOmega =  -8. * this.p75HrecvEffi / this.n_HrecvEffi + 6. * this.p50HrecvEffi / this.n_HrecvEffi + 3.;

}else{
   this.aOmega = 0.2616;
   this.bOmega = -0.6266;
   this.cOmega = 1.36495;
```

```
    }


    //冷却水放熱特性
    //  DC1=0のとき（部分負荷特性不明時）
    //  Aδ=-3.007
    //  Bδ=5.414
    //  Cδ=-1.406
    //
    //  DC1=1のとき（部分負荷特性入力時）
    //  δ75=Qd75/QF75
    //  Rδ75=δ75/δ100
    //  δ50=Qd50/QF50
    //  Rδ50=δ50/δ100
    //  Aδ=-16*Rδ75+8*Rδ50+8
    //  Bδ=24*Rδ75-14*Rδ50-10
    //  Cδ=-8*Rδ75+6*Rδ50+3
    //
    if( isOmega ){
       this.aDelta = -16. * this.p75DrecvEffi / this.n_DrecvEffi + 8. * this.p50DrecvEffi / this.n_DrecvEffi + 8.;
       this.bDelta =  24. * this.p75DrecvEffi / this.n_DrecvEffi -14. * this.p50DrecvEffi / this.n_DrecvEffi - 10.;
       this.cDelta =  -8. * this.p75DrecvEffi / this.n_DrecvEffi + 6. * this.p50DrecvEffi / this.n_DrecvEffi + 3.;

    }else{
       this.aDelta = -3.007;
       this.bDelta = 5.414;
       this.cDelta = -1.406;

    }


    //熱主運転時に蒸気負荷率Rst(-)を発電負荷率Rel(-)に換算する関数：
    //  関数形：  Rel=Ast*Rst*Rst＋Bst*Rst+Cst
    //  DC1=0のとき（部分負荷特性不明時）
    //  Rst75=0.7657
    //  Rst50=0.5166
    //
    //  DC1=1のとき（部分負荷特性入力時）
    //  Rst75=QS75/QS_N
    //  Rst50=QS50/QS_N
    //  Ast=(0.5*(1-Rst75)/(1-Rst50)-0.25)/(Rst75*Rst75-Rst75*(1+Rst50)+Rst50)
    //  Bst=0.5/(1-Rst50)-Ast*(1+Rst50)
    //  Cst=1-Ast-Bst
    //
}


int d1 = 0;//運転信号　停止=0、運転=1
int d2 = 0;
int d3 = 0;//自動発停　停止=０、運転=1
int d4 = 0;//冷却水系　異常=1、正常=0

int d6 = 1;

int dm1 = 0;


private void calc201503(){

    this.demandEleW = this.valInDemandPower.getValue();

    //②運転信号の処理
    //冷却水異常、容量制御下限未満では機器は停止する
    //d2：強制的に出力一定とする
    //d6：温水加熱の有無：なし=0,あり＝1
    //dm1=1：温水停止モードあり
    //
    //温水加熱の有無は温水停止モードありの時に温水流量または温度によって設定される
    //START

    if( Airswc.isON( this.swcIn )){
```

```
      d1 = 1;
    }else{
      d1 = 0;
    }
    d4=0;

    double rvh = 0;//温水流量率
    double vh = this.watIn.getFlowRate();//温水流量[g/s]
    double vh_n = this.n_m_watH;//定格温水流量[g/s]

    double rvd = 0;//冷却水流量率
    double vd = this.watInCD.getFlowRate();//冷却水流量[g/s]
    double vd_n = this.n_m_watCD;//定格冷却水流量[g/s]

    double tdin = this.watInCD.getTemp();//冷却水入口水温
    double thin = this.watIn.getTemp();//温水入口温度
    double thmax = 95.;//温水出口上限温度

    double rel = 0;//発電負荷率[-]
    double welreq = this.demandEleW;//発電要求[W]
    double welmin = this.minGenePowerW;//最小発電出力[W]
    //double relreq = 0;
    double wel_n = this.n_genePowerW;//定格発電出力[W]
    double wel = 0;//発電量[kW]


    double etael = 0;
    double qf = 0;
  // double vf = 0;
  // double qf_lhv = 0;
    double qs = 0;
    double tsw = this.watInS.getTemp();
    double sigmaSMax = 0;
    double qsMax = 0;
    double gsMax = 0;

  // double gsreq = 0;
    double gs = 0;
    double qskW = 0;
    double qhreq = 0;
    double thMax = this.tMax_watOut;

    double omegah = 0;
    double qhmax = 0;//最大温水加熱量[kW]
    double qdmin = 0;//最小冷却水放熱量
    double qh = 0;
    double thout = 0;//温水出口温度
    double qd = 0;
    double tdout = 0;//冷却水出口水温

    double waux = 0;
    double relaux = this.n_subEleRate;//補機電力率
    double eff = 0;

    if( d1==1 ){
      //運転中
      if( this.isRecord ){
        this.message.append( "(C)swcIn運転" );
      }

      //発電負荷率のチェック
      if( d2 == 1 ){
        //最大出力一定
        rel = 1;
      }else{
        if( welreq < welmin ){
          if( this.isRecord ){
            this.message.append( "(C)発電負荷率"+AirFormat.df_0(welreq)+"[W]が下限"+welmin+"未満で停止" );
          }
          stop();
          return;
        }
```

```
        double relreq = welreq / wel_n;

        if( relreq < 1 ){
           //容量制御
           rel = relreq;

        }else{
           //最大出力
           rel = 1;
        }
     }

     //温水・冷却水のチェック
     rvh = vh / vh_n;//温水流量率

     if( vd_n > 0 ){//20161208
        rvd = vd / vd_n;//冷却水流量率

        //冷却水流量チェック
        if( rvd < 0.8 * rel ){
           if( this.isRecord ){
              this.message.append( "(C)冷却水流量"+AirFormat.df_1(rvd)+"<"+AirFormat.df_1(0.8*rel)+"が不足で停止" );
           }
           d4 = 1;
           stop();
           return;
        }

        //冷却水温度チェック
        if( tdin > 34 ){
           if( this.isRecord ){
              this.message.append( "(C)冷却水入口温度"+AirFormat.df_1(tdin)+">34℃で停止" );
           }
           d4 = 1;
           stop();
           return;
        }
     }

     d4 = 0;

     if( rvh < 0.8 * rel || thin > thmax ){
        //温水流量不足、入口水温が高温
        if( dm1==1 ){
           //温水停止モード有
           d6 = 0;//温水加熱無し
        }else{
           if( this.isRecord ){
              this.message.append( "(C)温水流量"+AirFormat.df_1(rvh)+"<"+(AirFormat.df_1(0.8*rel))
                    +"が不足or入口水温"+AirFormat.df_1(thin)+">"+thmax+"最高出口水温で停止" );
           }
           stop();
           return;
        }

     }else{
        d6 = 1;//温水加熱あり
     }

     //蒸気発生量と温水・冷却水基準値の計算へ

  }else{
     //停止中
     if( this.isRecord ){
        this.message.append( "(C)swcIn停止" );
     }
     d4=0;
     stop();
     return;
  }
```

```
        //③蒸気発生量と温水・冷却水基準値の計算
        //蒸気発生量[kg/s]は、給水（還水）温度tsw[℃]から単位蒸気発生時の熱量qs[kJ/kg]を定めて算出するものとした。
        //ここで8[kgf/m2]（784.5kPa.G）の飽和蒸気エンタルピを2770.77[kJ/kg]とした。

        //発停状態
        d3 = 1;
        //発電量
        wel = wel_n / 1000. * rel;//[kW]

        //発電効率
        etael = this.n_geneEffi * ( this.aEta * rel * rel + this.bEta * rel + this.cEta );

        //燃料消費量
        if ( etael != 0.0){
           qf = wel / etael  * (45.0 / 40.63);//[kW]
           //qf = wel / etael;//[kW]
           // vf = ( qf / qf_lhv ) * 3600;//[Nm3/h]
        }

        //蒸気発生熱量
        qs = 2770.77 - 4.18605 * tsw;//[kJ/kg]

        //最大蒸気発生率
        sigmaSMax = this.n_SrecvEffi * ( this.aSigma * rel * rel + this.bSigma * rel + this.cSigma );


        //最大蒸気発生量
        //qsMax = qf * sigmaSMax;//[kW]
        qsMax = qf * sigmaSMax * (40.63 / 45.0);//[kW]20181025
        gsMax = qsMax / qs;//[kg/s]

    // if( gsreq <=0 ){
    //    gs = 0;
    //
    // }else if( gsreq < gsMax ){
    //    gs = gsreq;
    //
    // }else{
    //    gs = gsMax;
    // }
        gs = gsMax;

        qskW = gs * qs;//[kW] = [kg/s]*[kJ/kg]

        if( d6==1 ){
           //温水加熱
           //温水熱負荷
           qhreq = ( thMax - thin ) * ( 4.196 * ( vh / 60000 ) * 983.2 );
        }else{
           //温水停止
           qhreq = 0;

        }

        //温水加熱効率
        omegah = this.n_HrecvEffi * ( this.aOmega * rel * rel + this.bOmega * rel + this.cOmega );

        //最大温水加熱量
        //qhmax = qf * omegah;//[kW]
        qhmax = qf * omegah * (40.63 / 45.0);//[kW]20181025


        //this.message.append( "sigmaSMax="+AirFormat.df_3(sigmaSMax)+" omegah="+AirFormat.df_3(omegah)+ " qhmax=
"+AirFormat.df_0(qhmax) + " qhreq="+AirFormat.df_0(qhreq) );

        //④温水、冷却水出口温度と総合効率の計算
        //温水加熱量Qhは、エンジンの負荷率Relから決まる最大値Qhmax以下の範囲内で、
        //温水出口上限温度Thmaxから計算されたQhreqに追従する。
        //ここで余った熱（Qhmax-Qhreq）は冷却水放熱量に加算される

        if( qhreq <= 0 ){
```

```
            //温水加熱量qh
            qh = 0;
        }else if( qhreq < qhmax ){
            qh = qhreq;
        }else{
            qh = qhmax;

        }

        //温水出口温度
        //thout = thin + qh / ( 4.196 * ( vh / 60000 ) * 983.2 );
        thout = thin + qh * 1000 / ( 4.18605 * vh );

        //冷却水放熱量
        //最小冷却水放熱量
        //qdmin = qf * this.n_DrecvEffi * ( this.aDelta * rel * rel + this.bDelta * rel + this.cDelta );//[kW]
        qdmin = qf * this.n_DrecvEffi * ( this.aDelta * rel * rel + this.bDelta * rel + this.cDelta ) * (40.63 /
45.0);//[kW]20181025
        qd = qdmin + ( qhmax - qh );//[kW]
        if( vd_n > 0 ){//20161208

            //冷却水出口水温
            //tdout = tdin + qd / ( 4.178 * ( vd / 60000 ) * 995.7 );
            tdout = tdin + qd * 1000 / ( 4.18605 * vd );

            if( tdout > 40 ){
                //高温停止
                if( this.isRecord ){
                    this.message.append( "(C)冷却水出口水温"+AirFormat.df_1(tdout)+">40℃で停止" );
                }
                d4 = 1;
                stop();
                return;
            }
        }else{
            tdout = 0;
        }

        d4 = 0;

        //補機電力
        if( this.isHokiHirei ){
            waux = wel * 1000. * relaux;//[kW]->[W]
        }else{
            waux = wel_n * relaux;//[kW]->[W]
        }

        //総合効率
        eff = ( wel + qskW + qh ) / qf;//[-]

        //状態セット
        this.fuelInputW = qf * 1000.;

        this.watOut.setTemp( thout );
        this.watOut.setFlowRate( this.watIn.getFlowRate() );
        this.watOutCD.setTemp( tdout );
        this.watOutCD.setFlowRate( this.watInCD.getFlowRate() );//20160526
        this.subEleW = waux;

        this.qhrecvW = qh * 1000.;
        this.qsrecvW = qskW * 1000.;
        this.qdrecvW = qd * 1000.;

        this.geneEffi = etael;
        this.qhrecvEffi = omegah;
        this.qsrecvEffi = sigmaSMax;
        this.qdrecvEffi = this.qdrecvW / this.fuelInputW;

        this.geneLoadFactor = rel;
        this.allEffi = eff;

        //出口熱媒体に諸元を設定します
```

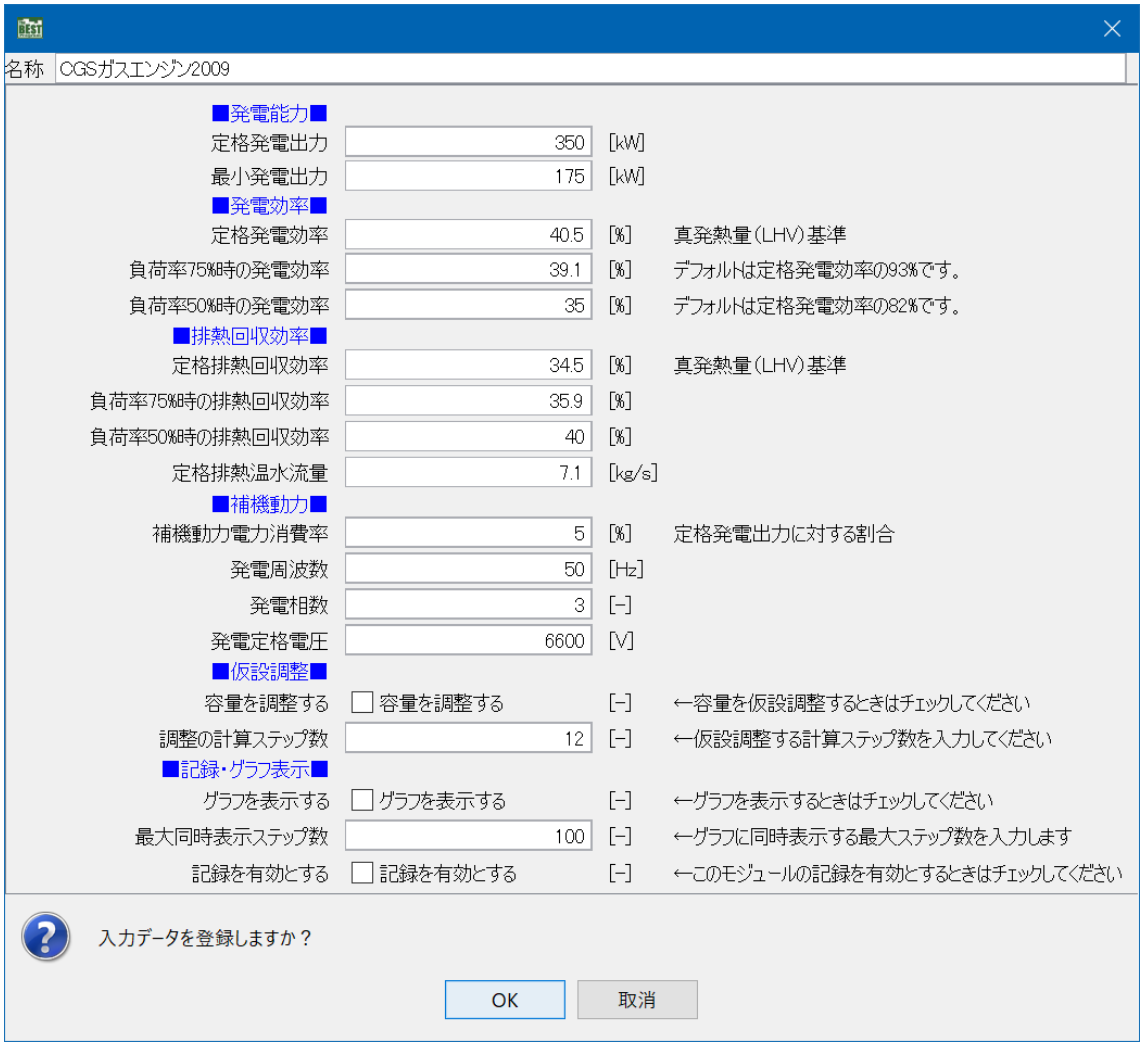
```
        this.eleIn.setActivePower( this.subEleW );
        this.eleIn.setReactivePower( this.subEleW );
        this.eleOut.setActivePower( wel * 1000. );
        this.eleOut.setReactivePower( wel * 1000. );

        this.gasIn.setWatt( qf * 1000. );

        this.steOut.setFlowRateSteam( gs * 1000. );

    }


    private void stop(){
        this.d3 = 0;
        this.d6 = 0;

        this.fuelInputW = 0;

        this.watOut.setTemp( this.watIn.getTemp() );
        this.watOut.setFlowRate( this.watIn.getFlowRate() );//20160526
        this.watOutCD.setTemp( this.watInCD.getTemp() );
        this.watOutCD.setFlowRate( this.watInCD.getFlowRate() );//20160526
        this.subEleW = 0;//waux = 0;

        this.qhrecvW = 0;//qh = 0;
        this.qsrecvW = 0;//qs = 0;
        this.qdrecvW =  0;//qd = 0;
        //vf = 0;
        //wel  = 0;

        this.geneEffi = 0;//eff = 0;
        this.qhrecvEffi = 0;
        this.qsrecvEffi = 0;
        this.qdrecvEffi = 0;

        this.geneLoadFactor = 0;
        this.allEffi = 0;

        this.demandEleW = 0.;
        this.eleIn.setActivePower( 0.);
        this.eleIn.setReactivePower( 0. );
        this.eleOut.setActivePower( 0. );
        this.eleOut.setReactivePower( 0. );
        this.gasIn.setWatt( 0. );
        this.steOut.setFlowRateSteam( 0. );

    }
}
```

「GasEngine20090101」（場所：設備 2015／コージェネ 2015／）

モジュール名	CGS ガスエンジン 2009
クラス	GasEngine20090101

（1）入力画面

・スペック



（2）モジュールの概要

CGS で使用する発電機の一つで、ガスを燃料とする。

このガスエンジンモジュールは、発電時の余剰熱は燃焼ガスと排温水にて放熱する。

このため、排温水回路には冷却塔などの冷却設備が必要である。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

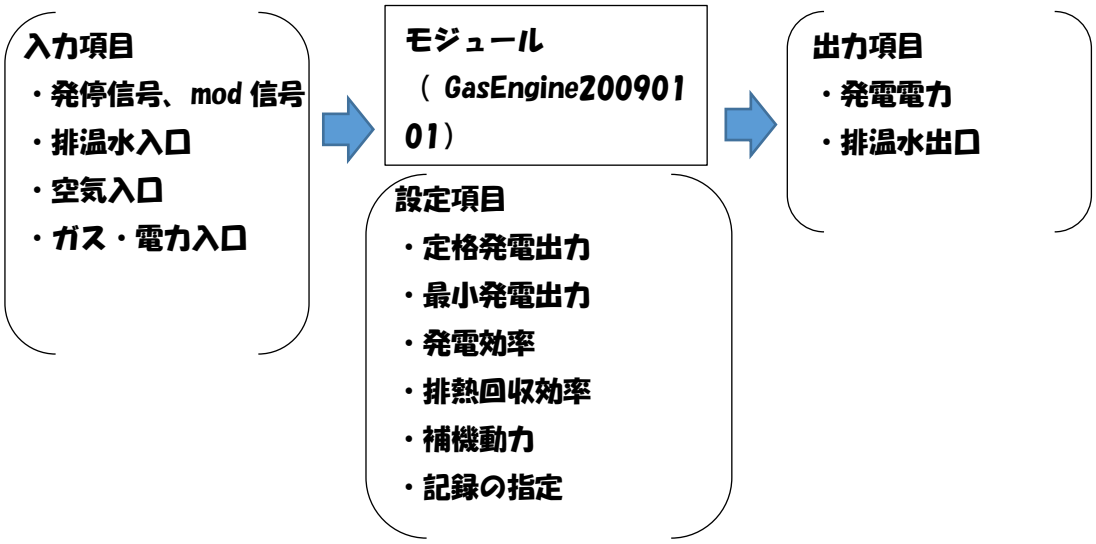


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図1における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String		ー	[-]	ー	ー		
1	■発電能力■					ー	ー		
2	定格発電出力	double	StateGenereteAmount	350	[kW]	ー	ー		
3	最小発電出力	double	MinGenerate	175	[kW]	ー	ー		
4	■発電効率■					ー	ー		
5	定格発電効率	double	GenerateEfficiencyRate	40.5	[%]	ー	ー		
6	負荷率 75%時の発電効率	double	a75	39.1	[%]	ー	ー		デフォルトは定格発電効率の 93%です。
7	負荷率 50%時の発電効率	double	a50	35	[%]	ー	ー		デフォルトは定格発電効率の 82%です。
8	■排熱回収効率■					ー	ー		
9	定格排熱回収効率	double	StateExhoustRecoverRate	34.5	[%]	ー	ー		
10	負荷率 75%時の排熱回収効率	double	reco75	35.9	[%]	ー	ー		デフォルトは定格排熱回収効率の　%です。
11	負荷率 50%時の排熱回収効率	double	reco50	40	[%]	ー	ー		デフォルトは定格排熱回収効率の　%です。
12	定格排熱温水流量	double	ExitFeedAmount	7.1	[kg/s]	ー	ー		
13	■補機動力■					ー	ー		
14	補機動力電力消費率	double	CoverActivationElecRate	5	[%]	ー	ー		
15	発電周波数	double	eleOut/frequency	50	[Hz]	ー	ー		
16	発電相数	int	eleOut/phase	3	[-]	ー	ー		
17	発電定格電圧	double	eleOut/Voltage	6600	[V]	ー	ー		

18	■仮設調整■					ー	ー		
19	容量を調整する	boolean	isAdjust2012	FALSE	[-]	ー	ー		←容量を仮設調整するときはチェックしてください
20	調整の計算ステップ数	int	numAdjustSteps	12	[-]	ー	1		←仮設調整する計算ステップ数を入力してください
21	■記録・グラフ表示■					ー	ー		
22	グラフを表示する	boolean	isGVisible	FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
23	最大同時表示ステップ数	int	maxItemCount	100	[-]	ー	ー		←グラフに同時表示する最大ステップ数を入力します
24	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録１	L20recOut1	recOut1	-	記録	メモリ	出口	
2	記録２	L21recOut2	recOut2	-	記録	メモリ	出口	
3	発停信号	L1_swcInCommand	swcInCommand	-	制御	OnOff 信号	入口	
4	発電要求値	L0_valInDemandPower	valInDemandPower	-	値	値	入口	
5	排温水出口	L0_watOut	watOut	-	状態	水	出口	
6	排温水入口	L0_watIn	watIn	-	状態	水	入口	
7	発電電力出口	L0_eleOut	eleOut	-	状態	電力	出口	
8	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	
9	ガス入口	L0_gasIn	gasIn	-	状態	ガス	入口	
10	空気入口	L0_airIn	airIn	-	状態	空気	入口	
11	空気出口	L0_airOut	airOut	-	状態	空気	出口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	GenGE_Message#-#-	メッセージ	—	メッセージ
2	GenGE_補機動力電力消費量#W#電力消費	補機動力の電力消費量	W	My
3	GenGE_発電量#W#発電電力	発電量	W	My
4	GenGE_発電効率#-#発電効率	発電効率（運転時）	—	My
5	GenGE_燃料消費量#W#燃料消費	燃料消費量（運転時）	W	My
6	GenGE_排熱回収量#W#排熱回収	排熱回収量（運転時）	W	My
7	GenGE_排熱回収効率#-#効率	排熱回収効率（運転時）	—	My
8	GenGE_負荷率#-#負荷率	発電の負荷率	—	My
9	GenGE_排熱出口温度#℃#温度	排温水の出口温度	℃	出口
10	GenGE_排熱戻り温度#℃#温度	排温水の入口温度	℃	入口
11	GenGE_排熱流量#g/s#質量流量	排温水の出入口流量	g/s	出口
12	GenGE_発電要求量#W#電力	要求発電量	W	入口
13	GenGE_調整発電容量#W#電力	容量自動調整時の調整発電容量	W	調整

（7）計算フロー・計算内容

.

（8）データ範囲と範囲外の取扱い

参考ソース

```
package jp.or.ibec.best.domain.sample.cogen;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestGas;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.cogen.AbstractPowerGenerator;
import jp.or.ibec.best.domain.sample.air.AirFormat;
import jp.or.ibec.best.domain.sample.air.Airswc;
import jp.or.ibec.best.domain.sample.air.CheckPrintModule;
import jp.or.ibec.best.domain.sample.air.GraphJFrameGasEngine20080909;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * ガス発電機
 * @author 浅野良吾
 * @author nakayama
 * @author HIROSHI NINOMIYA /20081010 /20090101
 */
public class GasEngine20090101 extends AbstractPowerGenerator {

   private final String moduleName = "(GasEngine20090101) ";

   // 状態ノード
   private final String S_NODE_valInDemandPower = "L0_valInDemandPower";     // 要求電力
   private final String S_NODE_watIn  = "L0_watIn";  // 排熱入口
   private final String S_NODE_watOut = "L0_watOut"; // 排熱出口
   private final String S_NODE_airIn  = "L0_airIn";  // 給気
   private final String S_NODE_airOut = "L0_airOut"; // 排気
   private final String S_NODE_eleIn  = "L0_eleIn";  // 消費電力
   private final String S_NODE_eleOut = "L0_eleOut"; // 発電電力
   private final String S_NODE_gasIn  = "L0_gasIn";  // 燃料

//private Double valInDemmandPower = null;
   private BestWater watIn  = null;
   private BestWater watOut = null;
   private BestAir airIn  = null;
   private BestAir airOut = null;
   private BestElectricity eleIn  = null;
   private BestElectricity eleOut = null;
   private BestGas gasIn = null;

   private BestValue valInDemandPower = null;
   // 制御ノード
   private final String C_NODE = "L1_swcInCommand";// 運転制御

   // 記録ノード
```

```
private final String RO_NODE = "L20recOut1";      // 必須記録ノード
private final String R1_NODE = "L21recOut2";      // オプション記録ノード

// 外部定義項目のキー
private final String SPEC_name   = "名称";        //外部定義項目
//private final String SPEC_ITEM_0 = "最大発電量";          // [kW/h]
private final String SPEC_ITEM_0 = "最小発電出力[W]";            // [W]
private final String SPEC_ITEM_1 = "定格発電出力[W]";            // [W]
private final String SPEC_ITEM_2 = "定格発電効率[-]";            //
private final String SPEC_ITEM_3 = "負荷率75%時の発電効率[-]";      //
private final String SPEC_ITEM_4 = "負荷率50%時の発電効率[-]";      //
private final String SPEC_ITEM_5 = "定格排熱回収効率[-]";         //
private final String SPEC_ITEM_6 = "負荷率75%時の排熱回収効率[-]";     //
private final String SPEC_ITEM_7 = "負荷率50%時の排熱回収効率[-]";     //
private final String SPEC_ITEM_8 = "定格排熱温水流量[g/s]";          // [g/sec]
private final String SPEC_ITEM_9 = "補機動力電力消費率[-]";       //
private final String SPEC_isHokiHire = "is補機発電量比例";        //
private final String SPEC_ITEM_10 = "発電周波数[Hz]";             // [Hz]
private final String SPEC_ITEM_11 = "発電相数[-]";
private final String SPEC_ITEM_12 = "発電定格電圧[V]";         // [V]
//private final String SPEC_ITEM_13 = "モジュール識別名";
//
private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
private final String SPEC_maxItemCount = "最大同時表示ステップ数";//最大同時表示ステップ数
private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

private final String SPEC_isAdjust2012   = "容量を調整する";
private final String SPEC_NumAdjustSteps = "調整の計算ステップ数[-]";

//記録項目
private final String RECORD_message  = "GenGE_Message#-#-" ;
private final String RECORD_ID1 = "GenGE_補機動力電力消費量#W#電力消費";
private final String RECORD_ID2 = "GenGE_発電量#W#発電電力";
private final String RECORD_ID3 = "GenGE_発電効率#-#発電効率";
private final String RECORD_ID4 = "GenGE_燃料消費量#W#燃料消費";
private final String RECORD_ID5 = "GenGE_排熱回収量#W#排熱回収";
private final String RECORD_ID6 = "GenGE_排熱回収効率#-#効率";

private final String DEBUG_ID1 = "GenGE_負荷率#-#負荷率";
private final String DEBUG_ID2 = "GenGE_排熱出口温度#℃#温度";
private final String DEBUG_ID3 = "GenGE_排熱戻り温度#℃#温度";
private final String DEBUG_ID6 = "GenGE_排熱流量#g/s#質量流量";
//private final String DEBUG_ID7 = "運転状態";            // [boolean]

private final String RECORD_in_GenDemand = "GenGE_発電要求量#W#電力";
private final String RECORD_out_GenAdjust = "GenGE_調整発電容量#W#電力";

// 外部定義項目
//private double MaxGenerate;             // 最大発電量   [kW]
private double MinGenerate;            // 最小発電出力 [W]
private double StateGenereteAmount;       // 定格発電出力 [W]
private double GenerateEfficiencyRate;       // 定格発電効率
private double a75;                  // 負荷率75%時の発電効率
private double a50;                  // 負荷率50%時の発電効率
private double StateExhoustRecoverRate;   // 定格排熱回収効率
private double reco75;    // 負荷率75%時の排熱回収効率
private double reco50;    // 負荷率50%時の排熱回収効率
private double ExitFeedAmount;            // 定格排熱温水流量     [g/sec]
private double CoverActivationElecRate;   // 補機動力電力消費率
private boolean isHokiHirei = false;   // isHokiHirei補機は発電量に比例で計算する=true
private double RunningCoverActivateElecAmount; // 運転時補機動力電力消費量  [W]
//private BestGeneratedPower generated;      // 発電
//private BestElectricity genPower;          // 発電量
```

```
//private BestFuel fuel;                  // 消費燃料
//private BestGas usedGass;               // ガス消費量
private String moduleName2;               // モジュール識別名
//
private boolean isGVisible = false;//このグラフを表示する=true
private int  maxItemCount = 100;//最大同時表示ステップ数
private boolean isRecord   = false;//記録を有効とする=true

//内部変数
private StringBuffer message = new StringBuffer();     //メッセージ
private String  name = null; //機器名称

//private boolean operating;         // 運転状態
//

private double coverActivateElecAmount;// 補機動力電力消費量
private double demandPower;  // 発電量
private double generateRate; // 発電効率
private double fuelConsumptionAmount;  // 燃料消費量
private double exhoustRecoverAmount; // 排熱回収量
private double exhoustRecoverRate;// 排熱回収効率
private double loadFactor;// 負荷率
private double exhoustExitTemp;// 排熱出口温度
//
private double copGEN;//COP発電[-]
private double copREC;//COP有効熱[-]
private double q;//発熱量[W]
private double qex;//放熱量[W]

private int swcIn;

//グラフ表示など
private GraphJFrameGasEngine20080909 gGE = null;


//仮設調整モード
//仮設調整モード2012
private boolean isAdjust2012  = false;//true="台数を調整する";//
private int numAdjustSteps;//"調整の計算ステップ数";

private double out_Gen_Adjust;//= "調整発電能力";

private LinkedList<Double> rGenList = null;//必要発電容量
private double max_rGenAdjust = 1.;
private double ave_rGenAdjust = 0;
private double out_rGenAdjust = 1.;//= "調整率冷房";

private double MinGenerate_1;             // 最小発電出力 [W]
private double StateGenereteAmount_1;        // 定格発電出力 [W]
//private double GenerateEfficiencyRate;     // 定格発電効率
//private double a75;                  // 負荷率75%時の発電効率
//private double a50;                  // 負荷率50%時の発電効率
//private double StateExhoustRecoverRate;    // 定格排熱回収効率
//private double reco75;     // 負荷率75%時の排熱回収効率
//private double reco50;     // 負荷率50%時の排熱回収効率
private double ExitFeedAmount_1;          // 定格排熱温水流量     [g/sec]
//private double CoverActivationElecRate;    // 補機動力電力消費率
private double RunningCoverActivateElecAmount_1;  // 運転時補機動力電力消費量  [W]

@Override
public void setProfile(BestSpecs spec) {
  //specを受け取れていなかったら何もしない
  if(spec == null){
```

```
    return;
}
//getSpec()の戻り値が取得できていない場合何もしない
Map<String, String> map = spec.getSpec();
if (map == null){
    return;
}

//名称を取得
if(null != map.get(this.SPEC_name)) {
    this.name = (String)map.get(this.SPEC_name);
}else{
    System.out.println(this.moduleName + "(W)SPEC_名称がありません");
}

// 最小発電出力を取得する
if (map.containsKey(this.SPEC_ITEM_0)){
    this.MinGenerate = Double.parseDouble(map.get(this.SPEC_ITEM_0));
}else{
    System.out.println( this.moduleName + "(W)なしSPEC_ITEM_0");
}

// 定格発電出力を取得する
if (map.containsKey(this.SPEC_ITEM_1)){
    this.StateGenereteAmount = Double.parseDouble(map.get(this.SPEC_ITEM_1));
}else{
    System.out.println( this.moduleName + "(W)なしSPEC_ITEM_1");
}

// 定格発電効率を取得する
if (map.containsKey(this.SPEC_ITEM_2)){
    this.GenerateEfficiencyRate = Double.parseDouble(map.get(this.SPEC_ITEM_2));
}else{
    System.out.println( this.moduleName + "(W)なしSPEC_ITEM_20");
}

// 負荷率75%時の発電効率を取得する
if (map.containsKey(this.SPEC_ITEM_3)){
    this.a75 = Double.parseDouble(map.get(this.SPEC_ITEM_3));
}else{
    System.out.println( this.moduleName + "(W)なしSPEC_ITEM_3");
}

// 負荷率50%時の発電効率を取得する
if (map.containsKey(this.SPEC_ITEM_4)){
    this.a50 = Double.parseDouble(map.get(this.SPEC_ITEM_4));
}else{
    System.out.println( this.moduleName + "(W)なしSPEC_ITEM_4");
}

// 定格排熱回収効率を取得する
if (map.containsKey(this.SPEC_ITEM_5)){
    this.StateExhoustRecoverRate = Double.parseDouble(map.get(this.SPEC_ITEM_5));
}else{
    System.out.println( this.moduleName + "(W)なしSPEC_ITEM_5");
}

// 負荷率75%時の排熱回収効率を取得する
if (map.containsKey(this.SPEC_ITEM_6)){
    this.reco75 = Double.parseDouble(map.get(this.SPEC_ITEM_6));
}else{
    System.out.println( this.moduleName + "(W)なしSPEC_ITEM_6");
}
```

```
// 負荷率50%時の排熱回収効率を取得する
if (map.containsKey(this.SPEC_ITEM_7)){
   this.reco50 = Double.parseDouble(map.get(this.SPEC_ITEM_7));
}else{
   System.out.println( this.moduleName + "(W)なしSPEC_ITEM_7");
}

// 出口流量を取得する
if (map.containsKey(this.SPEC_ITEM_8)){
   this.ExitFeedAmount = Double.parseDouble(map.get(this.SPEC_ITEM_8));
}else{
   System.out.println( this.moduleName + "(W)なしSPEC_ITEM_8");
}

// 補機動力電力消費率を取得する
if (map.containsKey(this.SPEC_ITEM_9)){
   this.CoverActivationElecRate = Double.parseDouble(map.get(this.SPEC_ITEM_9));
}else{
   System.out.println( this.moduleName + "(W)なしSPEC_ITEM_9");
}

// SPEC_isHokiHire
if (map.containsKey(this.SPEC_isHokiHire)){
   this.isHokiHirei = Boolean.parseBoolean( map.get(this.SPEC_isHokiHire));
}else{
   System.out.println( this.moduleName + "(W)なしSPEC_isHokiHire");
}

// 発電諸元を取得する
if (map.containsKey(this.SPEC_ITEM_10)) {
   this.eleOut = new BestElectricity();
   this.eleOut.setFrequency(Double.parseDouble(map.get(this.SPEC_ITEM_10)));
   if (map.containsKey(this.SPEC_ITEM_11))
      this.eleOut.setPhase(Integer.parseInt(map.get(this.SPEC_ITEM_11)));
   if (map.containsKey(this.SPEC_ITEM_12))
      this.eleOut.setVoltage(Double.parseDouble(map.get(this.SPEC_ITEM_12)));
   //generated = new BestGeneratedPower(this, this.genPower);
}

//isGVisibleを取得
if(null != map.get(this.SPEC_isGVisible)) {
   this.isGVisible = Boolean.parseBoolean( map.get(this.SPEC_isGVisible) );
}else{
   System.out.println(this.moduleName2 + "(W)グラフ表示指定がありません->=false");
   this.isGVisible = false;
}

// 最大同時表示ステップ数
if (map.containsKey(this.SPEC_maxItemCount)){
   this.maxItemCount = Integer.parseInt(map.get(this.SPEC_maxItemCount));
}else{
   this.maxItemCount = 100;
}

//isRecordを取得
if(null != map.get(this.SPEC_isRecord)) {
   this.isRecord = Boolean.parseBoolean( map.get(this.SPEC_isRecord) );
}else{
   System.out.println(this.moduleName2 + "(W)記録指定がありません->=false");
   this.isRecord = false;
}
```

```
      // 運転時補機動力電力消費量 = 定格発電出力 * 補機動力電力消費率
      this.RunningCoverActivateElecAmount = this.StateGenereteAmount * this.CoverActivationElecRate;


      //仮設調整
      //isAdjust2012 = "台数を調整する";//
      if(null != map.get(this.SPEC_isAdjust2012)) {
        this.isAdjust2012 = Boolean.parseBoolean( map.get(this.SPEC_isAdjust2012) );
      }else{
        System.out.println(this.moduleName + "(W)台数を調整する指定がありません->=false");
        this.isAdjust2012 = false;
      }

      // 調整の計算ステップ数
      if (map.containsKey(this.SPEC_NumAdjustSteps)){
        this.numAdjustSteps = Integer.parseInt(map.get(this.SPEC_NumAdjustSteps));
      }else{
        this.numAdjustSteps = 12;
      }

      this.rGenList = new LinkedList<Double>();
      for( int i=0; i<this.numAdjustSteps; i++ ){
        this.rGenList.add( 0. );
      }

      //
      this.MinGenerate_1 = this.MinGenerate;            // 最小発電出力 [W]
      this.StateGenereteAmount_1 = this.StateGenereteAmount;      // 定格発電出力 [W]
      this.ExitFeedAmount_1 = this.ExitFeedAmount;        // 定格排熱温水流量     [g/sec]
      this.RunningCoverActivateElecAmount_1 = this.RunningCoverActivateElecAmount;// 運転時補機動力電力消
費量 [W]

      if( this.isAdjust2012 ){
        if( this.StateGenereteAmount == 0 ){
          this.StateGenereteAmount_1 = 0.1;//[kW]発電機容量調整の初期値
          this.MinGenerate_1 = 0.05;
          this.ExitFeedAmount_1 = 0.00204;
          this.RunningCoverActivateElecAmount_1 = this.StateGenereteAmount_1 *
this.CoverActivationElecRate;
          this.StateGenereteAmount = 0.1;//[kW]発電機容量調整の初期値
        }else{
          this.StateGenereteAmount = 0.1;//[kW]発電機容量調整の初期値
        }

        this.select_Gen();

      }

    }


    public void initialize(IBestStateMessage stateNodes,
        IBestControlMessage commandNodes,
        IBestRecordMessage recordNodes,
        IBestDomainSchedule schedule) {

      //状態ノードを受け取る
      super.sm = stateNodes;
      //制御ノードを受け取る
      super.cm = commandNodes;
      //記録ノードを受け取る
      super.rm = recordNodes;
```

```
//ノードを受け取れてない時は何もしません 2007/08/21tuika
if (null == super.sm) {
   return;
}


//valInDemmand
if( super.sm.getState( super.getConnectionNode( this.S_NODE_valInDemandPower )) != null ){
   this.valInDemandPower
   = (BestValue) super.sm.getState( super.getConnectionNode(this.S_NODE_valInDemandPower ));
}else{
   this.valInDemandPower = new BestValue( );
   super.sm.setState( super.getConnectionNode( this.S_NODE_valInDemandPower ),
      this.valInDemandPower );
}

//冷温水
if( super.sm.getState( super.getConnectionNode( this.S_NODE_watOut )) != null ){
   this.watOut
   = (BestWater) super.sm.getState( super.getConnectionNode(this.S_NODE_watOut));
}else{
   this.watOut = new BestWater(); //冷温水
   super.sm.setState( super.getConnectionNode( this.S_NODE_watOut ),
      this.watOut );
}

//airOut
if( super.sm.getState( super.getConnectionNode( this.S_NODE_airOut )) != null ){
   this.airOut
   = (BestAir) super.sm.getState( super.getConnectionNode(this.S_NODE_airOut));
}else{
   this.airOut = new BestAir(); //airOut
   super.sm.setState( super.getConnectionNode( this.S_NODE_airOut ),
      this.airOut );
}

//電力
if( super.sm.getState( super.getConnectionNode( this.S_NODE_eleIn )) != null ){
   this.eleIn
   = (BestElectricity) super.sm.getState( super.getConnectionNode(this.S_NODE_eleIn));
}else{
   this.eleIn = new BestElectricity(); //電力
   super.sm.setState( super.getConnectionNode( this.S_NODE_eleIn ),
      this.eleIn );
}

//燃料ガス
if( super.sm.getState( super.getConnectionNode( this.S_NODE_gasIn )) != null ){
   this.gasIn
   = (BestGas) super.sm.getState( super.getConnectionNode(this.S_NODE_gasIn));
}else{
   this.gasIn = new BestGas(); //燃料ガス
   super.sm.setState( super.getConnectionNode( this.S_NODE_gasIn ),
      this.gasIn );
}

//接続ノード　入口
//入口冷温水
if( super.sm.getState( super.getConnectionNode( this.S_NODE_watIn)) != null ){
   this.watIn
   = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watIn)) ;
}else{
   this.watIn = new BestWater();
```

```
        super.sm.setState( super.getConnectionNode( this.S_NODE_watIn), this.watIn );
        System.out.println( this.moduleName + ">>Warningr<< watIn is null !!" );
        message.append( "(W)入口水の接続がない→作成");
      }

      //airIn
      if( super.sm.getState( super.getConnectionNode( this.S_NODE_airIn )) != null ){
        this.airIn
        = (BestAir) super.sm.getState( super.getConnectionNode(this.S_NODE_airIn));
      }else{
        this.airIn = new BestAir();  //airIn
        super.sm.setState( super.getConnectionNode( this.S_NODE_airIn ),
          this.airIn );
        System.out.println( this.moduleName + ">>Warningr<< airIn is null !!" );
        message.append( "(W)入口airの接続がない→作成");
      }


      //グラフ表示の準備
      if( this.isGVisible ){
        gGE = new GraphJFrameGasEngine20080909( this.name, this.maxItemCount,  this.StateGenereteAmount *
1.5, this.ExitFeedAmount * 1.5, this.StateGenereteAmount / this.GenerateEfficiencyRate * 1.5 );
      }
    }


    private void adjustSpecs(){

    }

    public void outputs() {
      //接続が存在しない場合は何もしない
      if (super.sm == null || super.cm == null ) {
      //if (super.sm == null && super.cm == null && super.rm == null) {
        //System.out.println("GE no operation");
        this.message.append( "(E)sm  cm接続なし");
        return;
      }

      //入口側状態値
      this.watIn = (BestWater)super.sm.getState(super.getConnectionNode( this.S_NODE_watIn ));
      this.airIn = (BestAir)super.sm.getState( super.getConnectionNode( this.S_NODE_airIn ));
      // 発電目標
      this.valInDemandPower =
(BestValue)super.sm.getState(super.getConnectionNode(this.S_NODE_valInDemandPower));

      //System.out.println( "GASEngine demandPower="+this.demandPower );

      //
      if( this.isAdjust2012 ){
        //
        this.rGenList.removeLast();

        this.rGenList.addFirst( this.valInDemandPower.getValue() );
        //

        double sumGen = 0;
        for( int i=0; i<this.numAdjustSteps; i++ ){
          sumGen += this.rGenList.get( i );
        }
        this.message.append( "(C)sumGen=" + AirFormat.df_0( sumGen ));

        this.ave_rGenAdjust = sumGen / this.numAdjustSteps;
```

```
    this.message.append( "(C)ave_rGenAdjust=" + AirFormat.df_0( this.ave_rGenAdjust ));
    //
    if( this.ave_rGenAdjust > this.max_rGenAdjust ){
      this.message.append( "(C)最大発電容量="+ AirFormat.df_0( this.max_rGenAdjust )+"-->" );

      this.max_rGenAdjust = this.ave_rGenAdjust;
      this.message.append( AirFormat.df_0( this.max_rGenAdjust ) );

      this.StateGenereteAmount = this.max_rGenAdjust;

      this.out_Gen_Adjust = this.StateGenereteAmount;
      //発電機の再選定
      this.select_Gen();

    }

    //循環水は定格水量が流れているとする
    this.watIn.setFlowRate( this.ExitFeedAmount );

  }


  //外部制御信号を取得
  this.swcIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE));


  if ( Airswc.isOFF( this.swcIn ) || this.watIn.getFlowRate() == 0 ) {
    // 停止
    //System.out.println( this.name + "(C)停止OFForMwatIn=0");
    this.message.append( "(C)停止OFForMwatIn=0");
    this.calc_Stop();

  }else if ( this.watIn.getTemp() > 95. ) {
    // 停止
    //System.out.println( this.name + "(C)入口温度>95℃→停止");
    this.message.append( "(C)入口温度>95℃→停止");
    this.calc_Stop();

  }else if ( this.valInDemandPower.getValue() < this.MinGenerate ) {
    // 停止
    //System.out.println( this.name + "(C)入口温度>95℃→停止");
    this.message.append( "(C)デマンド電力 < 最小発電電力→停止");
    this.calc_Stop();

  }else if ( Airswc.isON( this.swcIn ) ) {
    // 運転
    //System.out.println( this.name + "(C)運転");
    this.message.append( "(C)運転");
    this.calc_gene();

  } else {
    // その他
    System.out.println( this.name + "(C)その他");
    this.message.append( "(C)その他");
    this.demandPower =0;
    this.generateRate = 0.0;
    this.exhoustRecoverRate = 0.0;
    this.fuelConsumptionAmount = 0.0;
    this.exhoustRecoverAmount = 0.0;
    this.coverActivateElecAmount = 0.0;
    this.watOut.copyAllState( this.watIn );
    this.eleIn.setActivePower( 0.);
    this.eleIn.setReactivePower( 0. );
```

```
        this.eleOut.setActivePower( 0. );
        this.eleOut.setReactivePower( 0. );
        this.gasIn.setWatt( 0. );
      }

      //copGEN copREC
      if(  this.gasIn.getWatt() + this.eleIn.getActivePower() != 0 ){
        this.copGEN = this.eleOut.getActivePower() / ( this.gasIn.getWatt() +
this.eleIn.getActivePower() );
        this.copREC = this.exhoustRecoverRate / ( this.gasIn.getWatt() + this.eleIn.getActivePower() );
      }else{
        this.copGEN = 0;
        this.copREC = 0;
      }

      //q
      this.q = this.exhoustRecoverAmount;
      //qex
      this.qex = this.fuelConsumptionAmount - this.exhoustRecoverAmount;


      //チェック
      if( this.watOut.getTemp() - this.watIn.getTemp() > 100 ){
        this.message.append( "■■出口温度異常■■TwatIn=" + this.watIn.getTemp() + " / TOut=" +
this.watOut.getTemp() );
        //System.out.println( "TwatIn=" + this.watIn.getTemp() + " / TOut=" + this.watOut.getTemp() );
        //System.out.println( "■■GasEngine  出口温度異常■■");
      }

      // 出力ノードに設定
      // 排熱出口
      super.sm.setState(super.getConnectionNode(this.S_NODE_watOut), this.watOut );
      // 発電量
      super.sm.setState(super.getConnectionNode(this.S_NODE_eleOut), this.eleOut );
      //消費電力
      super.sm.setState(super.getConnectionNode(this.S_NODE_eleIn ), this.eleIn  );
      // 燃料消費量
      super.sm.setState(super.getConnectionNode(this.S_NODE_gasIn), this.gasIn );

      // 9) 記録ノードに計算結果を設定します

      //グラフデータ追加
      if( this.isGVisible ){
        gGE.addData( BestTimeManager.getDateWeatherTime(),
              this.watIn, this.watOut,
              this.airIn, this.airOut,
              this.eleOut, this.gasIn,
              this.copGEN, this.q, this.qex, this.copREC );
      }

      //記録ノード
      if( this.isRecord  && super.rm != null ){
        this.record();
      }

      //
      message.setLength(0);
    }


    private void select_Gen(){
      //発電機の仕様を再選定調整する。
      double rate = this.StateGenereteAmount / this.StateGenereteAmount_1;
```

```
    this.MinGenerate = this.MinGenerate_1 * rate;              // 最小発電出力 [W]
    //this.StateGenereteAmount = this.StateGenereteAmount_1 * rate;        // 定格発電出力 [W]
    this.ExitFeedAmount = this.ExitFeedAmount_1 * rate;          // 定格排熱温水流量     [g/sec]
    this.RunningCoverActivateElecAmount = this.RunningCoverActivateElecAmount_1 * rate; // 運転時補機動力
電力消費量 [W]
  }

  /**
   * 記録
   */
  private void record(){
    //System.out.println("GE set Record");
    if( CheckPrintModule.isPrintMessage ){
      super.rm.setRecord(super.getConnectionNode( this.RO_NODE), this.RECORD_message, this.name,
this.message.toString());
    }

    //if( CheckPrintModule.isPrintEnergy ){
    //}

    //if( CheckPrintModule.isPrintLoad ){
    //}

    if( CheckPrintModule.isPrintStateOut ){
      // 排熱出口温度
      super.rm.setRecord(super.getConnectionNode(this.R1_NODE), this.DEBUG_ID2, this.name,
AirFormat.df_2( this.exhoustExitTemp) );
      // 排熱流量
      //if (this.operating)
      // super.rm.setRecord(super.getConnectionNode(this.R1_NODE), this.DEBUG_ID6, this.name,
this.ExitFeedAmount);
      //else
        super.rm.setRecord(super.getConnectionNode(this.R1_NODE), this.DEBUG_ID6, this.name,
AirFormat.df_2( this.watOut.getFlowRate()));
    }

    if( CheckPrintModule.isPrintStateMy ){
      // 補機動力電力消費量
      super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_ID1, this.name,
AirFormat.df_2( this.coverActivateElecAmount));
      // 発電量
      super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_ID2, this.name,
AirFormat.df_0( this.demandPower));
      // 発電効率
      super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_ID3, this.name,
AirFormat.df_4( this.generateRate));
      // 燃料消費量
      super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_ID4, this.name,
AirFormat.df_2( this.fuelConsumptionAmount));
      // 排熱回収量
      super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_ID5, this.name,
AirFormat.df_4( this.exhoustRecoverAmount));
      // 排熱回収効率
      super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_ID6, this.name,
AirFormat.df_4( this.exhoustRecoverRate));
      // 負荷率
      super.rm.setRecord(super.getConnectionNode(this.R1_NODE), this.DEBUG_ID1, this.name,
AirFormat.df_4( this.loadFactor));
      // 運転状態
      //super.rm.setRecord(super.getConnectionNode(this.R1_NODE), this.DEBUG_ID7, this.name,
this.operating);
    }
```

```
      if( CheckPrintModule.isPrintStateIn ){
        // 排熱戻り温度
        super.rm.setRecord(super.getConnectionNode(this.R1_NODE), this.DEBUG_ID3, this.name,
AirFormat.df_2( this.watIn.getTemp()));
        // 発電要求量
        super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_in_GenDemand, this.name,
AirFormat.df_0( this.valInDemandPower.getValue() ));//発電要求量[W]
      }

      if( CheckPrintModule.isPrintAdjust ){
        //
        if( this.isAdjust2012 ){
          //記録ノードに発電調整容量を設定
          super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_out_GenAdjust, this.name,
AirFormat.df_0( this.out_Gen_Adjust ));//発電機調整容量[W]
        }
      }

    }


    public void update() {

    }

    public Object viewInternal(TestCommand cmd) {
      List<Object>  result = new ArrayList<Object>();
      result.add(super.sm);                 // 状態ノード
      result.add(super.cm);                 // 制御ノード
      result.add(super.rm);                 // 記録ノード

      result.add(this.MinGenerate);           // 最小発電出力
      result.add(this.StateGenereteAmount);     // 定格発電出力
      result.add(this.a75);                 // 負荷率75%時の発電効率  [%]
      result.add(this.a50);                 // 負荷率50%時の発電効率  [%]
      result.add(this.StateExhoustRecoverRate); // 定格排熱回収効率
      result.add(this.reco75);  // 負荷率75%時の排熱回収効率[%]
      result.add(this.reco50);  // 負荷率50%時の排熱回収効率[%]
      result.add(this.ExitFeedAmount);        // 出口流量
      result.add(this.CoverActivationElecRate); // 補機動力電力消費率  [%]
      result.add(this.eleOut.getActivePower()); // 発電電力

      result.add(this.watOut);              // 排熱出口

      result.add(this.GenerateEfficiencyRate);  // 定格発電効率
      return result;
    }

    private void calc_gene(){
      // 2) 発電量を求めます
      //上限チェック
      this.demandPower = this.valInDemandPower.getValue() > this.StateGenereteAmount ?
this.StateGenereteAmount : this.valInDemandPower.getValue();
      //下限チェック（追加20081023nino）20090226訂正
      //ガスエンジンは下限チェックを行わない→止める
      //this.demandPower = this.valInDemandPower.getValue() < this.MinGenerate ? this.MinGenerate :
this.demandPower;

      //System.out.println ( "DemmandPower = " + this.demandPower + "  /swcIn = "+ this.swcIn + "
/T_watIn = "+this.watIn.getTemp() );

      // 3) 発電効率を求めます
```

```
      // 負荷率 = （発電量 / 定格発電出力 ）
      this.loadFactor = this.demandPower / this.StateGenereteAmount;

      //発電の負荷率は最大１とする
      //if( this.loadFactor > 1 ){
      // this.loadFactor = 1;
      //}

      // 発電効率 = (8 * 定格効率 - 16 * a75 + 8 * a50) * x^2 + ( -10 * 定格効率 + 24 * a75 -14 * a50 ) * x
+ (3 * 定格効率 - 8 * a75 + 6 * a50 )
      this.generateRate =
           (8.0 * this.GenerateEfficiencyRate - 16.0 * a75 + 8.0 * a50) * Math.pow(this.loadFactor, 2.0)
          + (-10.0 * this.GenerateEfficiencyRate + 24.0 * a75 - 14.0 * a50) * this.loadFactor
          + (3.0 * this.GenerateEfficiencyRate - 8.0 * a75 + 6.0 * a50);
      // 4) 排熱回収効率を求めます
      // 排熱回収効率 = 定格発電効率 + 定格排熱回収効率 - 発電効率
      //this.exhoustRecoverRate = this.GenerateEfficiencyRate
      //           + this.StateExhoustRecoverRate
      //           - this.generateRate;
      this.exhoustRecoverRate =
          (  8.0 * this.StateExhoustRecoverRate - 16.0 * reco75 +  8.0 * reco50) *
Math.pow(this.loadFactor, 2.0)
        + (-10.0 * this.StateExhoustRecoverRate + 24.0 * reco75 - 14.0 * reco50) * this.loadFactor
        + (  3.0 * this.StateExhoustRecoverRate -  8.0 * reco75 +  6.0 * reco50);

      // 5) 燃料消費量を求めます
      // 燃料消費量 = (発電量 / 発電効率 ) * (45 / 40.63)
      if (this.generateRate != 0.0){
        this.fuelConsumptionAmount = (this.demandPower / this.generateRate ) * (45.0 / 40.63);
      }
      this.gasIn.setWatt( this.fuelConsumptionAmount );

      // 6) 排熱回収量を求めます
      // 排熱回収量 = (燃料消費量 * 排熱回収効率 ) * (40.63 / 45)
      if (this.fuelConsumptionAmount != 0.0 && this.exhoustRecoverRate != 0.0)
        this.exhoustRecoverAmount = (this.fuelConsumptionAmount * this.exhoustRecoverRate ) * (40.63 /
45.0);

      // 7) 排熱出口温度を求めます
      double c = 4.186;          // 水の比熱 (8/15モジュール仕様書により4.186)

      // 排熱入口温度は既知なのでノードのインスタンスを確認します
      //this.watIn = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watIn));
      //if (null == this.watIn) {
      // this.watIn = new BestWater();
        //watIn.setTemp(80.0);
      //}
      //this.exhoustExitTemp = 0.0;
      if (this.exhoustRecoverAmount != 0.0)
        //排熱出口温度 = 排熱入口温度 + 排熱回収量 / c * G
        this.exhoustExitTemp = watIn.getTemp() + ( this.exhoustRecoverAmount / (c *
this.watIn.getFlowRate()));

      // 8) 補機動力の電力消費量を求めます
      if( this.isHokiHirei ){
        this.coverActivateElecAmount = this.demandPower * this.CoverActivationElecRate;
      }else{
        this.coverActivateElecAmount = this.RunningCoverActivateElecAmount;
      }

      // 8) 出口熱媒体に諸元を設定します
      this.eleIn.setActivePower( this.coverActivateElecAmount );
      this.eleIn.setReactivePower( this.coverActivateElecAmount );
```

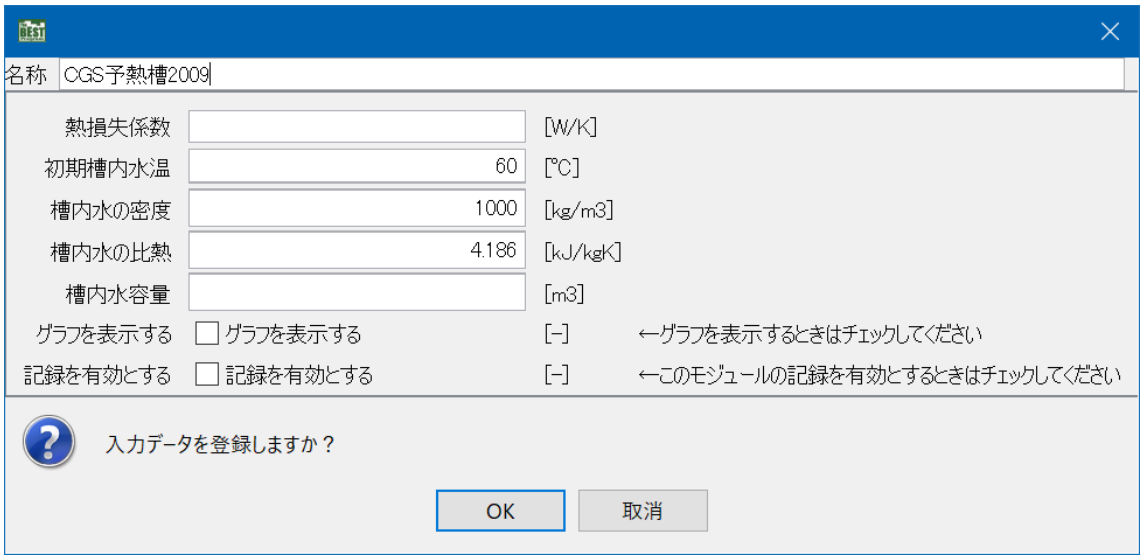
```
        this.eleOut.setActivePower( this.demandPower );
        this.eleOut.setReactivePower( this.demandPower );

        this.watOut.setTemp(this.exhoustExitTemp);
        this.watOut.setFlowRate(this.watIn.getFlowRate() );

    }

    private void calc_Stop(){
        //停止時の処理
        this.demandPower = 0.;
        this.loadFactor = 0;
        this.generateRate = 0.0;
        this.exhoustRecoverRate = 0.0;
        this.fuelConsumptionAmount = 0.0;
        this.exhoustRecoverAmount = 0.0;
        this.coverActivateElecAmount = 0.0;
        this.watOut.copyAllState( this.watIn );
        this.eleIn.setActivePower( 0.);
        this.eleIn.setReactivePower( 0. );
        this.eleOut.setActivePower( 0. );
        this.eleOut.setReactivePower( 0. );
        this.gasIn.setWatt( 0. );
        if( this.watIn.getFlowRate()> 0 ){
            this.exhoustExitTemp = this.watOut.getTemp();//20170307
        }

    }
}
```

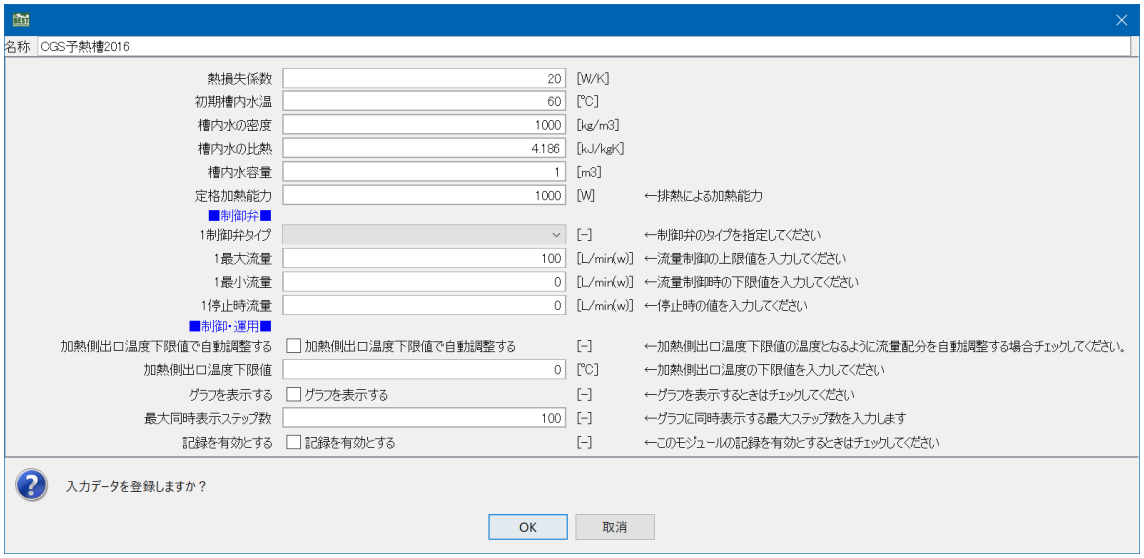
「CGS 予熱槽 2009」（場所：設備 2015／コージェネ 2015／）

モジュール名	CGS 予熱槽 2009
クラス	CGSHeatTank20090101

（1）入力画面

・スペック





（2）モジュールの概要

CGS で排熱温水を給湯利用する時の予熱槽モジュール。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	熱損失係数	double	ka		[W/K]	ー	ー		
2	初期槽内水温	double	sTemp	60	[℃]	ー	ー		
3	槽内水の密度	double	density	1000	[kg/m3]	ー	ー		
4	槽内水の比熱	double	specificHeat	4.186	[kJ/kgK]	ー	ー		
5	槽内水容量	double	capacity		[m3]	ー	ー		
6	グラフを表示する	boolean	isGVisible	FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
7	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2OrecOut	recOut	-	記録	メモリ	出口	
2	上位からの発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	
3	上位からの運転モード信号	L1_modIn	modIn	-	制御	制御モード	入口	
4	加熱側温水１入口	LO_watInH1	watInH1	-	状態	水	入口	
5	加熱側温水１出口	LO_watOutH1	watOutH1	-	状態	水	出口	
6	市水入口	LO_watInCW	watInCW	-	状態	水	入口	
7	給湯出口	LO_watOutHW	watOutHW	-	状態	水	出口	
8	空気（外気）入口	LO_airInOA	airInOA	-	状態	空気	入口	
9	水出口観察	LO_watOutAbs	watOutAbs	-	状態	水	出口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	CGSHTank_槽内水温#℃#-	槽内の水温	℃	My
2	CGSHTank_放熱量#W#熱量	放熱量	W	My
3	CGSHTank_加熱側入口温度#℃#-	加熱側の入口温度	℃	入口
4	CGSHTank_加熱側流量#g/s#-	加熱側流量	g/s	入口
5	CGSHTank_加熱側出口温度#℃#-	加熱側の出口温度	℃	出口
6	CGSHTank_加熱側熱量#W#熱量	加熱側の熱量	W	My
7	CGSHTank_給湯側入口温度#℃#-	給湯側の入口温度	℃	入口
8	CGSHTank_給湯側流量#g/s#-	給湯側の流量	g/s	入口
9	CGSHTank_給湯側出口温度#℃#-	給湯側の出口温度	℃	出口
10	CGSHTank_給湯側熱量#W#熱量	給湯側の熱量	W	My

（7）計算フロー・計算内容

.

（8）データ範囲と範囲外の取扱い

参考ソース

```
package jp.or.ibec.best.domain.sample.cogen;

import java.util.ArrayList;
import java.util.List;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.sample.air.AirFormat;
import jp.or.ibec.best.domain.sample.air.CheckPrintModule;
import jp.or.ibec.best.domain.sample.air.GraphJFrameHeatExchangerWaterWater20080909;
import jp.or.ibec.best.domain.share.AbstractBestTank;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * 予熱槽のサンプルです
 * @author T_Nakayama
 * @author h_okada
 *
 *  091007 tabata
 *  コージェネ排熱利用の予熱槽として修正
 */
public class CGSHeatTank20090101 extends AbstractBestTank implements IBestMetaModule {

   private String moduleName = "CGSHeatTank20090101";    //識別名

   //接続ノード

   private final String S_NODE_watInH1  = "LO_watInH1";     //加熱側入口
   private final String S_NODE_watOutH1 = "LO_watOutH1";    //加熱側出口
   private final String S_NODE_watInCW  = "LO_watInCW";     //給水
   private final String S_NODE_watOutHW = "LO_watOutHW";    //給湯負荷
   private final String RO_NODE = "L2OrecOut";
   //tabata追記→外気制御モジュールから得る補正外気のノード
   private final String S_NODE_airInOA = "LO_airInOA";      //放熱計算用
   private final String S_NODE_watOutAbs = "LO_watOutAbs";     //

   // 制御ノード
   private final String C_NODE_swcIn = "L1_swcIn";       // 運転制御
   private final String C_NODE_modIn = "L1_modIn";

   private final String SPEC_name    = "名称";          //外部定義項目
   private final String SPEC_KA      = "KA";         //[W/k]
   //private final String SPEC_ITEM_1 = "Tamb";       //[℃]
   private final String SPEC_Temp         = "初期槽内水温";//[℃]
   private final String SPEC_Density      = "槽内水の密度";//[g/m3]
   private final String SPEC_specificheat = "槽内水の比熱";//[J/gK]
   private final String SPEC_mCapacity    = "槽内水容量";//[m3]
   //
   private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

   private final String RECORD_T_watMy = "CGSHTank_槽内水温#℃#-";      //[℃]
   private final String RECORD_Losst_watMy = "CGSHTank_放熱量#W#熱量";        //[W]
   private final String RECORD_T_watInH1  = "CGSHTank_加熱側入口温度#℃#-";   //[℃]
   private final String RECORD_M_watInH1  = "CGSHTank_加熱側流量#g/s#-";
   private final String RECORD_T_watOutH1 = "CGSHTank_加熱側出口温度#℃#-";   //[℃]
   private final String RECORD_Watt_watInH1 = "CGSHTank_加熱側熱量#W#熱量";      //[W]
   private final String RECORD_T_watInCW  = "CGSHTank_給湯側入口温度#℃#-";   //[℃]
   private final String RECORD_M_watInCW  = "CGSHTank_給湯側流量#g/s#-";
   private final String RECORD_T_watOutHW = "CGSHTank_給湯側出口温度#℃#-";   //[℃]
   private final String RECORD_Watt_watOutHW = "CGSHTank_給湯側熱量#W#熱量";     //[W]
```

```
private StringBuffer message = new StringBuffer();    //メッセージ
private String  name = null;  //機器名称

private double ka;
private double tamb;
private double sTemp;               /* 初期槽内水温 */
private double density;             /* 槽内水の密度 */
private double specificHeat;        /* 槽内水の比熱 */
private double capacity;            /* 槽内水容量 */

private double oldTemp;          /* 前時刻槽内水温 */
//
private boolean isGVisible = false;//このグラフを表示する=true
private boolean isRecord   = false;//記録を有効とする=true
//
private int swcIn;
private int modIn;

private BestWater watInH1;
private BestWater watOutH1;
private BestWater watInCW;
private BestWater watOutHW;
private BestWater watOutAbs;
private BestAir airInOA;

//グラフ表示など
private GraphJFrameHeatExchangerWaterWater20080909 gGL = null;


@Override
public void setProfile(BestSpecs spec) {

   if (null == spec) {
      return;
   }
   java.util.Map<String, String> map = spec.getSpec();
   if (null == map) {
      return;
   }

   /* 外部定義項目を受け取ります */

   //名称を取得
   this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

   this.ka = spec.getSpecValue( this.SPEC_KA, 0.8);

   /*
   this.tamb = spec.getSpecValue( this.SPEC_ITEM_1, 15. );
   */

   this.sTemp = spec.getSpecValue( this.SPEC_Temp, 15. );

   this.density = spec.getSpecValue( this.SPEC_Density, 1000000. );

   this.specificHeat = spec.getSpecValue( this.SPEC_specificheat, 4.18605 );

   this.capacity = spec.getSpecValue( this.SPEC_mCapacity, 1000000. );

   //isGVisibleを取得
   this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

   //isRecordを取得
   this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

}


@Override
public void initialize(IBestStateMessage stateNodes,
```

```
            IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
            IBestDomainSchedule schedule) {

        /* ノードのインスタンスを取得します */
        super.cm = commandNodes ;     //制御ノードを取得
        super.sm = stateNodes;
        super.rm = recordNodes;

        //放熱計算用の外気
        this.airInOA = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airInOA );

        //温水１次側　入口
        this.watOutH1 = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutH1 );

        //温水２次側　入口
        this.watOutHW = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutHW );

        //温水１次側　出口
        this.watInH1 = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInH1 );

        //温水２次側　出口
        this.watInCW = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInCW );

        //温水２次側　入口
        this.watOutAbs = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutAbs );

        //グラフ表示の準備
        if( this.isGVisible ){
            gGL = new GraphJFrameHeatExchangerWaterWater20080909( this.name, 100,  0, 0 );
        }
        /* 加熱側出口の水熱媒をインスタンス化し、L0Oの流量を設定します */
//      double tempFlowRate = 0.0;

/*
        this.tempFlowRate = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watInH1 );
*/
        /* 給湯負荷の水熱媒をインスタンス化します */
/*
        this.transferMapState = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOutHW );
*/
        /* 初期槽内水温を前時刻槽内水温に設定します */
        this.oldTemp = this.sTemp;

    }

    @Override
    public void outputs() {

        if (  super.cm == null || null == super.sm || null == super.rm) {
            return;
        }

        //ノード読込み
        this.tamb = airInOA.getTempDB();
        this.watInH1  = (BestWater) super.sm.getState( super.getConnectionNode( this.S_NODE_watInH1 ));
        this.watInCW = (BestWater) super.sm.getState( super.getConnectionNode( this.S_NODE_watInCW ));

        this.swcIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ));
        this.modIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn ));


        double c = 4.186;    /* 係数 */
        /* 計算時間間隔を取得します */
        double dt = BestTimeManager.getCurrentInterval();  //秒

        if (0 == dt)
            return;

        /* 槽内水温を算出します */
        //M = 槽内水の密度 * 槽内水の比熱 * 槽内水容量
        //double m =   this.density * this.specificHeat * this.capacity;
```

```
        //tabata091007修正
        //UI画面に入力した数値は、m3→1000000gに自動変換されるため
        //ただ温水であるので、密度を変えたい要求に答えるため、密度入力は残すJ/K
        double m = (this.density/1000.0) * this.specificHeat * this.capacity;

        //槽内水温 = (M * Ts / dt + c * G1 * Tin1 + c * G2 * Tin2 + KA * Tamb) / (M / dt + c * G1  + c * G2 + KA)
        //double temp = ( m * this.oldTemp / dt
        //    + c * this.watInCW.getFlowRate() * this.watInCW.getTemp()
        //    + c * this.watInH1.getFlowRate() * this.watInH1.getTemp() + this.ka * this.tamb)
        //    / (m / dt + c * this.watInCW.getFlowRate() + c * this.watInH1.getFlowRate() + this.ka);

        double temp;
        //予熱槽よりも流出量が大きい場合
        if (m * this.oldTemp < ((c * (this.watInCW.getFlowRate() + this.watInH1.getFlowRate()) * this.oldTemp) * dt)) {
           //x:流入流量と予熱槽の比率、
           //watInTotalOverFlowRate：オーバーしている水量
           double x=((this.density/1000.0)*this.capacity/dt)/(this.watInCW.getFlowRate() + this.watInH1.getFlowRate());
           double watInTotalOverFlowRate=(this.watInCW.getFlowRate()+this.watInH1.getFlowRate())*(1-x);
           double mixTemp =
(this.watInCW.getFlowRate()*this.watInCW.getTemp()+this.watInH1.getFlowRate()*this.watInH1.getTemp())/(this.watInCW.getFl
owRate() + this.watInH1.getFlowRate());
           temp = this.oldTemp + (c * this.watInCW.getFlowRate() * this.watInCW.getTemp()
                 + c * this.watInH1.getFlowRate() * this.watInH1.getTemp() + this.ka * (this.tamb - this.oldTemp)
                 - (m * this.oldTemp)/dt -(c*watInTotalOverFlowRate*mixTemp)) * dt / m;
        } else {
           temp = this.oldTemp + (c * this.watInCW.getFlowRate() * this.watInCW.getTemp()
                 + c * this.watInH1.getFlowRate() * this.watInH1.getTemp() + this.ka * (this.tamb - this.oldTemp)
                 - c * (this.watInCW.getFlowRate() + this.watInH1.getFlowRate()) * this.oldTemp) * dt / m;
        }

        /* 放熱量を算出します */
        //放熱量 = KA * (Tamb - Ts)
        double adiation = this.ka * (this.tamb - temp);

        //System.out.print("adiation = " + adiation + "¥n");

        //System.out.println(adiation);  /* テスト用出力です */

        /* 流入水の熱量を算出します */
        //加熱側熱量 = c * G2 * (Tin2 - 槽内水温)
        double heatCapacityHeatingSide = c * this.watInH1.getFlowRate() * (this.watInH1.getTemp() - temp);
        this.watOutH1.setTemp(temp);

        //給湯負荷熱量 = c * G1 * (Tin1 - 槽内水温)
        double heatcapacityHotWaterSupplySide = c * this.watInCW.getFlowRate() * (this.watInCW.getTemp() - temp);
        this.watOutHW.setTemp(temp);

        //制御用の貯湯槽内の温度ABS
        this.watOutAbs.setTemp(temp);

        //tabata流量追加
        this.watOutH1.setFlowRate(this.watInH1.getFlowRate());
        this.watOutHW.setFlowRate(this.watInCW.getFlowRate());

        super.sm.setState(super.getConnectionNode(this.S_NODE_watOutH1), this.watOutH1);
        super.sm.setState(super.getConnectionNode(this.S_NODE_watOutHW), this.watOutHW);
        super.sm.setState(super.getConnectionNode(this.S_NODE_watOutAbs), this.watOutAbs);

        //グラフデータ追加
        if( this.isGVisible ){
           gGL.addData( BestTimeManager.getDateWeatherTime(),
                 this.watInH1, this.watOutH1,
                 this.watInCW, this.watOutHW,
                 heatcapacityHotWaterSupplySide );
        }

        //記録ノード
        if( this.isRecord  && super.rm != null ){
           this.record( temp, adiation, heatCapacityHeatingSide, heatcapacityHotWaterSupplySide);
        }

        /* 前時刻の槽内水温に、計算した槽内水温を設定します */
```

```
        this.oldTemp = temp;
        //
        message.setLength(0);


        //System.out.println(this.capacity + "¥t" );

    }

    /**
     * 記録
     */
    private void record( double temp, double adiation, double heatCapacityHeatingSide, double
heatcapacityHotWaterSupplySide ){
        /* 記録ノードに設定します */
        /* 出口温度を記録ノードに設定します */
        if( CheckPrintModule.isPrintMessage ){
        }

        //if( CheckPrintModule.isPrintEnergy ){
        //}

        //if( CheckPrintModule.isPrintLoad ){
        //}

        if( CheckPrintModule.isPrintStateOut ){
            super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_T_watOutH1, this.name,
AirFormat.df_2( this.watOutH1.getTemp()));
            super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_T_watOutHW, this.name,
AirFormat.df_2( this.watOutHW.getTemp()));
        }

        if( CheckPrintModule.isPrintStateMy ){
            super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_T_watMy, this.name,
AirFormat.df_2( temp));
            super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_Losst_watMy, this.name,
AirFormat.df_2( adiation));
            super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_Watt_watInH1, this.name,
AirFormat.df_2( heatCapacityHeatingSide));
            super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_Watt_watOutHW, this.name,
AirFormat.df_2( heatcapacityHotWaterSupplySide));
        }

        if( CheckPrintModule.isPrintStateIn ){
            super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_T_watInH1, this.name,
AirFormat.df_2( this.watInH1.getTemp()));
            super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_T_watInCW, this.name,
AirFormat.df_2( this.watInCW.getTemp()));
            super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_M_watInH1, this.name,
AirFormat.df_2( this.watInH1.getFlowRate()));
            super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_M_watInCW, this.name,
AirFormat.df_2( this.watInCW.getFlowRate()));
        }


    }

    @Override
    public void update() {

    }

    public Object viewInternal(TestCommand cmd) {
        List<Object>  result = new ArrayList<Object>();

        result.add(super.sm);               // 状態ノード
        result.add(super.rm);               // 記録ノード

        result.add(this.ka);              // KA
        result.add(this.tamb);              // Tamb
        result.add(this.sTemp);             // 初期槽内水温
        result.add(this.density);           // 槽内水の密度
```

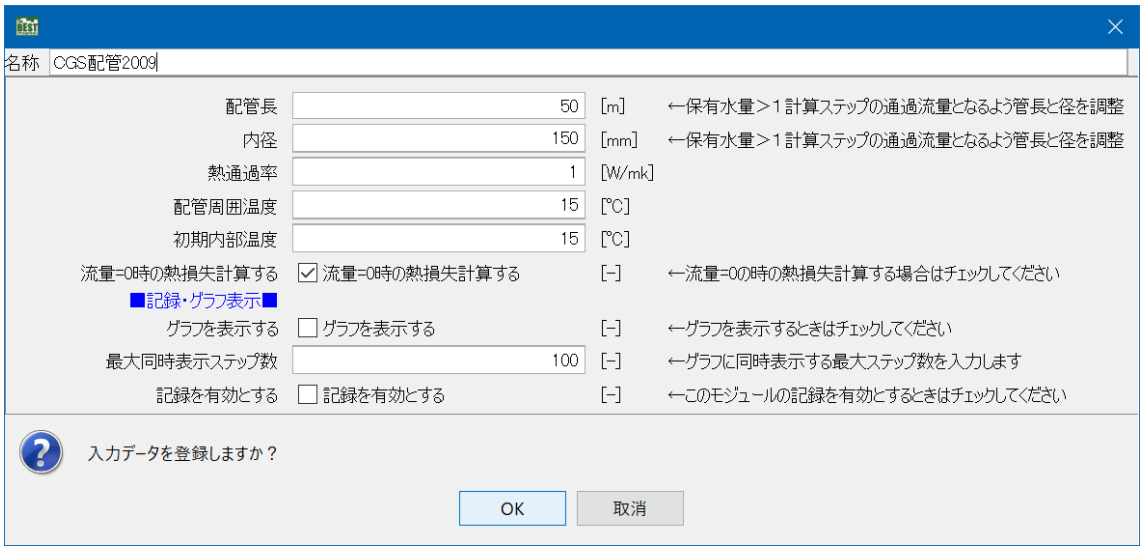
```
        result.add(this.specificHeat);          // 槽内水の比熱
        result.add(this.capacity);          // 槽内水容量

        result.add(this.oldTemp);           // 前時間の槽内水温
        result.add(this.swcIn);
        result.add(this.modIn);

        return result;
    }

}
```

「CGS 配管 2009」（場所：設備 2015／コージェネ 2015／）

モジュール名	CGS 配管 2009
クラス	CGSPipe20090101

（1）入力画面

・スペック



（2）モジュールの概要

配管からの熱損失を計算モジュール。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	配管長	double	PipeLength	10	[m]	ー	ー		←保有水量＞１計算ステップの通過流量となるよう管長と径を調整
2	内径	double	BoreDiameter	100	[mm]	ー	ー		←保有水量＞１計算ステップの通過流量となるよう管長と径を調整
3	熱通過率	double	kp	1	[W/mk]	ー	ー		
4	配管周囲温度	double	EnvironmentTemp	15	[℃]	ー	ー		
5	初期内部温度	double	InitInnerTemp	15	[℃]	ー	ー		
6	流量=0時の熱損失計算する	boolean	isCalcLossM0	TRUE	[-]	ー	ー		←流量=0 の時の熱損失計算する場合はチェックしてください
7	■記録・グラフ表示■					ー	ー		
8	グラフを表示する	boolean	isGVisible	FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
9	最大同時表示ステップ数	int	maxItemCount	100	[-]	ー	ー		←グラフに同時表示する最大ステップ数を入力します
10	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2OrecOut	recOut	-	記録	メモリ	出口	
2	水入口	LO_watIn	watIn	-	状態	水	入口	
3	水出口	LO_watOut	watOut	-	状態	水	出口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	CGSpipe_Message#-#-	メッセージ	—	
2	CGSpipe_入口温度#℃#温度	配管入口温度	℃	
3	CGSpipe_出口温度#℃#温度	配管出口温度	℃	
4	CGSpipe_入口質量流量#g/s#質量流量	配管入口流量	g/s	
5	CGSpipe_熱損失#W#熱量	熱損失	W	
6	CGSpipe_損失熱量#J#熱量	損失熱量	J	
7	CGSpipe_累積損失熱量#J#熱量	累積損失熱量	J	
8	CGSpipe_周囲温度#℃#温度	周囲温度	℃	
9	CGSpipe_出入口熱量差#W#熱量	配管出入口温度差	W	

（7）計算フロー・計算内容

.

（8）データ範囲と範囲外の取扱い

参考ソース

```
package jp.or.ibec.best.domain.sample.cogen;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.sample.air.CheckPrintModule;
import jp.or.ibec.best.domain.sample.air.GraphCommonData2015;
import jp.or.ibec.best.domain.sample.air.GraphCommonJFrame2015;
import jp.or.ibec.best.domain.share.AbstractBestPipe;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.message.BestEngineMessageHandler;
import jp.or.ibec.best.message.BestEngineMessageParam;
import jp.or.ibec.best.message.constants.SystemMessageConstants;

/**
 * コージェネレーションシステムのパイプのサンプルです
 * 空調や衛生のパイプモデルとは異なります
 * @author HIROSHI NINOMIYA
 *
 * 20131107 配管int→double  不具合訂正
 * 20160303  出口水温、処理熱量修正iida
 * 20171010 流量＝０の時の熱損失を計算しない機能を追加
 */
public class CGSPipe20090101 extends AbstractBestPipe implements
      IBestMetaModule {

   private String moduleName = "CGSPipe20090101";     //識別名

   // 状態
   private final String S_NODE_watIn  = "L0_watIn";   // 入口ノード L00
   private final String S_NODE_watOut = "L0_watOut";  // 出口ノード L01
   //記録ノード
   private final String R_NODE  = "L2OrecOut";

   // Mapのキー
   private final String SPEC_name   = "名称";         //外部定義項目
   private final String SPEC_length     = "配管長[m]";
   private final String SPEC_diameterIn = "内径[m]";
   private final String SPEC_kp         = "熱通過率[W/mk]";
   //private String SPEC_ITEM_2 = "計算時間間隔";
   private final String SPEC_T_environment = "配管周囲温度[℃]";
   private final String SPEC_T_initial     = "初期内部温度[℃]";
   private final String SPEC_isCalcLossMO  = "isCalcLossMO[-]";//20171010
   //
   private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
   private final String SPEC_maxItemCount = "最大同時表示ステップ数";//最大同時表示ステップ数
   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする
   //記録
   private final String RECORD_message  = "CGSpipe_Message#-#-" ;
   private final String RECORD_T_watIn  = "CGSpipe_入口温度#℃#温度";
   private final String RECORD_T_watOut = "CGSpipe_出口温度#℃#温度";
   private final String RECORD_M_watIn  = "CGSpipe_入口質量流量#g/s#質量流量";
   private final String RECORD_HeatLoss = "CGSpipe_熱損失#W#熱量";
   private final String RECORD_JHLoss   = "CGSpipe_損失熱量#J#熱量";
   private final String RECORD_JAccumulate = "CGSpipe_累積損失熱量#J#熱量";
   private final String RECORD_T_airObs = "CGSpipe_周囲温度#℃#温度";
   private final String RECORD_Watt_wat = "CGSpipe_出入口熱量差#W#熱量";

   // 外部定義項目
```

```
    private StringBuffer message = new StringBuffer();    //メッセージ
    private String  name = null;  //機器名称

    private double  PipeLength;         // 配管長       [m]
    private double  BoreDiameter;       // 内径         [mm]
    private int  TimeSpan;           // 計算時間間隔    [秒]
    private double EnvironmentTemp;     // 配管周囲温度    [℃]
    private double InitInnerTemp;    // 初期内部温度    [℃]
    private double kp;//熱通過率[W/mk]
    //
    private boolean isCalcLossM0 = true;// 流量０時の熱損失計算するしない[-]//20171010
    //
    private boolean isGVisible = false;//このグラフを表示する=true
    private int  maxItemCount = 100;//最大同時表示ステップ数
    private boolean isRecord   = false;//記録を有効とする=true

    private double Tp_Aster;         // 前回内部温度    [℃]

    private BestWater watIn;
    private BestWater watOut;
// private BestWater watMy;

    private double heatLoss;
    private double mPipe;//[J/K]
    private double watt_watInOut;//[W]

    private double jHeatLoss;
    private double jAccumulate;

    //グラフ表示など
    private GraphCommonJFrame2015 gGCJF = null;
    private GraphCommonData2015[] gData = null;


    @Override
    public void setProfile(BestSpecs spec) {
        //specを受け取れていなかったら何もしない
        if(null == spec){
            return;
        }
        //getSpec()の戻り値が取得できていない場合何もしない
        Map<String, String> map = spec.getSpec();
        if (null == map){
            return;
        }

        //名称を取得
        this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

        // 配管長[m]を取得する
        this.PipeLength = spec.getSpecValue( this.SPEC_length, 10. );

        // 内径[m]を取得する
        this.BoreDiameter = spec.getSpecValue( this.SPEC_diameterIn, 0.1 );

        // 熱通過率[W/mk]を取得する
        this.kp = spec.getSpecValue( this.SPEC_kp, 0.5 );

        // 計算時間間隔を取得する
        /*
        if (map.containsKey(this.SPEC_ITEM_2)){
            this.TimeSpan = Integer.parseInt(map.get(this.SPEC_ITEM_2));
        }
        */
        // 配管周囲温度を取得する
        this.EnvironmentTemp = spec.getSpecValue( this.SPEC_T_environment, 15. );

        // 初期内部温度を取得する
        this.InitInnerTemp = spec.getSpecValue( this.SPEC_T_initial, 15. );
```

```
        // 流量０時の熱損失計算するしない//20171010
        this.isCalcLossM0 = spec.getSpecValue( this.SPEC_isCalcLossM0, true );

        //isGVisibleを取得
        this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

        // 最大同時表示ステップ数
        this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

        this.mPipe = getMp(this.PipeLength, this.BoreDiameter);//[J/K]


        System.out.println( "PipeLength= " +PipeLength +"BoreDiameter= " +BoreDiameter);

    }

    public void initialize(IBestStateMessage stateNodes,
                           IBestControlMessage commandNodes,
                           IBestRecordMessage recordNodes,
                           IBestDomainSchedule schedule) {
        // 状態ノードがNULLならリターン
        if (stateNodes == null) {
            return ;
        }
        // 状態ノードのインスタンスを受け取ります
        super.sm = stateNodes;

        //記録ノードを取得
        super.rm = recordNodes;


        //水 入口
        this.watOut = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watOut );

        //初期水温の設定
        this.watOut.setTemp( this.InitInnerTemp );

        //水 出口
        this.watIn = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watIn );


        // 前回内部温度を初期内部温度で初期化します
        Tp_Aster = InitInnerTemp;

        //グラフ表示の準備
        if( this.isGVisible ){
            this.gData = new GraphCommonData2015[6];

            int i = 0;

            this.gData[i++] = new GraphCommonData2015( "T_watIn[℃]", this.maxItemCount, 0, 100, 0 );
            this.gData[i++] = new GraphCommonData2015( "T_watOut[℃]", this.maxItemCount, 0, 100, 0 );
            this.gData[i++] = new GraphCommonData2015( "T_environment[℃]", this.maxItemCount, 0, 100, 0 );

            this.gData[i++] = new GraphCommonData2015( "Q_heatloss[W]", this.maxItemCount, 0, 10000, 1 );
            this.gData[i++] = new GraphCommonData2015( "Q_watInOut[W]", this.maxItemCount, 0, 10000, 1 );

            this.gData[i++] = new GraphCommonData2015( "M_watIn[g/s]", this.maxItemCount, 0,
this.BoreDiameter*this.BoreDiameter/4*3.14*1000000*3, 2 );

            String[] yName = { "温度[℃]","熱量[W]","質量流量[g/s]" };

            gGCJF = new GraphCommonJFrame2015( this.name+"_CGSPipe", this.gData, yName );
        }

        this.jAccumulate = 0;
    }

    public void outputs() {
        // 状態ノードがNULLならリターン
        if(null == super.sm) {
```

```
        return;
    }

    this.watIn = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watIn));
    //現在の計算時間間隔を取得します
    this.TimeSpan = BestTimeManager.getCurrentInterval();

    if( this.isRecord ){
        this.message.append( "(C)T_watIn="+this.watIn.getTemp()+" /T_watOut="+this.watOut.getTemp());
        if( this.watIn.getTemp() > 100. ){
            this.message.append( "(W)watIn>100℃" );
        }
    }

    this.calc_HeatLoss();//20140819
    this.watt_watInOut = (this.watIn.getTemp()-this.watOut.getTemp())*this.watIn.getFlowRate()*4.18605;

    //グラフデータ追加
    if( this.isGVisible ){
        double[] gData = new double[6];

    /*
        this.gData[i++] = new GraphCommonData2015( "T_watIn[℃]", this.maxItemCount, 0, 100, 0 );
        this.gData[i++] = new GraphCommonData2015( "T_watOut[℃]", this.maxItemCount, 0, 100, 0 );
        this.gData[i++] = new GraphCommonData2015( "T_environment[℃]", this.maxItemCount, 0, 100, 0 );

        this.gData[i++] = new GraphCommonData2015( "Q_heatloss[W]", this.maxItemCount, 0, this.ua*100, 1 );
        this.gData[i++] = new GraphCommonData2015( "Q_watInOut[W]", this.maxItemCount, 0, this.ua*100, 1 );

        this.gData[i++] = new GraphCommonData2015( "M_watIn[g/s]", this.maxItemCount, 0, this.maxFlowRate1, 2 );
    */

        int i=0;
        gData[i++] = this.watIn.getTemp();
        gData[i++] = this.watOut.getTemp();
        gData[i++] = this.EnvironmentTemp;

        gData[i++] = this.heatLoss;
        gData[i++] = this.watt_watInOut;

        gData[i++] = this.watIn.getFlowRate();

        this.gGCJF.addData( BestTimeManager.getDateWeatherTime(), gData );
    }

    //記録ノード
    if( this.isRecord  && super.rm != null ){
        this.record();
    }

    //
    message.setLength(0);
}

/**
 * 記録
 */
private void record(){
    /* 記録ノードに設定します */
    /* 出口温度を記録ノードに設定します */
    //System.out.println( "*********(C)T_watIn="+this.watIn.getTemp()+" /T_watOut="+this.watOut.getTemp());
    if( CheckPrintModule.isPrintMessage ){
        super.rm.setRecord(super.getConnectionNode( this.R_NODE),
              this.RECORD_message, this.name, this.message.toString() );
    }

    //if( CheckPrintModule.isPrintEnergy ){
    //}

    //if( CheckPrintModule.isPrintLoad ){
    //}
```

```
        if( CheckPrintModule.isPrintStateOut ){
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_T_watOut, this.name, this.watOut.getTemp() );
        }

        if( CheckPrintModule.isPrintStateMy ){
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_HeatLoss, this.name, this.heatLoss );
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_JHLoss, this.name, this.jHeatLoss );
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_Watt_wat, this.name, this.watt_watInOut );
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_JAccumulate, this.name, this.jAccumulate );
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_T_airObs, this.name, this.EnvironmentTemp );
        }

        if( CheckPrintModule.isPrintStateIn ){
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_T_watIn, this.name, this.watIn.getTemp() );
            super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                  this.RECORD_M_watIn, this.name, this.watIn.getFlowRate() );
        }


    }

    public void calc_HeatLoss() {

        // 2) 出口温度を計算します
        double TpO = this.watOut.getTemp(); // 前回内部温度
        double dt = this.TimeSpan;       // 計算時間間隔
        double c = 4.186;             // 熱媒比熱
        double G = this.watIn.getFlowRate();// 流量
        double Tin = this.watIn.getTemp();  // 入口温度
        double Kp = this.kp;              // 熱線通過率 (0.5W/mKに固定)
        double lPipeW  = this.mPipe         /dt - c * G;; //パイプ内の残り熱量W

        //Tout = (Mp*TpO/dt + c*G*Tin + Kp*Lp*Tamb)/(Mp/dt + c*G + Kp*Lp)
/*      double Tout = ((this.mPipe * TpO / dt) + c * G * Tin + Kp * this.PipeLength * this.EnvironmentTemp)
                    / ((this.mPipe / dt) + c * G + Kp * this.PipeLength);*/

        if(lPipeW <= 0){
            lPipeW = 0;
            BestEngineMessageHandler.addMessage(
                  new BestEngineMessageParam(SystemMessageConstants.WARNINGPIPE, new String[]{this.name}));
        }
        if( !this.isCalcLossMO && G==0 ){//20171010
            //流量０時の熱損失計算しない場合
            Kp = 0;
        }
        double Tout = ((lPipeW * TpO ) + c * G * Tin + Kp * this.PipeLength * this.EnvironmentTemp)
              / (lPipeW + c * G + Kp * this.PipeLength);//160303iida  前のｻｲｸﾙの保有水が出ていった後の計算とする。

        this.heatLoss = Kp * this.PipeLength * (Tout - this.EnvironmentTemp);// 160310iida


        // 3) 出口熱媒に諸元を設定します
        this.watOut.setTemp(Tout);
        this.watOut.setFlowRate(G);

        if( this.watOut.getTemp() > 100. ){
            if( this.isRecord ) {
                this.message.append( "(W)watOut>100℃" );
            }
        }

        this.jHeatLoss = this.heatLoss * this.TimeSpan;
        this.jAccumulate += this.jHeatLoss;

        //出口温度を前回内部温度に設定します
```

```
    //this.Tp_Aster = Tout;
    super.sm.setState( super.getConnectionNode( this.S_NODE_watOut), this.watOut );
}


public void update() {

}


/*
 * getMp
 * 組み合わせ表による管内熱水容量の取得
 * PipeLength   : 管長
 * BoreDiameter : 内径
 * 戻り値: 組み合わせ表による管内熱水容量[J/K]
 */
private double getMp(int PipeLength, int BoreDiameter) {
    //熱容量=熱媒密度[g/m3]×比熱[J/g K]×熱媒容量[m3]
    //熱媒密度 = 1000000 [g/m3]
    //熱媒容量[m3] = π * ( 配管内径[m]/2 ) ^2 * 配管長[m]
    double result = 1000000.0 * 4.186 * Math.PI * Math.pow(BoreDiameter/1000. / 2., 2.0) * PipeLength;
    return result;
    /*
    double dAry[][] = {
        { 1683290,  25249357  },
        { 3787404,  5681054   },
        { 5155077,  77326156  },
        { 15149614, 227244214 }
    };
    int x = 0;
    int y = 0;

   // 内径を設定
      switch (BoreDiameter) {
      case 160:
       x = 0;
        break;
      case 240:
        x = 1;
        break;
    case 280:
        x = 2;
        break;
    case 480:
       x = 3;
       break;
    default:
       x = 999;
       break;
     }

      // 管長を設定
    switch (PipeLength) {
    case 20:
       y = 0;
       break;
    case 300:
       y = 1;
       break;
    default:
       y = 999;
       break;
    }

    if ((x == 999) | (y == 999)) {
       return 0;
    } else {
       return dAry[x][y];
    }
    */
}
```

```
/**
 * @param PipeLength 配管長[m]
 * @param BoreDiameter 内径[m]
 * @return [g/K]
 */
private double getMp(double PipeLength, double BoreDiameter) {
   //熱容量=熱媒密度[g/m3]×比熱[J/g K]×熱媒容量[m3]
   //熱媒密度 = 1000000 [g/m3]
   //熱媒容量[m3] = π * ( 配管内径[m]/2 ) ^2 * 配管長[m]
   double result = 1000000.0 * 4.186 * Math.PI * Math.pow(BoreDiameter / 2., 2.0) * PipeLength;
   return result;
   /*
   double dAry[][] = {
      { 1683290,  25249357  },
      { 3787404,  5681054   },
      { 5155077,  77326156  },
      { 15149614, 227244214 }
   };
   int x = 0;
   int y = 0;

  // 内径を設定
     switch (BoreDiameter) {
     case 160:
      x = 0;
       break;
     case 240:
       x = 1;
       break;
   case 280:
       x = 2;
       break;
   case 480:
      x = 3;
      break;
   default:
      x = 999;
      break;
    }

     // 管長を設定
   switch (PipeLength) {
   case 20:
      y = 0;
      break;
   case 300:
      y = 1;
      break;
   default:
      y = 999;
      break;
   }

   if ((x == 999) | (y == 999)) {
      return 0;
   } else {
      return dAry[x][y];
   }
   */
}

public Object viewInternal(TestCommand cmd) {
   List<Object>  result = new ArrayList<Object>();
   result.add(super.sm);   // 状態ノード
   //result.add(super.cm); // コマンドノード
   //result.add(super.rm); // 記録ノード
   result.add(this.PipeLength);// 配管長
   result.add(this.BoreDiameter); //内径
   result.add(this.TimeSpan); // 計算時間間隔
   result.add(this.EnvironmentTemp);   // 配管周囲温度
   result.add(this.InitInnerTemp);     // 初期内部温度
```

```
        result.add(this.Tp_Aster);        // 前回内部温度

        return result;
    }
}
```

名称　　：Duct 分岐（場所：設備 2015／ダクト配管 2015）
モジュール名　：DuctT_nOut1InModule20090101

名称　　：Duct 集合（場所：設備 2015／ダクト配管 2015）
モジュール名　：DuctT_nIn1OutModule20090101

(1) **入力画面**

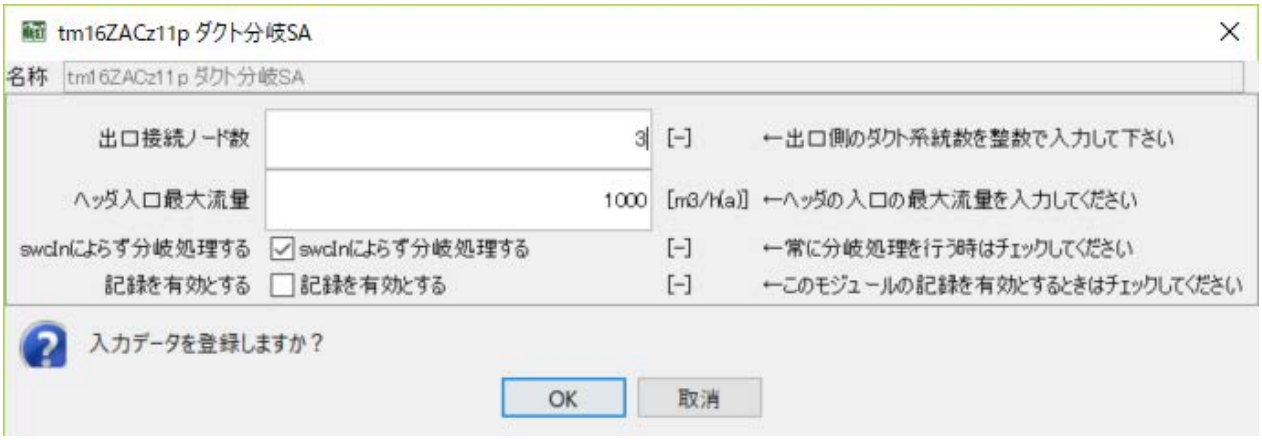


図 1　Duct 分岐モジュールの入力画面

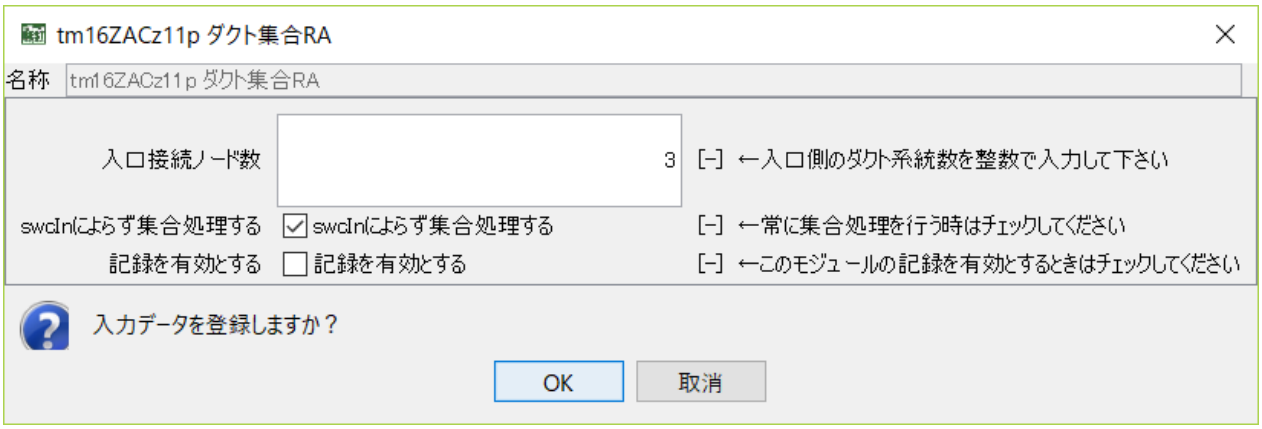


図 2　Duct 集合モジュールの入力画面

(2) **モジュールの概要**

本モジュールは、ダクトの分岐・集合を再現するモジュールである。

Duct 分岐とは、メインダクトから各分岐に入口空気状態（乾球温度・絶対湿度・CO2 濃度）を伝える。

Duct 集合とは、各分岐の空気状態から集合後の空気状態及び送風量を伝える。

(3) **モジュールの入出力**

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

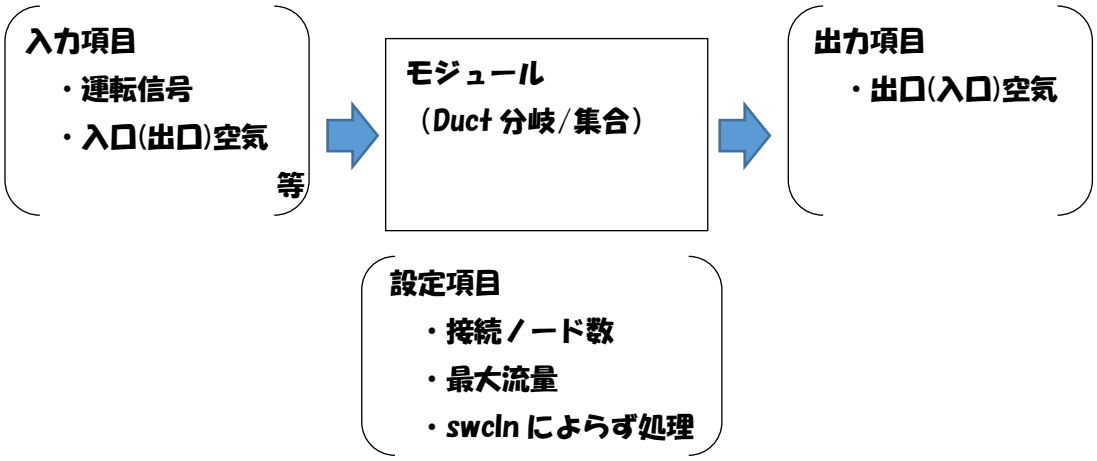


図 3 ダクトモジュール

(4) スペック入力項目（図 3 における設定項目）

表 1　Duct 分岐モジュールのスペック入力項目

NO.	項目	型・ｸﾗｽ名	記号	ﾃﾞﾌｫﾙﾄ値	単位	上限値	下限値	変更の重要度	備考
0	出口接続ﾉｰﾄﾞ数	int		2	[-]		1	◎	出口側のダクト系統数を整数で入力
1	ヘッダ入口最大流量	double		1000	[m3/h(a)]		0	◎	ヘッダの入口の最大流量を入力
2	swcIn によらず分岐処理する	boolean		FALSE	[-]			◎	SWC によらず常に分岐処理を行う時はﾁｪｯｸ
3	記録を有効とする	boolean	isRecord	FALSE	[-]				記録を有効とするときはﾁｪｯｸ

表 2　Duct 集合モジュールのスペック入力項目

NO.	項目	型・ｸﾗｽ名	記号	ﾃﾞﾌｫﾙﾄ値	単位	上限値	下限値	変更の重要度	備考
0	出口接続ﾉｰﾄﾞ数	int		2	[-]		1	◎	出口側のダクト系統数を整数で入力
1	swcIn によらず分岐処理する	boolean		FALSE	[-]			◎	SWC によらず常に分岐処理を行う時はﾁｪｯｸ
2	記録を有効とする	boolean	isRecord	FALSE	[-]				記録を有効とするときはﾁｪｯｸ

(5) シーケンス接続（図 3 における入力項目、出力項目）

表 3　Duct 分岐モジュールのシーケンス接続項目

No.	項目	接続端子名	記号	単位	ﾉｰﾄﾞ区分	媒体区分	InOut 区分	備考
0	入口空気	L0_airIn	DB_{in} ,X_{in} ,$CO2_{in}$,GA_{in}	℃、g/g、ppm、g/s	状態	空気	入口	
1	出口空気[i]	L0_airOut[i]	DB_{out}[i] ,X_{out}[i] ,$CO2_{out}$[i] ,GA_{out}[i]	℃、g/g、ppm、g/s	状態	空気	出口	
2	運転状態	L1_swcIn			制御	On/Off 信号	入口	
3	記録	L2_recOut			記録	ﾒﾓﾘ	出口	

注：i=0,1,2,3…と接続ﾉｰﾄﾞ数に応じて自動的に振られる。

表 4　Duct 集合モジュールのシーケンス接続項目

No.	項目	接続端子名	記号	単位	ﾉｰﾄﾞ区分	媒体区分	InOut 区分	備考
0	出口空気	L0_airOut	DB_{out} ,X_{out} ,$CO2_{out}$,GA_{out}	℃、g/g、ppm、g/s	状態	空気	出口	
1	入口空気[i]	L0_airIn[i]	DBi_{in}[i] ,X_{in}[i] ,$CO2_{in}$[i] ,GA_{in}[i]	℃、g/g、ppm、g/s	状態	空気	入口	
2	運転状態	L1_swcIn			制御	On/Off 信号	入口	
3	記録	L2_recOut			記録	ﾒﾓﾘ	出口	

注：i=0,1,2,3…と接続ﾉｰﾄﾞ数に応じて自動的に振られる。

(6) **記録項目**

表 5　Duct 分岐モジュールの記録項目

No.	項目	出力内容	単位	分類
4	メッセージ	メッセージ	-	メッセージ
5	乾球温度	温度	℃	入口
6	質量流量	質量流量	g/s	入口
7	絶対湿度	絶対湿度	g/g'	入口
8	CO2 濃度	質量流量	ppm	入口
9	乾球温度[i]	温度	℃	出口
10	質量流量[i]	質量流量	g/s	出口
11	絶対湿度[i]	絶対湿度	g/g'	出口
12	CO2 濃度[i]	質量流量	ppm	出口

注：i=0,1,2,3…と接続ノード数に応じて自動的に振られる。

表 6　Duct 集合モジュールの記録項目

No.	項目	出力内容	単位	分類
0	メッセージ	メッセージ	-	メッセージ
1	乾球温度	温度	℃	出口
2	質量流量	質量流量	g/s	出口
3	絶対湿度	絶対湿度	g/g'	出口
4	CO2 濃度	質量流量	ppm	出口
5	乾球温度[i]	温度	℃	入口
6	質量流量[i]	質量流量	g/s	入口
7	絶対湿度[i]	絶対湿度	g/g'	入口
8	CO2 濃度[i]	質量流量	ppm	入口

注：i=0,1,2,3…と接続ノード数に応じて自動的に振られる。

(7) 計算フロー・計算内容

Duct 分岐は、メインダクトから各分岐に入口空気状態（乾球温度・絶対湿度・CO2 濃度）を伝える。分岐部の風量に関しては、入口風量を出口風量の前ステップの風量の比率に応じて分配し、前ステップが停止時の場合および起動時は均等風量とする。また、ダクトから周囲空間への放熱ロスについては計算していない。

Duct 集合は、各分岐の空気状態から集合後の空気状態及び送風量を伝える。風量に応じた混合状態（乾球温度・絶対湿度・CO2 濃度・全風量）を計算する。

(8) データ範囲と適用範囲外の取扱い

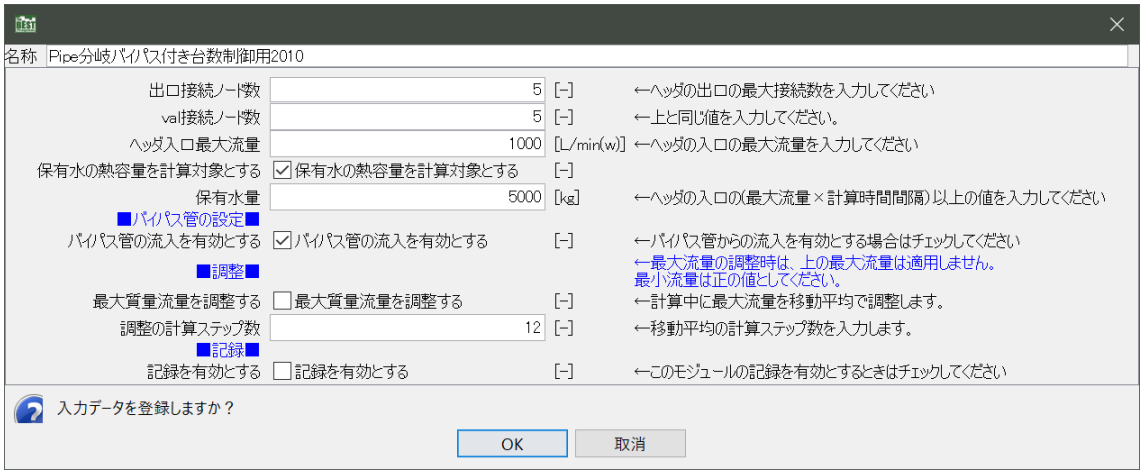
特になし。

「Pipe 分岐バイパス付き台数制御用」（場所：設備 2015／ダクト配管 2015／）

モジュール名	Pipe 分岐バイパス付き台数制御用
クラス	Pipe_1InnOutBypModule20101111

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestWater 媒体の複数の出口接続ノードと 1 個の入口接続ノードおよびバイパス管用入口接続ノードを有する配管ヘッダである。

入口とバイパス管からの流入する水を混合したヘッダ内状態を計算する。出口接続ノードから熱源群やポンプ群などの機器モジュールへ分岐送水する。

バイパス管用の接続ノードは入口（L0_watInBypass）扱いの名称であるが、バイパス管から流出する場合はその質量流量は負の値で判定し処理する。

出口接続ノードの数は設定可能で、その数の接続ノードが自動で用意されシーケンス接続画面で操作できる。val 接続ノードは出口接続ノードに対応したもので、熱源台数制御モジュールなどから各出口接続ノードの流量が設定される。

「保有水量の熱容量を計算対象とする」を有効とした場合、混合計算結果の出口水温は update()メソッド内で値を更新する。出口接続ノードの相手モジュールの outputs()メソッド時は更新前の水温が伝わることとなる。

「保有水量の熱容量を計算対象とする」が有効でない場合、混合計算結果の水温は outputs()メソッド内で出口水温として値を更新する。出口接続ノードの相手モジュールの outputs()メソッド時は更新された水温が伝わることとなる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

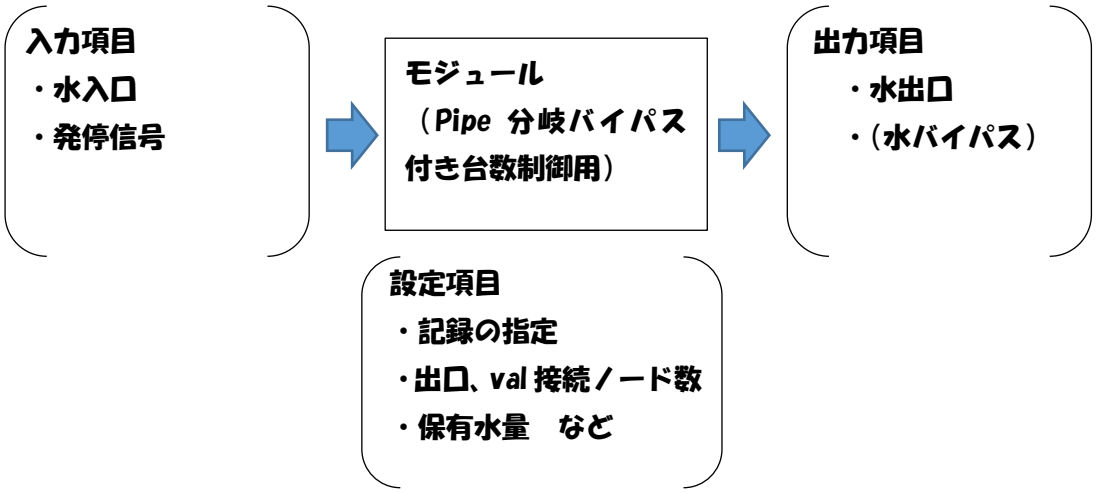


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	―	―		
1	入口接続ノード数	int	numberOfInlet	5	[-]	―	1		←ヘッダの入口の最大接続数を入力してください
2	val 接続ノード数	Int	numberOfInlet	5	[-]	―	1		←上と同じ値を入力してください
3	ヘッダ入口最大流量	double	watIn_maxFlowRate	1000	[L/min(w)]	―	0		←ヘッダの入口の最大流量を入力してください
4	保有水の熱容量を計算対象とする	boolean	isCalcTBuffer	TRUE	[-]	―	―		←選定可能な風量静圧に台数を調整して計算します。
5	保有水量	double	m_Buffer	5000	[kg]	―	0		←ヘッダの出口の（最大流量×計算時間間隔）以上の値を入力してください
6	■バイパス管の設定■								
7	バイパス管の流入を有効とする	Boolean	isBypassIn	TRUE	[-]	―	―		←バイパス管からの流入を有効とする場合はチェックしてください
8	■調整■					―	―		←最大流量の調整時は、上の最大流量は適用しません。流量は正の値としてください。
9	最大質量流量を調整する	boolean	isAdjust	FALSE	[-]				←計算中に最大流量を移動平均で調整します。
10	調整の計算ステップ数	int	numAdjustSteps	12	[-]	―	―		←移動平均の計算ステップ数を入力します。
11	■記録■					―	―		
12	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください

（５）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号
3	水入口	L0_watIn	watIn	-	状態	水	入口	水の入口
4	水出口[n]	L0_watOut[n]	watOut[]	-	状態	水	出口	水の出口[0]から[入口接続ノード数 - 1]
5	Val 入口[n]	L0_valInMwatOut[n]	valInMwatOut[]	-	状態	値	入口	出口[n]に対応する流量の値（熱源台数制御モジュール等が設定した流量）
6	水バイパス	L0_watInBypass	watInBypass	-	状態	水	入口	水のバイパス
7	水 My	L0_watMy	watMy	-	状態	水	My	ヘッダ内の水

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	–	メッセージ
2	入口温度	温度	℃	入口
3	入口流量	流量	g/s	入口
4	出口[n]温度（L0_watOut[n]_温度）	温度	℃	出口
5	出口[n]流量（L0_watOut[n]_流量）	流量	g/s	出口
6	バイパス入口温度	温度	℃	入口
7	バイパス入口流量	流量	g/s	入口
8	ヘッダ内温度	温度	℃	My
9	調整最大流量	流量	g/s	調整
10	調整ステップ平均流量	流量	g/s	調整

（7）計算フロー・計算内容

ヘッダ（分岐配管）への入口接続ノードからの水、バイパス用接続ノードからの水、保有水とから混合した水の状態を計算する。

「保有水量の熱容量を計算対象とする」を有効とした場合、出口水温は outputs()メソッド内では計算ステップ開始時のヘッダ内水温である。入口接続ノード、バイパス管ノード及び保有水量とから混合計算した結果の水温は、update()メソッド内で出口水温として値を更新する。出口接続ノードの相手モジュールの outputs()メソッド時は更新前の水温が伝わることとなる。

「保有水量の熱容量を計算対象とする」が有効でない場合、混合計算した結果の水温はoutputs()メソッド内で出口水温として値を更新する。出口接続ノードの相手モジュールのoutputs()メソッド時は更新された水温が伝わることとなる。

（8）データ範囲と範囲外の取扱い

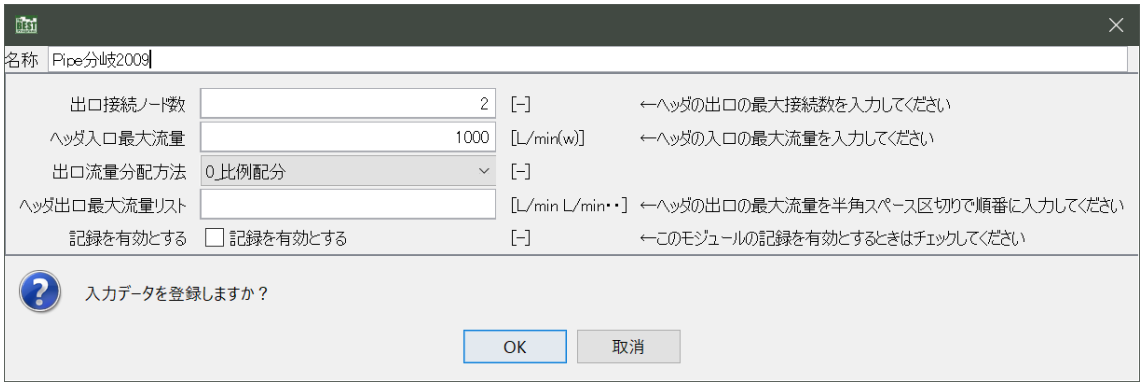
特になし。

「Pipe 分岐」（「Pipe 分岐 2009」）（場所：設備 2015／ダクト配管 2015／）

モジュール名	Pipe 分岐（2009）
クラス	PipeT_nOut1InModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestWater 媒体の複数の出口接続ノードと 1 個の入口接続ノードおよびバイパス管用出口接続ノードを有する水用分岐配管ヘッダである。

入口とバイパス管からの流入する水を混合したヘッダ内状態を計算する。出口接続ノードから熱源群やポンプ群などの機器モジュールへ分岐送水する。

バイパス管からは流出を正の値として扱う。

出口接続ノードの数は設定可能で、その数の接続ノードが自動で用意されシーケンス接続画面で操作できる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

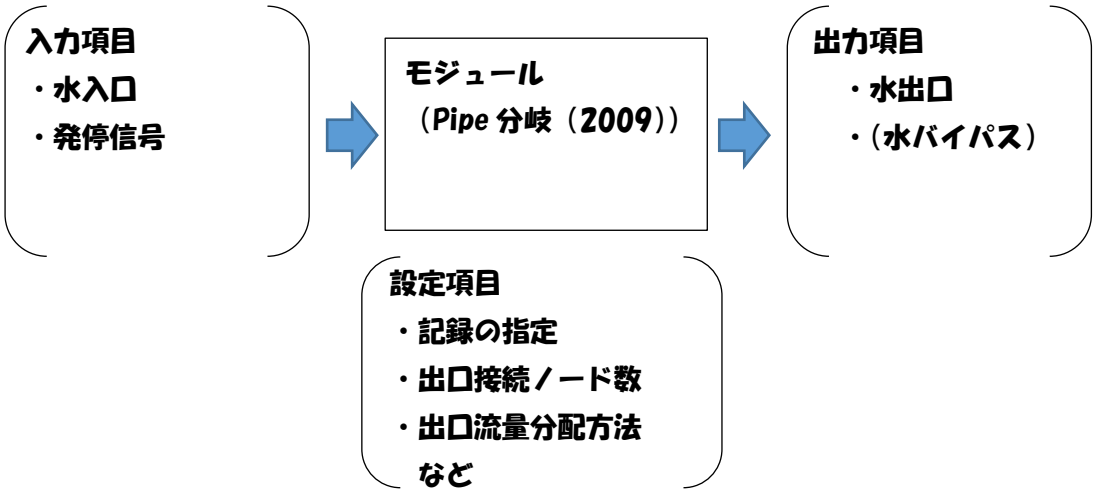


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	出口接続ノード数	int	numberOfOutlet	2	[-]	ー	1		←ヘッダの出口の最大接続数を入力してください
3	ヘッダ入口最大流量	double	watIn_maxFlowRate	1000	[L/min(w)]	ー	0		←ヘッダの入口の最大流量を入力してください
4	出口流量分配方法	String->int	num_distributionType	0_比例配分	[-]	ー	ー		
5	ヘッダ出口最大流量リスト	double	mMaxwatOut		[L/min(w) L/min(w)・・]	ー	ー		←ヘッダの出口の最大流量を半角スペース区切りで順番に入力してください
6	記録を有効とする	Boolean	isRecorde	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号
3	水入口	L0_watIn	watIn	-	状態	水	入口	水の入口
4	水出口[n]	L0_watOut[n]	watOut[]	-	状態	水	出口	水の出口[0]から[入口接続ノード数 - 1]
7	水バイパス	L0_watOutBypass	watOutBypass	-	状態	水	出口	水のバイパス

（6）記録項目

No.	項目	出力内容	単位	分類
1	PipeT_nOut1InMessage#-#-　メッセージ	メッセージ	－	メッセージ
2	PipeT_nOut1In 入口温度#℃#温度　入口温度	温度	℃	入口
3	PipeT_nOut1In 入口質量流量#g/s#質量流量	流量	g/s	入口
4	PipeT_nOut1In_watOut[]温度#℃#温度	温度	℃	出口
5	PipeT_nOut1In_watOut[]質量流量#g/s#流量	流量	g/s	出口
6	PipeT_nOut1In バイパス出口温度#℃#温度	温度	℃	出口
7	PipeT_nOut1In バイパス出口質量流量#g/s#質量流量	流量	g/s	出口
8	バイパス出口温度	温度	℃	出口
9	バイパス出口流量	流量	g/s	出口

（7）計算フロー・計算内容

ヘッダ（分岐配管）への入口接続ノードからの水、バイパス用接続ノードからの水とから混合した水の状態を計算する。

・L1_swcIn の OnOff 信号による動作

L1_swcIn の信号により、運転か停止かを判断する。
停止の場合は、出口水の流量を 0 とする。
運転の場合、指定された分配の方法によって次の動作となる。

0_比例配分

比例配分は、入口側の流量を、前のステップの分岐流量の比率で分配する。
前のステップの分岐流量の合計が 0 の場合は均等割りで計算する。

flowV＝Σ（ watOut[i].getFlowRate() ）　←前のステップの出口合計水量
flowV＝0 の時：
流量は均等割りとする。前ステップの流量がない時は起動時などで起こる。
watOut[i].setFlowRate(watIn.getFlowRate() / this.numberOfOutlet　)
各出口の温度は入口温度とする。
watOut[i].setTemp(watIn.getTemp());

flowV≠0 の時：
各出口の流量は、入口流量を前ステップの出口流量比で按分する。
watOut[i].setFlowRate(　watIn.getFlowRate() * watOut[i].getFlowRate() / flowV　)
各出口の温度は入口温度とする。
watOut[i].setTemp(watIn.getTemp());
バイパスについては、流量は 0、温度は入口温度とする。
watOutBypass.setFlowRate(0)
watOutBypass.setTemp(watIn.getTemp())

1_最大流量比配分

各出口の流量は、ヘッダ出口最大流量リストの流量比で入口流量を按分する。
sum_mMaxwatOut＝Σ(mMaxwatOut[i])　←ヘッダ出口最大流量リストの合計値
watOut[i].setFlowRate(watIn.getFlowRate() * mMaxwatOut[i] / sum_mMaxwatOut)
各出口の温度は入口温度とする。
watOut[i].setTemp(watIn.getTemp())
バイパスについては、流量は 0、温度は入口温度とする。
watOutBypass.setFlowRate(0)

```
watOutBypass.setTemp( watIn.getTemp() )
```

2_優先最大流量配分

優先最大流量配分は、入口流量をヘッダ出口最大流量リストの流量までリスト順に割り当てる。各出口の水温は入口水温とする。

```
flowRatewatIn = this.watIn.getFlowRate();
for( int i=0; i<this.watOut.length; i++ ){
    if( flowRatewatIn > 0 ){
        watOut[i].setFlowRate( Math.min( mMaxwatOut[i], flowRatewatIn ))
        flowRatewatIn -= watOut[i].getFlowRate()
    }else{
        watOut[i].setFlowRate( 0. )
    }
    watOut[i].setTemp( watIn.getTemp() );
}
```

バイパスについては、流量は出口へ割り当て後の余剰流量、温度は入口温度とする。

```
watOutBypass.setFlowRate( flowRatewatIn );
if( flowRatewatIn>0 ){
    watOutBypass.setTemp( watIn.getTemp() );
}
```

（8）データ範囲と範囲外の取扱い

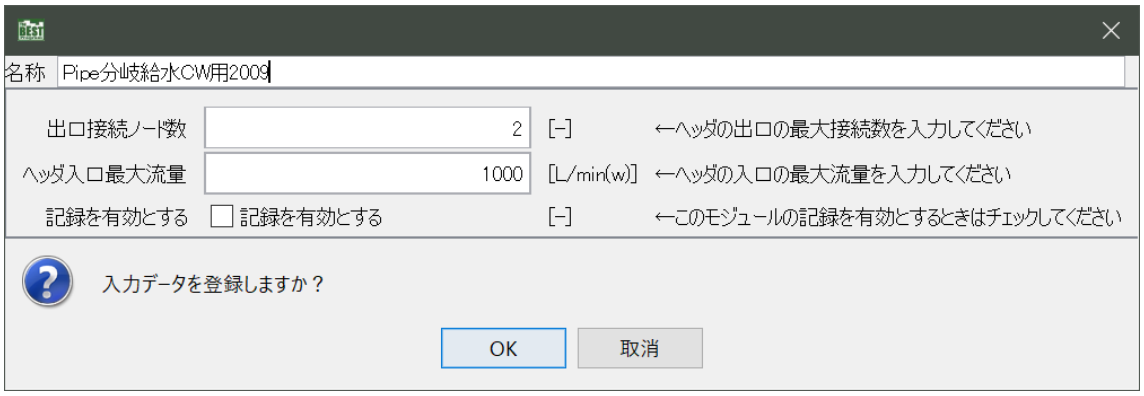
特になし。

「Pipe 分岐給水 CW 用」(「Pipe 分岐給水 CW2009」)(場所：設備 2015／ダクト配管 2015／)

モジュール名	Pipe 分岐給水 CW 用（2009）
クラス	PipeT_nOut1InforCWModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestWater 媒体の複数の出口接続ノードと 1 個の入口接続ノードを有する給水 CW 分岐用配管ヘッダである。

給水 CW 出口の接続相手は水を消費する機器・器具などである。

給水 CW 出口に接続された水の使用量の合計値を入口の水の使用量として上位（給水 CW 供給側）へ伝達する。

給水 CW 出口接続ノードの数は設定可能で、その数の接続ノードが自動で用意されシーケンス接続画面で操作できる。

☞ 水用の配管分岐は、入口側から出口側を設定する。

給水 CW 用の配管分岐は、使用量は出口側から入口側を設定し、温度は入口側から出口側を設定する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

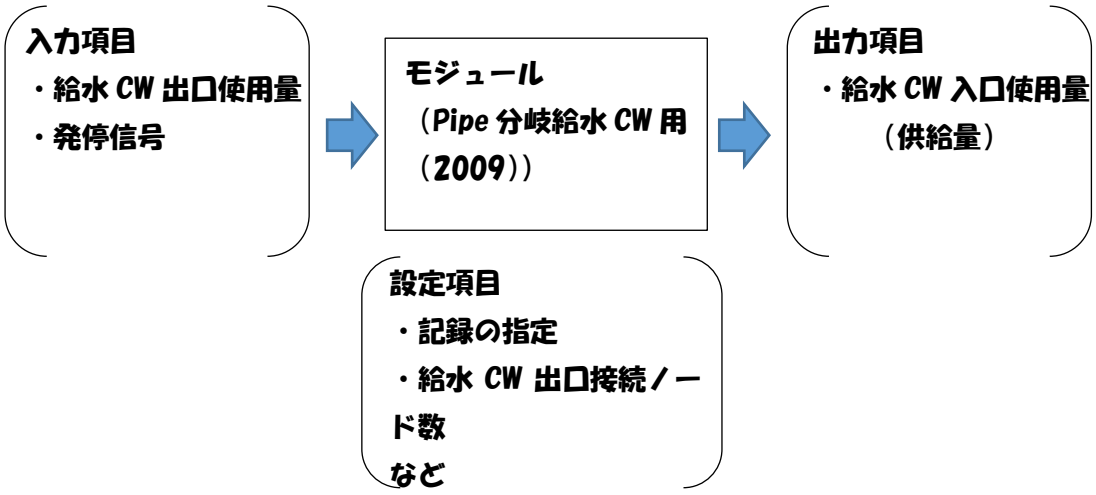


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	出口接続ノード数	int	numberOfOutlet	2	[-]	ー	1		←ヘッダの出口の最大接続数を入力してください
2	ヘッダ入口最大流量	double	watIn_maxFlowRate	1000	[L/min(w)]	ー	0		←ヘッダの入口の最大流量を入力してください
3	記録を有効とする	Boolean	isRecorde	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号
3	給水 CW 入口	L0_watInCW	watInCW	-	状態	給水 CW	入口	給水 CW の入口
4	給水 CW 出口[n]	L0_watOutCW[n]	watOutCW[]	-	状態	給水 CW	出口	給水 CW の出口[0]から[入口接続ノード数 - 1]

（6）記録項目

No.	項目	出力内容	単位	分類
1	PipeTnOut1InCW_Message#-#-	メッセージ	ー	メッセージ
2	PipeTnOut1InCW_入口温度#℃#温度	温度	℃	入口
3	PipeTnOut1InCW_入口質量流量#g/s#質量流量	質量流量	g/s	入口
4	PipeTnOut1InCW_watOutCW[]温度#℃#温度	温度	℃	出口
5	PipeTnOut1InCW_watOutCW[]入口質量流量#g/s#質量流量	質量流量	g/s	出口

（7）計算フロー・計算内容

ヘッダ（給水 CW 分岐配管）の出口接続ノードの水使用量から入口の給水量（給水 CW 出口の使用量の合計）を計算する。

・L1_swcln の OnOff 信号による動作

L1_swcln の信号により、運転か停止かを判断する。

停止の場合は、入口給水 CW の流量を 0 とする。

運転の場合、出口給水 CW の流量の合計値を入口給水 CW の流量とする。

出口給水 CW の温度は入口給水 CW の温度とする。

（8）データ範囲と範囲外の取扱い

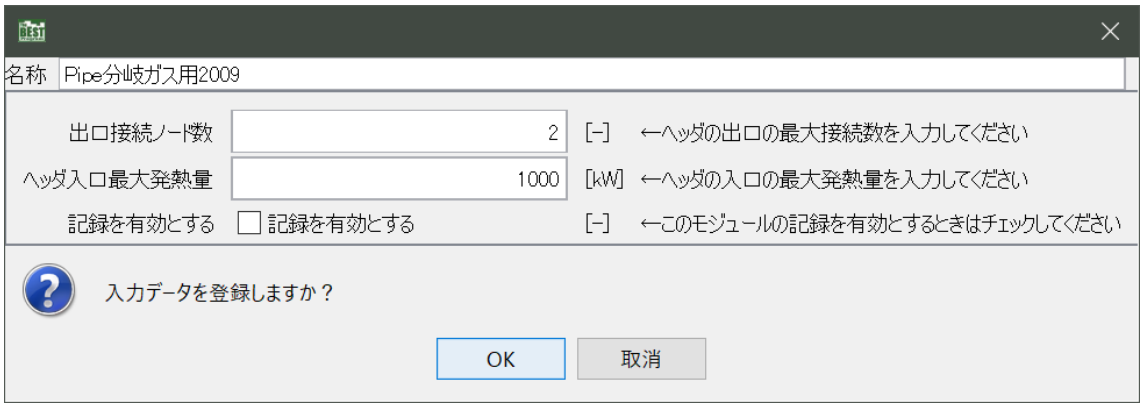
特になし。

「Pipe 分岐ガス用」(「Pipe 分岐 2009」)（場所：設備 2015／ダクト配管 2015／）

モジュール名	Pipe 分岐ガス用（2009）
クラス	PipeT_nOut1InforGasModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestGas 媒体の複数の出口接続ノードと 1 個の入口接続ノードを有するガス分岐用配管ヘッダである。

ガス出口の接続相手はガスを消費する機器などである。

ガス出口に接続されたガス消費量の合計値を入口のガス消費量として上位（ガス供給側）へ伝達する。

ガス出口接続ノードの数は設定可能で、その数の接続ノードが自動で用意されシーケンス接続画面で操作できる。

☞ 水用の配管分岐は、入口側から出口側を設定する。

ガス用の配管分岐は、発熱量は出口側から入口側を設定し、温度は入口側から出口側を設定する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

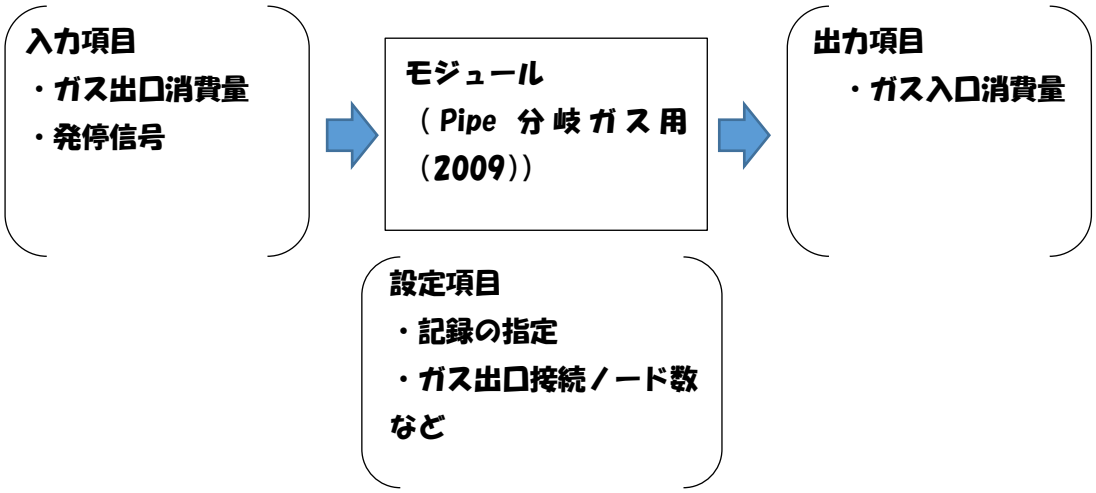


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	出口接続ノード数	int	numberOfOutlet	2	[-]	ー	1		←ヘッダの出口の最大接続数を入力してください
2	ヘッダ入口最大発熱量	double	gasIn_maxWatt	1000	[kW]	ー	0		←ヘッダの入口の最大発熱量を入力してください
3	記録を有効とする	Boolean	isRecorde	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号
3	ガス入口	L0_gasIn	gasIn	-	状態	ガス	入口	ガスの入口
4	ガス出口[n]	L0_gasOut[n]	gasOut[]	-	状態	ガス	出口	ガスの出口[0]から[入口接続ノード数 - 1]

（6）記録項目

No.	項目	出力内容	単位	分類
1	PipeTnOut1InGas_Message#-#-	メッセージ	ー	メッセージ
2	PipeTnOut1InGas_入口温度#℃#温度	温度	℃	入口
3	PipeTnOut1InGas_入口発熱量#W#熱量	発熱量	W	入口
4	PipeTnOut1InGas_gasOut[]温度#℃#温度	温度	℃	出口
5	PipeTnOut1InGas_gasOut[]発熱量#W#熱量	発熱量	W	出口

（７）計算フロー・計算内容

ヘッダ（ガス分岐配管）の出口接続ノードのガス発熱量から入口のガス発熱量（ガス出口の発熱量の合計）を計算する。

・L1_swcln の OnOff 信号による動作

L1_swcln の信号により、運転か停止かを判断する。

停止の場合は、入口ガスの発熱量を 0 とする。

運転の場合、出口ガスの発熱量の合計値を求めそれを入口ガスの発熱量とする。

出口ガスの温度は入口ガスの温度とする。

（８）データ範囲と範囲外の取扱い

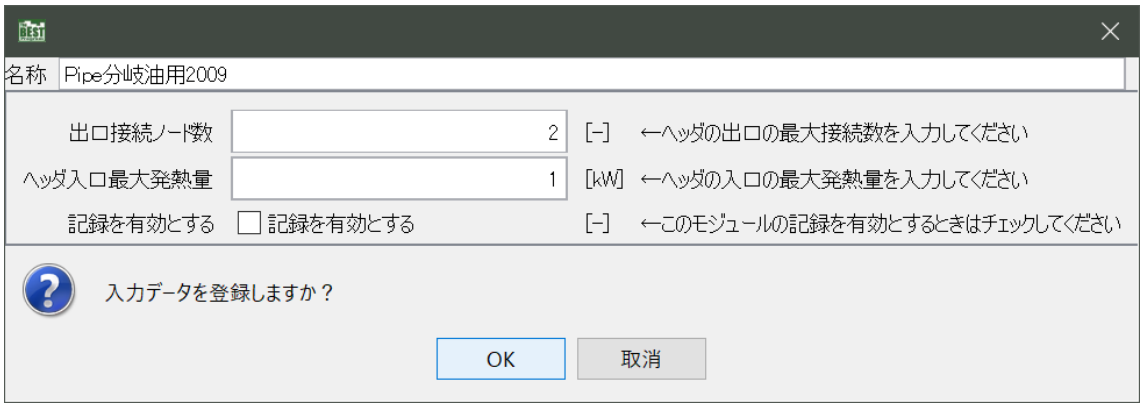
特になし。

「Pipe 分岐油用」（「Pipe 分岐油 2009」）（場所：設備 2015／ダクト配管 2015／）

モジュール名	Pipe 分岐油用（2009）
クラス	PipeT_nOut1InforOilModule20090101

（１）入力画面

・スペック



（２）モジュールの概要

本モジュールは、BestOil 媒体の複数の出口接続ノードと 1 個の入口接続ノードを有する油分岐用配管ヘッダである。

油出口の接続相手は油を消費する機器などである。

油出口に接続された油消費量の合計値を入口の油消費量として上位（油供給側）へ伝達する。

油出口接続ノードの数は設定可能で、その数の接続ノードが自動で用意されシーケンス接続画面で操作できる。

☞　水用の配管分岐は、入口側から出口側を設定する。

油用の配管分岐は、発熱量は出口側から入口側を設定し、温度は入口側から出口側を設定する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

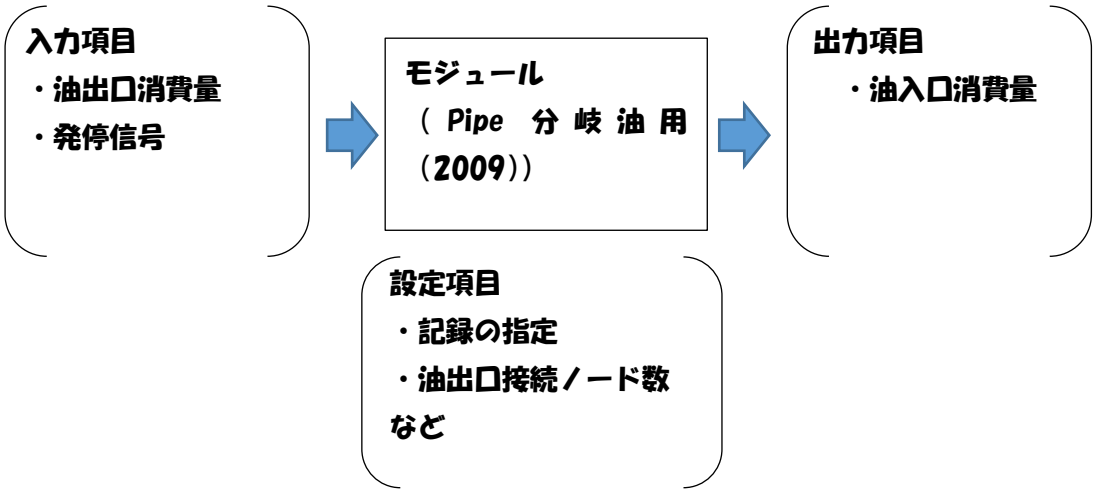


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	出口接続ノード数	int	numberOfOutlet	2	[-]	ー	1		←ヘッダの出口の最大接続数を入力してください
2	ヘッダ入口最大発熱量	double	oilIn_maxWatt	1000	[kW]	ー	0		←ヘッダの入口の最大発熱量を入力してください
3	記録を有効とする	Boolean	isRecorde	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号
3	油入口	L0_oilIn	oilIn	-	状態	油	入口	油の入口
4	油出口[n]	L0_oilOut[n]	oilOut[]	-	状態	油	出口	油の出口[0]から[入口接続ノード数 - 1]

（6）記録項目

No.	項目	出力内容	単位	分類
1	PipeTnOut1InOil_Message#-#-	メッセージ	ー	メッセージ
2	PipeTnOut1InOil_入口温度#℃#温度	温度	℃	入口
3	PipeTnOut1InOil_入口発熱量#W#熱量	発熱量	W	入口
4	PipeTnOut1InOil_oilOut[]温度#℃#温度	温度	℃	出口
5	PipeTnOut1InOil_oilOut[]発熱量#W#熱量	発熱量	W	出口

（７）計算フロー・計算内容

ヘッダ（油分岐配管）の出口接続ノードのガス発熱量から入口の油発熱量（油出口の発熱量の合計）を計算する。

・L1_swcln の OnOff 信号による動作

L1_swcln の信号により、運転か停止かを判断する。

停止の場合は、入口油の発熱量を 0 とする。

運転の場合、出口油の発熱量の合計値を求めそれを入口油の発熱量とする。

出口油の温度は入口油の温度とする。

（８）データ範囲と範囲外の取扱い

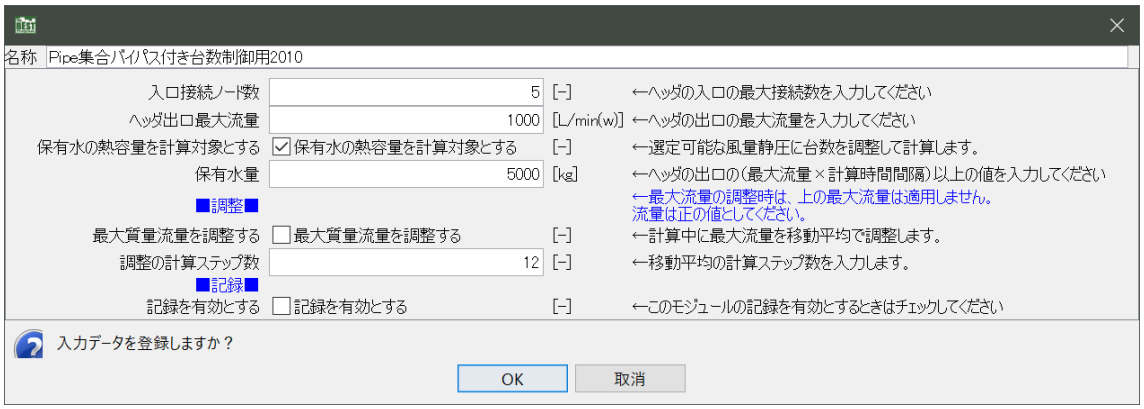
特になし。

「Pipe 集合バイパス付き台数制御用」（場所：設備 2015／ダクト配管 2015／）

モジュール名	Pipe 集合バイパス付き台数制御用
クラス	Pipe_nIn1OutBypOutModule20101111

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestWater 媒体の複数の入口接続ノードと 1 個の出口接続ノードおよびバイパス管用出口接続ノードを有する配管ヘッダである。

熱源群やポンプ群などで機器モジュールからの冷温水などを集合し、それらを混合した出口状態を計算する。

バイパス管用の接続ノードは出口（L0_watOutBypass）扱いの名称であるが、バイパス管から流入する場合はその質量流量は負の値で判定し処理する。

入口接続ノードの数は設定可能で、その数の接続ノードが自動で用意されシーケンス接続画面で操作できる。

「保有水量の熱容量を計算対象とする」を有効とした場合、混合計算結果の出口水温は update()メソッド内で値を更新する。出口接続ノードの相手モジュールの outputs()メソッド時は更新前の水温が伝わることとなる。

「保有水量の熱容量を計算対象とする」が有効でない場合、混合計算結果の水温は outputs()メソッド内で出口水温として値を更新する。出口接続ノードの相手モジュールの outputs()メソッド時は更新された水温が伝わることとなる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

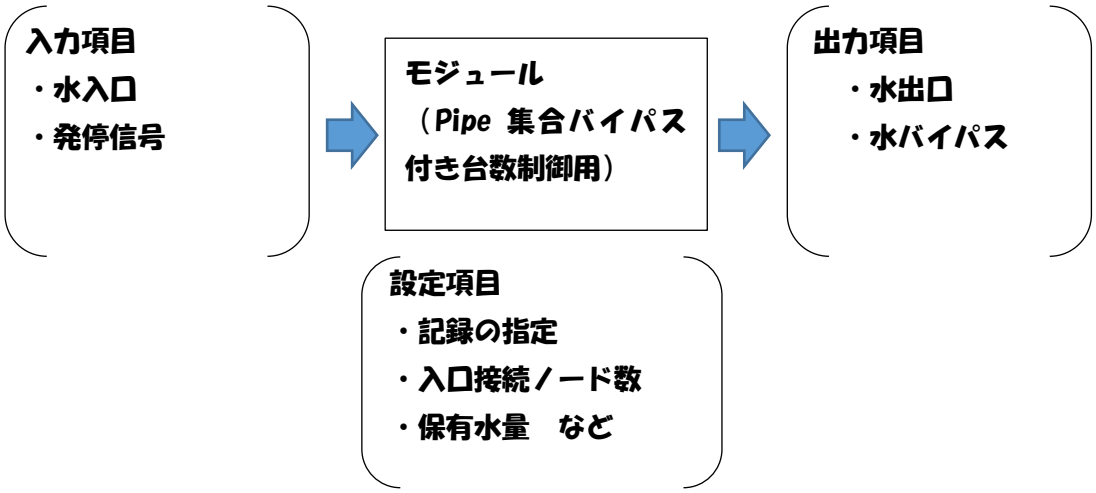


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	入口接続ノード数	int	numberOfInlet	5	[-]	ー	1		←ヘッダの入口の最大接続数を入力してください
2	ヘッダ出口最大流量	double	maxM_wOut	1000	[L/min(w)]	ー	0		←ヘッダの出口の最大流量を入力してください
3	保有水の熱容量を計算対象とする	boolean	isCalcTBuffer	TRUE	[-]	ー	ー		←選定可能な風量静圧に台数を調整して計算します。
4	保有水量	double	m_Buffer	5000	[kg]	ー	0		←ヘッダの出口の（最大流量×計算時間間隔）以上の値を入力してください
5	■調整■					ー	ー		←最大流量の調整時は、上の最大流量は適用しません。流量は正の値としてください。
6	最大質量流量を調整する	boolean	isAdjust	FALSE	[-]				←計算中に最大流量を移動平均で調整します。
7	調整の計算ステップ数	int	numAdjustSteps	12	[-]	ー	ー		←移動平均の計算ステップ数を入力します。
8	■記録■					ー	ー		
9	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（５）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号
3	水出口	L0_watOut	watOut	-	状態	水	出口	水の出口
4	水入口[n]	L0_watIn[n]	watIn[]	-	状態	水	出口	水の入口[0]から[入口接続ノード数 - 1]
5	水バイパス	L0_watOutBypass	watOutBypass	-	状態	水	出口	水のバイパス
6	水 My	L0_watMy	watMy	-	状態	水	My	ヘッダ内の水

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
2	出口温度	温度	℃	出口
3	出口流量	流量	g/s	出口
4	入口[n]温度	温度	℃	入口
5	入口[n]流量	流量	g/s	入口
6	バイパス出口温度	温度	℃	出口
7	バイパス出口流量	流量	g/s	出口
8	バイパス出口流量（−）	流量	g/s	出口
9	ヘッダ内温度	温度	℃	My
10	調整最大流量	流量	g/s	調整
11	調整ステップ平均流量	流量	g/s	調整

（7）計算フロー・計算内容

ヘッダ（集合配管）への入口接続ノードからの水、バイパス用接続ノードからの水、保有水とから混合した水の状態を計算する。

「保有水量の熱容量を計算対象とする」を有効とした場合、出口水温は outputs()メソッド内では計算ステップ開始時のヘッダ内水温である。入口接続ノード、バイパス管ノード及び保有水量とから混合計算した結果の水温は、update()メソッド内で出口水温として値を更新する。出口接続ノードの相手モジュールの outputs()メソッド時は更新前の水温が伝わることとなる。

「保有水量の熱容量を計算対象とする」が有効でない場合、混合計算した結果の水温はoutputs()メソッド内で出口水温として値を更新する。出口接続ノードの相手モジュールのoutputs()メソッド時は更新された水温が伝わることとなる。

（8）データ範囲と範囲外の取扱い

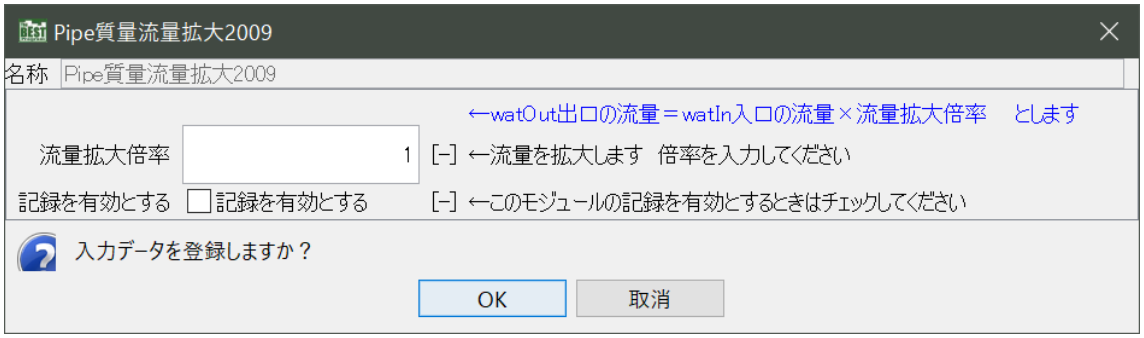
特になし。

「Pipe 質量流量拡大」（場所：設備 2015／ダクト配管 2015／）

モジュール名	Pipe_質量流量拡大
クラス	EnlargeFlowRateWaterModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestWater 媒体の入口接続ノードの流量にユーザーが入力した倍率を乗じたものを出口接続ノードの流量として渡すものである。拡大するのは流量のみで、その他の水温等は入口状態＝出口状態とする。

例えば、事務所ビルなどで基準階の空調機の冷温水負荷を基準階の階数倍する場合に使用する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

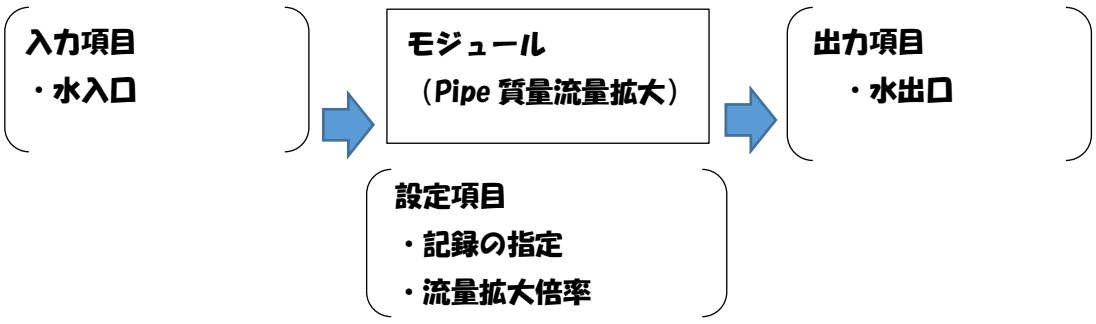


図 1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	流量拡大倍率	double	enlarge	1	[-]	ー	0		←流量を拡大します　倍率を入力してください
2	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号 今は、このモジュールの動作には影響しない
3	水入口	L0_watIn	watIn	-	状態	水	入口	水の入口
4	水出口	L0_watOut	watOut	-	状態	水	出口	水の出口

（6）記録項目（EnFRwat_）

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
2	入口流量	流量	g/s	入口
3	出口流量	流量	g/s	出口

（7）計算フロー・計算内容

入口接続ノードからの水の流量にユーザー入力の拡大倍率を乗じたものを出口接続ノードの水の流量として渡す。

watOut 出口の流量[g/s]　＝　watIn 入口の流量[g/s]　×　流量拡大倍率

その他は入口と同じ状態値とする。

watOut 出口の温度[℃]　＝　watIn 入口の温度[℃]

（8）データ範囲と範囲外の取扱い

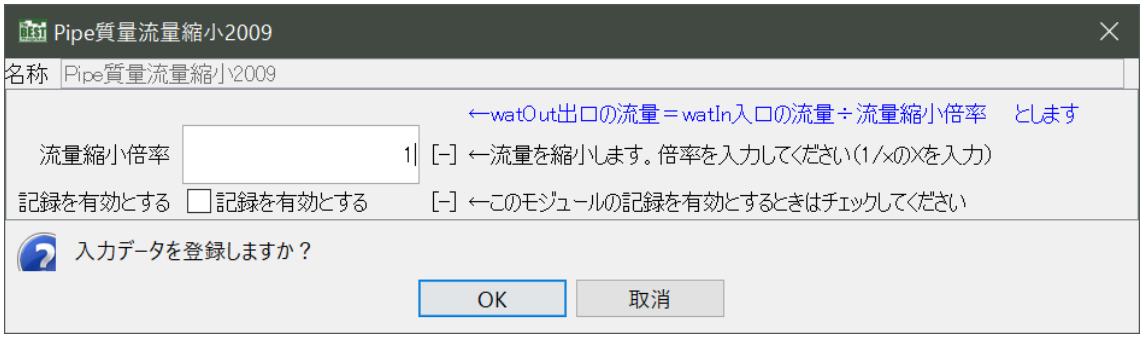
特になし。

「Pipe 質量流量縮小」（場所：設備 2015／ダクト配管 2015／）

モジュール名	Pipe_質量流量縮小
クラス	ReduceFlowRateWaterModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestWater 媒体の入口接続ノードの流量にユーザーが入力した倍率で割ったものを出口接続ノードの流量として渡すものである。縮小するのは流量のみで、その他の水温等は入口状態＝出口状態とする。

例えば、事務所ビルなどで熱源からの冷温水を基準階の空調機の冷温水負荷の 1 階分に縮小する場合に使用する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

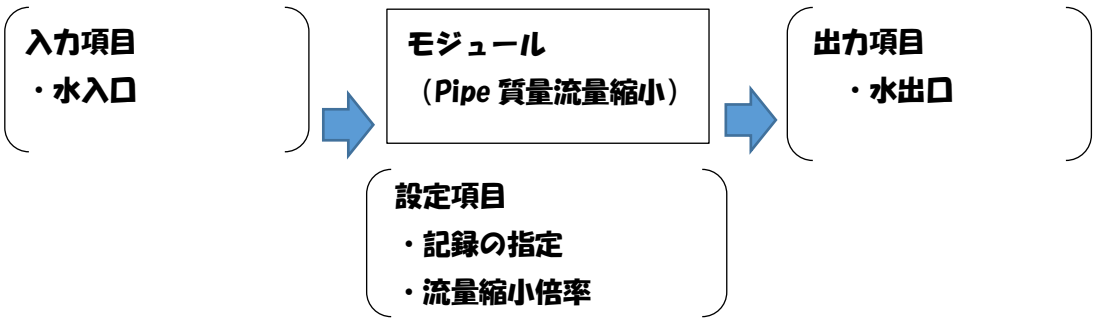


図 1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	流量拡大倍率	double	reduce	1	[-]	ー	0		←流量を縮小します。倍率を入力してください（1/ｘのＸを入力）
2	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号 今は、このモジュールの動作には影響しない
3	水入口	L0_watIn	watIn	-	状態	水	入口	水の入口
4	水出口	L0_watOut	watOut	-	状態	水	出口	水の出口

（6）記録項目（WFRP）

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
2	入口流量	流量	g/s	入口
3	出口流量	流量	g/s	出口

（7）計算フロー・計算内容

入口接続ノードからの水の流量をユーザー入力の縮小倍率で割ったものを出口接続ノードの水の流量として渡す。

watOut 出口の流量[g/s]　＝　watIn 入口の流量[g/s]　÷　流量縮小倍率

その他は入口と同じ状態値とする。

watOut 出口の温度[℃]　＝　watIn 入口の温度[℃]

（8）データ範囲と範囲外の取扱い

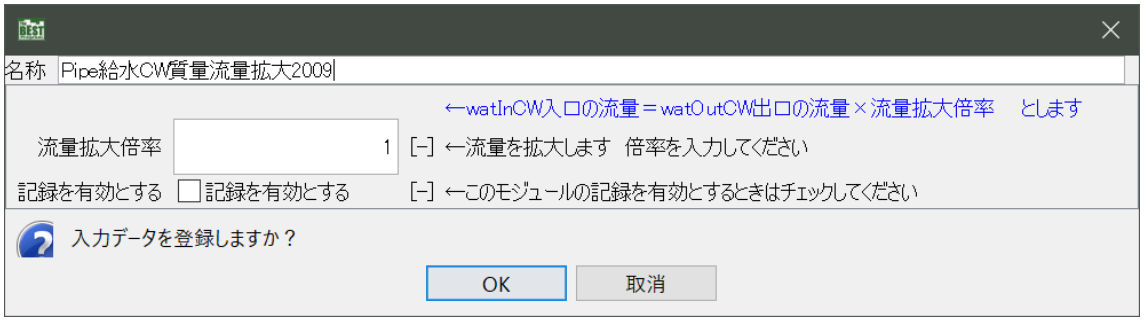
特になし。

「Pipe 質量流量拡大（給水 CW)」（場所：設備 2015／ダクト配管 2015／）

モジュール名	Pipe_質量流量拡大（給水 CW）
クラス	EnlargeFlowRateWaterCWModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestWater 媒体の出口接続ノードの流量にユーザーが入力した倍率を乗じたものを入口接続ノードの流量として渡すものである。拡大するのは流量のみで、その他の水温等は入口状態＝出口状態とする。

例えば、事務所ビルなどで基準階の衛生給水負荷を基準階の階数倍する場合に使用する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

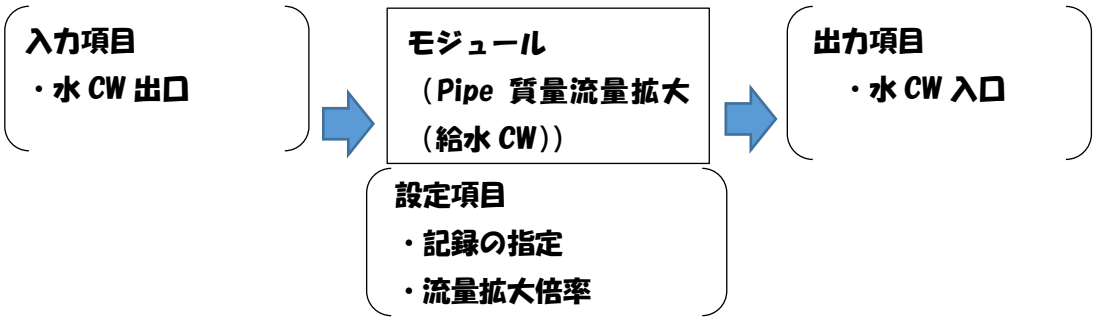


図 1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	流量拡大倍率	double	enlarge	1	[-]	ー	0		←流量を拡大します　倍率を入力してください
2	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号 今は、このモジュールの動作には影響しない
3	水入口	L0_watInCW	watInCW	-	状態	水 CW	入口	水 CW の入口
4	水出口	L0_watOutCW	watOutCW	-	状態	水 CW	出口	水 CW の出口

（6）記録項目（EnFRwatCW_）

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	−	メッセージ
2	入口流量	流量	g/s	入口
3	出口流量	流量	g/s	出口
4	入口温度	温度	℃	入口
5	出口温度	温度	℃	出口

（7）計算フロー・計算内容

入口接続ノードからの水の流量にユーザー入力の拡大倍率を乗じたものを出口接続ノードの水の流量として渡す。

watInCW 入口の流量[g/s]　＝　watOutCW 出口の流量[g/s]　×　流量拡大倍率

☞　給水 CW の出口水温は入口水温とする。

watOutCW 出口の温度[℃]　＝　watInCW 入口の温度[℃]

（8）データ範囲と範囲外の取扱い

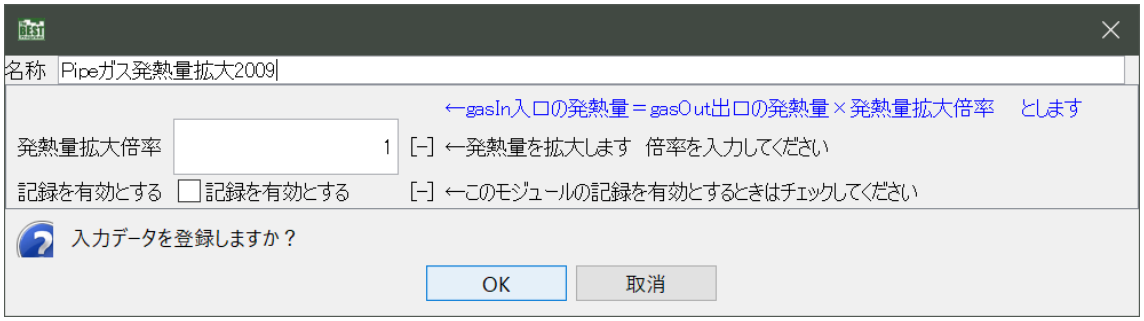
特になし。

「Pipe 発熱量拡大（ガス）」（場所：設備 2015／ダクト配管 2015／）

モジュール名	Pipe_発熱量拡大（ガス）
クラス	EnlargeWattGasModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestGas 媒体の出口接続ノードの電力量にユーザーが入力した倍率を乗じたものを入口接続ノードの発熱量として渡すものである。拡大するのは発熱量のみで、その他の項目は入口状態＝出口状態とする。

例えば、事務所ビルなどで基準階の GHP ビルマルチ等のガス発熱量を基準階の階数倍する場合に使用する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

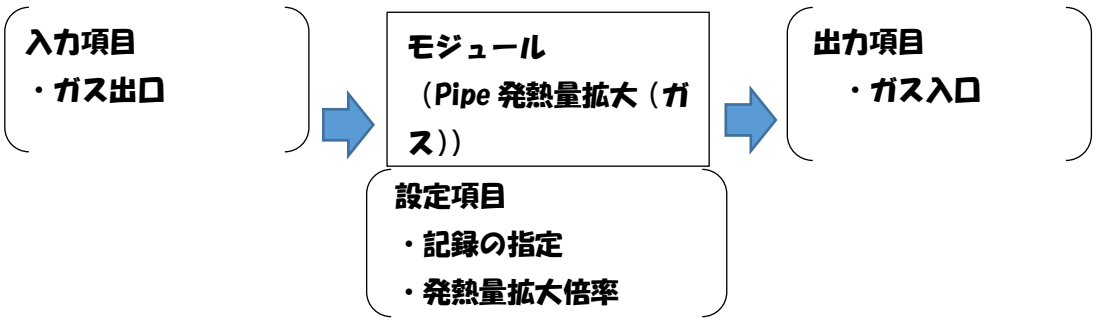


図 1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	発熱量拡大倍率	double	enlarge	1	[-]	ー	0		←発熱量を拡大します　倍率を入力してください
2	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号 今は、このモジュールの動作には影響しない
3	ガス入口	L0_gasIn	gasIn	-	状態	ガス	入口	ガスの入口
4	ガス出口	L0_gasOut	gasOut	-	状態	ガス	出口	ガスの出口

（6）記録項目（EnWattGas_）

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	－	メッセージ
2	入口発熱量	ガス発熱量	W	入口
3	出口発熱量	ガス発熱量	W	出口

（7）計算フロー・計算内容

出口接続ノードからの電力量にユーザー入力の拡大倍率を乗じたものを入口接続ノードの電力量として渡す。

gasIn 入口の発熱量[W]　＝　gasOut 出口の発熱量[W]　×　発熱量拡大倍率

その他は入口と同じ状態値とする。

gasIn 入口の圧力[Pa]　＝　gasOut 出口の圧力[Pa]

（8）データ範囲と範囲外の取扱い

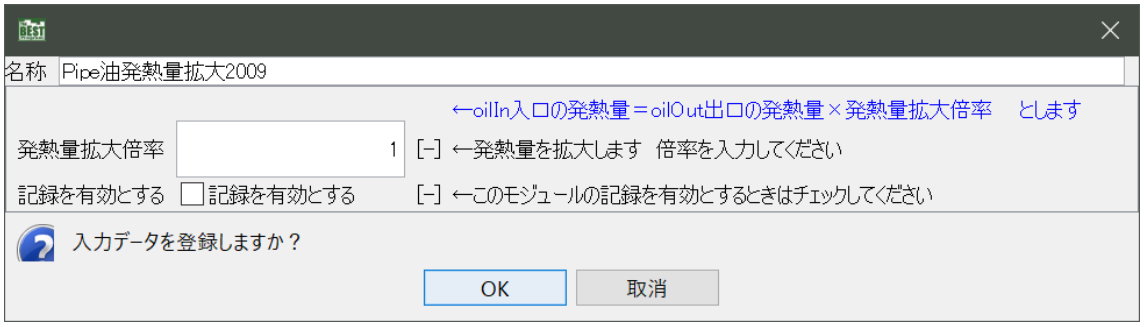
特になし。

「Pipe 発熱量拡大（油）」（場所：設備 2015／ダクト配管 2015／）

モジュール名	Pipe_発熱量拡大（油）
クラス	EnlargeWattOilModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestOil 媒体の出口接続ノードの電力量にユーザーが入力した倍率を乗じたものを入口接続ノードの発熱量として渡すものである。拡大するのは発熱量のみで、その他の項目は入口状態＝出口状態とする。

例えば、事務所ビルなどで基準階の KHP ビルマルチ等の油の発熱量を基準階の階数倍する場合に使用する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

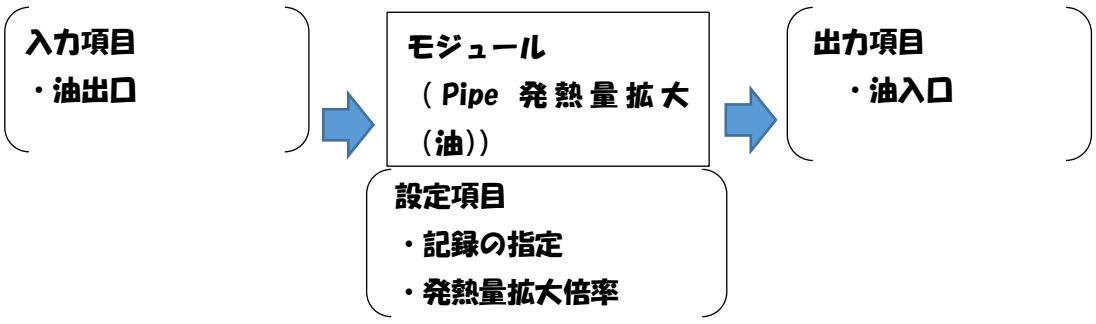


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	発熱量拡大倍率	double	enlarge	1	[-]	ー	0		←発熱量を拡大します　倍率を入力してください
2	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号 今は、このモジュールの動作には影響しない
3	油入口	L0_oilIn	oilIn	-	状態	油	入口	油の入口
4	油出口	L0_oilOut	oilOut	-	状態	油	出口	油の出口

（6）記録項目（EnWattOil_）

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	－	メッセージ
2	入口発熱量	油発熱量	W	入口
3	出口発熱量	油発熱量	W	出口

（7）計算フロー・計算内容

出口接続ノードからの電力量にユーザー入力の拡大倍率を乗じたものを入口接続ノードの電力量として渡す。

oilIn 入口の発熱量[W]　＝　oilOut 出口の発熱量[W]　×　発熱量拡大倍率

その他は入口と同じ状態値とする。

oilIn 入口の圧力[Pa]　＝　oilOut 出口の圧力[Pa]

（8）データ範囲と範囲外の取扱い

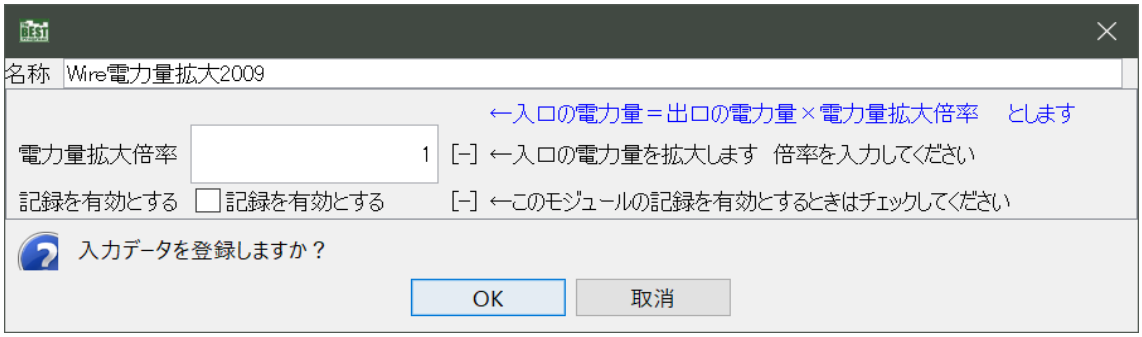
特になし。

「Wire 電力量拡大」（場所：設備 2015／ダクト配管 2015／）

モジュール名	Wire_電力量拡大
クラス	EnlargeWattEleModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestElectricity 媒体の出口接続ノードの電力量にユーザーが入力した倍率を乗じたものを入口接続ノードの電力量として渡すものである。拡大するのは電力量のみで、その他の電圧等は入口状態＝出口状態とする。

例えば、事務所ビルなどで基準階の空調機ファン等の電力量を基準階の階数倍する場合に使用する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

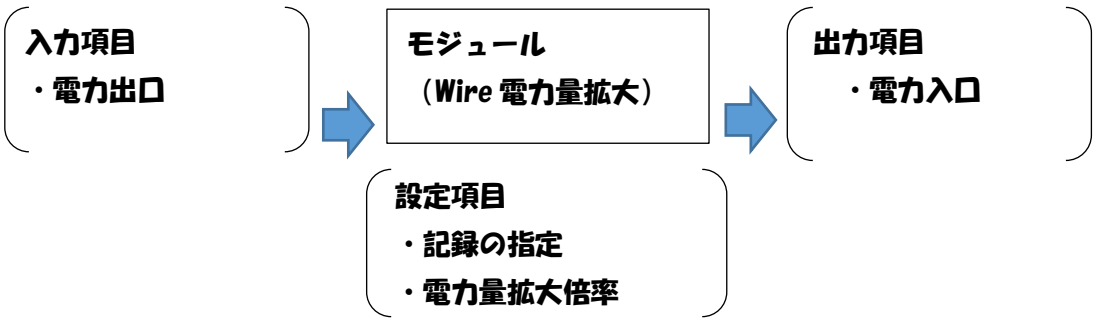


図 1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	電力量拡大倍率	double	enlarge	1	[-]	ー	0		←入口の電力量を拡大します　倍率を入力してください
2	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号 今は、このモジュールの動作には影響しない
3	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	電力の入口
4	電力出口	L0_eleOut	eleOut	-	状態	電力	出口	電力の出口

（6）記録項目（EnWattEle_）

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	－	メッセージ
2	入口電力量	電力量	W	入口
3	出口電力量	電力量	W	出口

（7）計算フロー・計算内容

出口接続ノードからの電力量にユーザー入力の拡大倍率を乗じたものを入口接続ノードの電力量として渡す。

eleIn 入口の電力量[W]　＝　eleOut 出口の電力量[W]　×　電力量拡大倍率

その他は入口と同じ状態値とする。

eleIn 入口の電圧[V]　＝　eleOut 出口の電圧[V]

（8）データ範囲と範囲外の取扱い

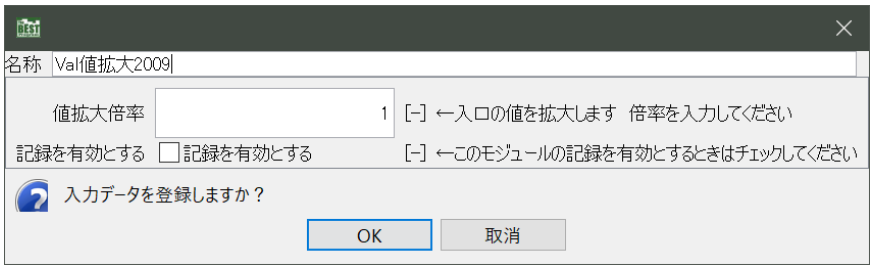
特になし。

「Val 値拡大」（場所：設備 2015／ダクト配管 2015／）

モジュール名	Val_値拡大
クラス	EnlargeValueValModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、BestValue 媒体の入口接続ノードの値にユーザーが入力した倍率を乗じたものを出口接続ノードの値として渡すものである。拡大するのは値のみで、その他の項目は入口状態＝出口状態とする。

例えば、事務所ビルなどで基準階の空調機の外気導入等の制御量を基準階の階数倍する場合に使用する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

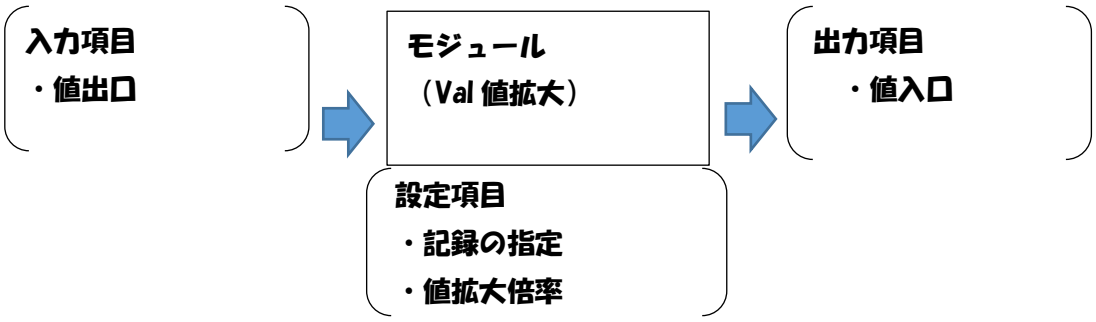


図１　モジュールの設定項目・入力項目・出力項目

（４）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	値拡大倍率	double	enlarge	1	[-]	ー	0		←入口の値を拡大します　倍率を入力してください
2	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（５）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号 今は、このモジュールの動作には影響しない
3	値入口	L0_valIn	valIn	-	状態	値	入口	値の入口
4	値出口	L0_valOut	valOut	-	状態	値	出口	値の出口

（6）記録項目（EnVal_）

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	－	メッセージ
2	入口値	値	－	入口
3	出口値	値	－	出口

（7）計算フロー・計算内容

入口接続ノードからの値にユーザー入力の拡大倍率を乗じたものを出口接続ノードの値として渡す。

valOut 出口の値[-]　＝　valIn 入口の値[-]　×　値拡大倍率

その他は入口と同じ状態値とする。

valOut 出口のデフォルト値[-]　＝　valIn 入口のデフォルト値[-]

など。

（8）データ範囲と範囲外の取扱い

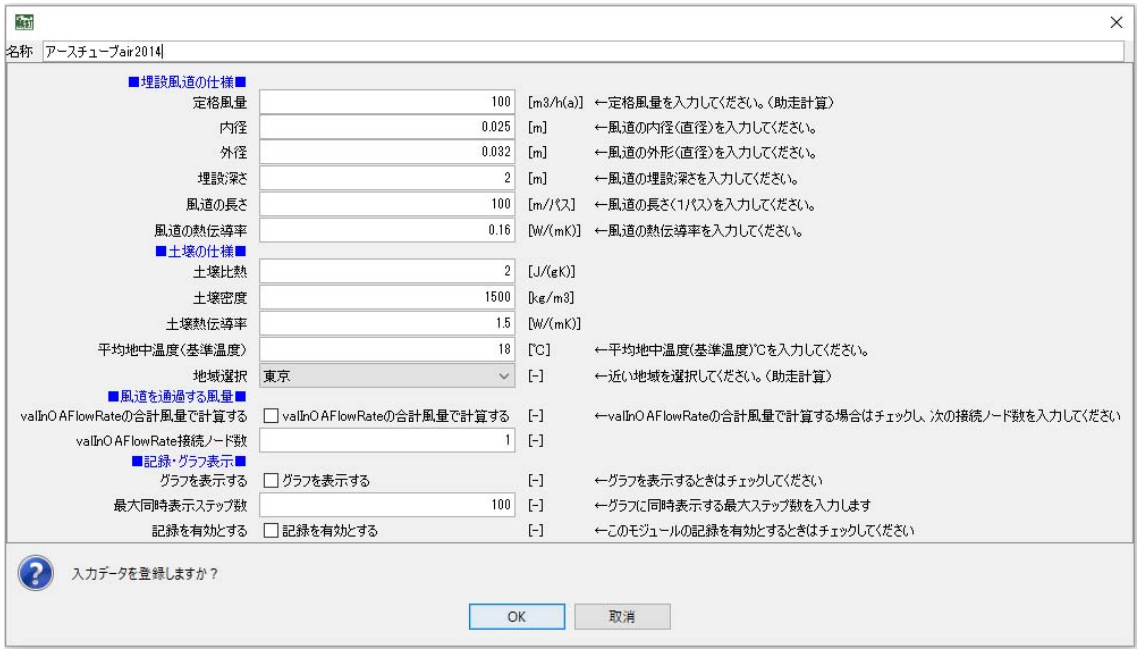
特になし。

「アースチューブ」（場所：設備 2015／ダクト配管 2015）

モジュール名	アースチューブ air2014
クラス	EarthTube2014

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、地中に水平に埋設された円管内に外気を通した場合の円管出口の空気温湿度を計算する。導入外気の予冷・予熱のために用いられることを想定している。1 本の配管が単独で地中に埋設されたものとし、複数の管が近接した場合や地表面のごく近くに埋設された場合への適用はできない。

（3）モジュールの入出力

アースチューブの入口空気状態（温湿度・風量）を入力とし、出口空気状態（温湿度・風量）を出力とする。入力の風量については、入口空気（BestAir クラス）が持つ風量情報の他に他のモジュールが出力する double 値（valInOAFlowRate）を用いることもできる。スペック情報としては、配管形状、埋設深さ、土壌物性等の他に、助走計算時の気象条件、アースチューブ内の風量についても指定する。

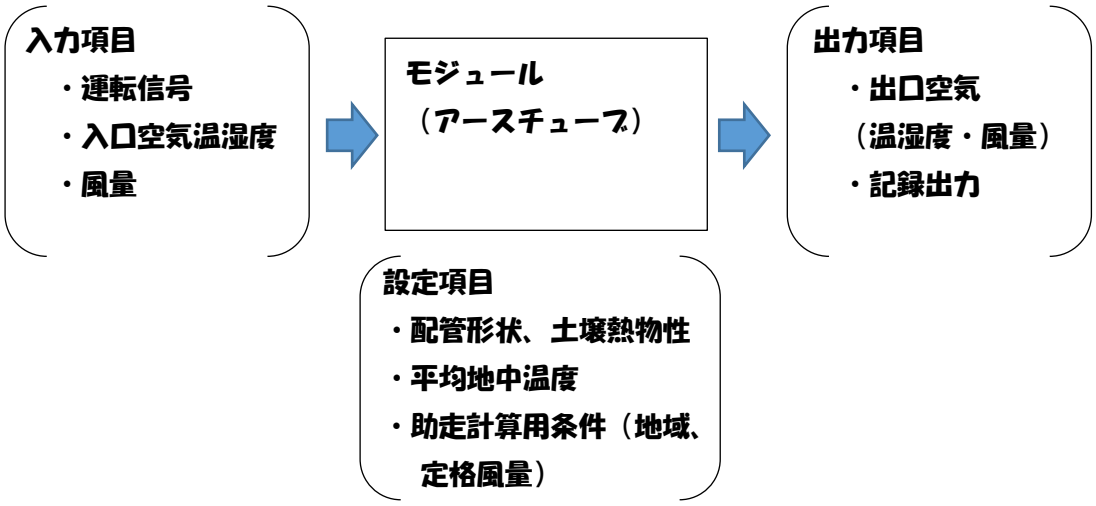


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
①	■埋設風道の仕様■								
1	定格風量	double	maxVoa_in	100	[m3/h(a)]	9999999	0	○	定格風量（助走計算用）
2	内径	double	d_0	0.025	[m]	5	0	◎	風道の内径（直径）を入力
3	外径	double	d_1	0.032	[m]	5	0	◎	風道の外径（直径）を入力
4	埋設深さ	double	D	2	[m]	1000	0	◎	地表から配管中央までの深さ
5	風道の長さ	double	L	100	[m]	10000	0	◎	
6	風道の熱伝導率	double	λ_d	0.16	[W/(m･K)]	10000	0	○	
②	■土壌の仕様■								
7	土壌比熱	double	$C_{p,s}$	2	[J/(g･K)]	―	0	○	
8	土壌密度	double	ρ_s	1500	[kg/m3]	―	0	○	
9	土壌熱伝導率	double	λ_s	1.5	[W/(m･K)]	10000	0	○	
10	平均地中温度（基準温度）	double	T_{ref}	18	[℃]	1000	-50	◎	平均地中温度（基準温度）を入力
11	地域選択	String	areaName	東京	[-]	―	―	○	近い地域を選択（助走計算用）
③	■風道を通過する風量■								
12	valInOAFlowRate の合計風量で計算する	Boolean	isvalInOAFlowRate	FALSE(チェック無し)	[-]	―	―	◎	valInOAFlowRate の合計風量で計算する場合はチェックし、次の接続ノード数を入力
13	valInOAFlowRate 接続ノード数	Int	numberOfvalInOAFlowRate	1	[-]	―	0	◎	
⑧	■記録・グラフ表示■								
14	グラフを表示する	boolean	isGVisible	FALSE(チェック無し)	[-]	―	―	○	グラフを表示するときはチェック
15	最大同時表示ステップ数	Int	maxItemCount	100	[-]	―	―	○	グラフに同時表示する最大ステップ数を入力
16	記録を有効とする	boolean	isRecord	FALSE(チェック無し)	[-]	―	―	○	このモジュールの記録を有効とするときはチェック

（5）シーケンス接続（図1における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	運転状態	L1_swcIn	SWC		制御	On/Off 信号	入口	空調制御モジュール等から On/Off 信号を受け取る。
2	チューブ入口空気	L0_airIn	DB_{oa_in} XG_{oa_in} GA_{oa_in}	[℃] [g/g] [g/s]	状態	空気	入口	この端子の風量情報（GA_{oa_in}）を使うかどうかは Spec での選択による
3	チューブ出口空気	L0_airOut	DB_{oa_out} XG_{oa_out} GA_{oa_out}	[℃] [g/g] [g/s]	状態	空気	出口	
4	入口風量	L0_valInOAFlowRate[0]	GA_{oa_in}		状態	Double 値	入口	この端子の風量情報（GA_{oa_in}）を使うかどうかは Spec での選択による
5	記録	L2_recOut	REC		記録	メモリ	出口	空調記録モジュールへ運転状態等を出力する。

（6）記録項目

No.	項目	出力内容	単位	分類
1	メッセージ	メッセージ	–	メッセージ
2	入口乾球温度	温度	℃	入口
3	入口絶対湿度	絶対湿度	g/g	入口
4	出口乾球温度	温度	℃	出口
5	出口絶対湿度	絶対湿度	g/g	出口
6	出口質量流量	流量	g/s	出口
7	交換熱量	熱量	W	負荷
8	チューブ内空気温度分布	温度	℃	My

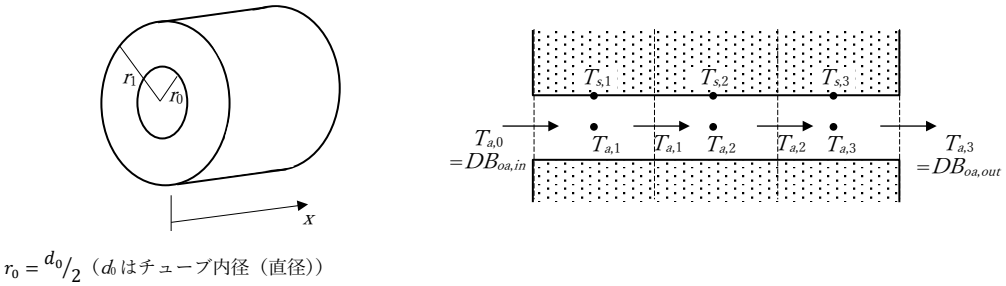
（7）計算フロー・計算内容

a) 概要

アースチューブモジュールは、永田の計算法 [1), 2)] をもとに、固定公比法による応答係数を算出し、地盤との熱交換量を求めつつ、流体の流れ方向の温度変化を算出するものである。このモデルの適用条件・仮定を以下に挙げる。

- 埋設管は十分な深さに水平に設置され、地表面温度の影響は定常成分のみとする（地表面温度は基準温度＝平均地中温度で年間固定）
- 地中の熱移動は埋設管の径方向のみとする
- 管材の熱容量は無視する
- 隣接する別の埋設管の影響は無視する

これらの仮定のもとでは、土壌および埋設管を図 2（左）のような、外周が地表面で内周をチューブ内表面とするような均質な材料からなる円筒の計算モデルで近似でき [1)]、さらに管長方向を適当な数（ここでは 10）で区分することにより、区分ごとの管内流体温度を求める（図 2（右））。なお、管材の熱抵抗と表面熱伝達抵抗の合計値を改めて表面熱伝達抵抗値と置くことにより管材の存在は無視する。



$r_0 = {}^{d_0}/_2$（d_0 はチューブ内径（直径））

図 2　アースチューブ・土壌の解析モデル

各区分の固体部分は 2 次元伝熱（回転体固体として考えれば 1 次元伝熱）を仮定し、区分ごとの管内表面熱流励振に対する管内表面温度の応答係数を求める（応答係数は異なる区分で共通）。畳み込みにおける状態変数は区分ごとに保存し、流体温度（$T_{a,1}$, $T_{a,2}$,…）および表面温度（$T_{s,1}$, $T_{s,2}$,…）を未知数とし、区分ごとに固体・管内表面・流体に関する熱収支式を立て、順次空気温度を求める。各区分の応答係数は、永田の示す吸熱伝達関数 G_a（ラプラス領域）から固定公比法により求める。

b) 前処理

i）ステップ応答の係数の算出

永田 [1), 2)] によれば、円筒形固体の内表面熱流[W/m]（単位配管長あたり）を励振とし、内表面温度変化を応答とする周波数伝達関数は、

$$H_A(\omega) = \left(2\pi\lambda_s G_A(\omega)\right)^{-1} = (X + iY)/2\pi\lambda_s \tag{1}$$

と近似できる。ただし、ωは角周波数[rad/s]であり、$2\pi\ \lambda_s G_A(\omega)$は、内表面温度を励振とし、内表面熱流[W/m]を応答とする周波数伝達関数である。また、

$$X = \frac{1}{2}\log\left[1 + \frac{1 + \sqrt{2(\zeta + \xi)}}{\zeta}\right] \tag{2}$$

$$Y = -\tan^{-1}\frac{\sqrt{\zeta - \xi}}{\sqrt{2}\zeta + \sqrt{\zeta + \xi}} \tag{3}$$

さらに、 $\xi = \left(\frac{r_0}{r_1 - r_0}\right)^2$, $\zeta = \sqrt{\xi^2 + \eta^2}$, $\eta = \frac{\omega r_0^2}{a}$, $a = \frac{\lambda_s}{C_{p,s}\rho_s}$。ここで、$r_1$は、円筒形個体の外径（半径）[m]であるが、管の埋設深さ（地表面から菅中央までの深さ）をD[m]として、$r_1 = 2D$とする。

一方、この応答のステップ応答$h(t)$（実時間）を以下で近似する。

$$h(t) = A_0 + \sum_{k=1}^{N} A_k e^{-\alpha_k t} \tag{4}$$

上式のラプラス変換は、

$$\tilde{h}(s) = \frac{A_0}{s} + \sum_{k=1}^{N}\frac{A_k}{s + \alpha_k} \tag{5}$$

よって、この応答の伝達関数（インパルス応答のラプラス変換）は、

$$H(s) = s\tilde{h}(s) = A_0 + \sum_{k=1}^{N}\frac{sA_k}{s + \alpha_k} \tag{6}$$

周波数伝達関数は、式(6)のsを$i\omega$に置き換えたものとなる（iは虚数単位）。

$$H(\omega) = A_0 + \sum_{k=1}^{N}\frac{i\omega A_k}{i\omega + \alpha_k} \tag{7}$$

式(7)のうち、右辺のΣ内の各項は、

$$\frac{i\omega A_k}{i\omega + \alpha_k} = \frac{i(i\omega - \alpha_k)\omega A_k}{(i\omega + \alpha_k)(i\omega - \alpha_k)} = \frac{(i^2\omega - i\alpha_k)\omega A_k}{i^2\omega^2 - \alpha_k^2} = \frac{(\omega + i\alpha_k)\omega A_k}{\omega^2 + \alpha_k^2} \tag{8}$$

となるので、式(1)と式(7)を等しく置くことにより、式(8)を参照して、

$$X/2\pi\lambda = R_e\left(H(\omega)\right) = A_0 + \sum_{k=1}^{N}\frac{\omega^2 A_k}{\omega^2 + \alpha_k^2} \tag{9}$$

$$Y/2\pi\lambda = I_m\left(G(\omega)\right) = \sum_{k=1}^{N}\frac{\omega\alpha_k A_k}{\omega^2 + \alpha_k^2} \tag{10}$$

ここで、

$$X' = X/2\pi\lambda - A_0 \tag{11}$$

$$Y' = Y/2\pi\lambda \tag{12}$$

と置き換えると、式(9), (10)は、

$$X' = \sum_{k=1}^{N}\left(\frac{\omega^2}{\omega^2+\alpha_k^2}\right)A_k \tag{13}$$

$$Y' = \sum_{k=1}^{N}\left(\frac{\omega\alpha_k}{\omega^2+\alpha_k^2}\right)A_k \tag{14}$$

となる。α_kとして、適当な数値を仮定して固定すれば（固定公比法)、式(13), (14)は、A_k（k=1, 2,…, N）を未知数とする線形回帰式と見なすことができる。適当な数の角周波数ωに対して、「観測値」X', Y'を式(2), (3), (11), (12)より算出し、またA_kの各係数を算出すれば、線形最小二乗法により未知パラメータA_kを決定することができる。

α_kおよびωの値として、ここでは周期 T_k（k=1, 2,…, N）を、0.09375h を初項とし、公比 4、項数 N=10 の等比数列として設定し（T_N=24576h)、

$$\alpha_k = 1/T_k \tag{15}$$

$$\omega_k = 2\pi/T_k \quad (k=1, 2,\cdots, N) \tag{16}$$

とする。

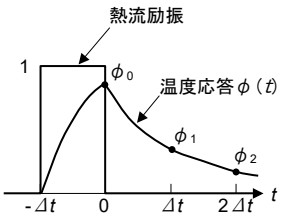
なお、式(11)の算出において必要となる A_0 の値は、この応答の定常成分なので、式(1)においてω=0 とおくことで、

$$A_0 = \frac{\log(r_1/r_0)}{2\pi\lambda} \tag{17}$$

と求める。

ii) 応答係数の算出

前項の手順で、ステップ応答（式 4）の係数、α_k, A_k（k=1, 2,…, N）が求められた。これらの係数をもとに、二等辺三角波､あるいは四角波の励振による応答係数を求める｡以下では、四角波の場合の応答係数の算出について示す。



右図のような励振・応答関係にある場合、時刻 t における応答$\phi(t)$は、式(4)を参照して、

$$\phi(t) = h(t+\Delta t) - h(t) = \sum_{k=1}^{N} A_k\left\{e^{-\alpha_k(t+\Delta t)} - e^{-\alpha_k t}\right\} \tag{19}$$

よって、瞬時応答係数 ϕ_0 は、

$$\phi_0 = \sum_{k=1}^{N} A_k \left(e^{-\alpha_k \Delta t} - 1 \right) \tag{20}$$

また、ステップ j の応答係数 ϕ_j は、

$$\begin{aligned} \phi_j = \phi(j\Delta t) &= \sum_{k=1}^{N} A_k \left\{ e^{-\alpha_k (j+1)\Delta t} - e^{-\alpha_k j \Delta t} \right\} \\ &= \sum_{k=1}^{N} e^{-\alpha_k (j-1)\Delta t} A_k \left(e^{-\alpha_k 2\Delta t} - e^{-\alpha_k \Delta t} \right) = \sum_{k=1}^{N} \rho_k^{j-1} \phi_{1,k} \end{aligned} \tag{21}$$

となる。ただし、

$$\rho_k = e^{-\alpha_k \Delta t}, \quad \phi_{1,k} = A_k \left(e^{-\alpha_k 2\Delta t} - e^{-\alpha_k \Delta t} \right) \tag{22}$$

上記の下線部の記号を用いると、時刻ステップを n として、内表面熱流 $q_{l,n}$ [W/m]を励振とし、内表面温度 $T_{s,n}$ を応答とする畳み込み演算は式(21)を利用して以下の手順で実行できる。

$$\begin{aligned} T_{s,n} &= \sum_{j=0}^{\infty} \phi_j q_{l,n-j} = \phi_0 q_{l,n} + \sum_{j=1}^{\infty} q_{l,n-j} \sum_{k=1}^{N} \rho_k^{j-1} \phi_{1,k} \\ &= \phi_0 q_{l,n} + \sum_{k=1}^{N} \phi_{1,k} \sum_{j=1}^{\infty} \rho_k^{j-1} q_{l,n-j} = \phi_0 q_{l,n} + \sum_{k=1}^{N} T'_{s,n,k} \end{aligned} \tag{23}$$

$$T'_{s,n,k} = \phi_{1,k} \sum_{j=1}^{\infty} \rho_k^{j-1} q_{l,n-j} = \phi_{1,k} q_{l,n-1} + \phi_{1,k} \sum_{j=2}^{\infty} \rho_k^{j-1} q_{l,n-j}$$

$$= \phi_{1,k} q_{l,n-1} + \rho_k \left(\phi_{1,k} \sum_{j=1}^{\infty} \rho_k^{j-1} q_{l,(n-1)-j} \right) = \phi_{1,k} q_{l,n-1} + \rho_k T'_{s,n-1,k} \tag{24}$$

iii）配管長方向の分割

上記(1), (2)によって決定される応答係数を持つような、長さ⊿x[m]の円筒形の断片 N_x 個を考え（1 ページ目の図を参照)、各断片について、式(24)の項別の出力 $T'_{s,n,k}$[℃]を初期化（=0）する（$\Delta x = L/N_x$)。

c) Time Loop 処理

各時刻ステップにおいて、V:流量[m³/s]（= $GAoa\_in$[g/s] ／1200g/m³)、$T_{a,in}$（= $DB_{oa,in}$) 入口空気温度[℃]を入力とし、$T_{a,out}$（= $DB_{oa,out}$)：出口空気温度[℃]を出力とするような下記の計算処理を行う。熱伝達率、流量、入口空気温度は、いずれも時刻によって変化しうる。

i）各区分における流体・土壌の熱収支

区分 i（i =1, 2,…, N_x）における、円筒形固体（土壌）および管内の流体の間に成り立つ熱収支式は以下となる（式(25)は、式(23)の再掲)。

$$T_{s,n,i} = \phi_0 q_{l,n,i} + T_{st,n,i} \qquad \left(T_{st,n,i} \equiv \sum_{k=1}^{N} T'_{s,n,i,k} \right) \tag{25}$$

$$q_{l,n,i} = 2\pi r_0 h \left(T_{a,i} - T_{s,n,i} \right) \tag{26}$$

$$q_{l,n,i} \Delta x = c\rho_a V \left(T_{a,i-1} - T_{a,i} \right) \tag{27}$$

ただし、$c\rho_a$は空気の容積比熱（＝1200J/(m³･K)）、$T_{a,i}$は、現在時刻における区分 iの空気温度[℃]を表わす。また、h：管表面熱伝達率[W/(m²･K)]であり、管材の熱抵抗を加味した値として以下より算出する。

$$h = \left\{ \frac{1}{h_c} + \frac{d_0}{2\lambda_d} \ln \frac{d_1}{d_0} \right\} \tag{28}$$

ここで h_cは管表面の対流熱伝達率[W/(m²･K)]であり、ユルゲスの式より、

$$h_c = 5.6 + 3.9v \quad （v \le 4.9 \text{ m/s}） \tag{29}$$

$$h_c = 7.2v^{0.78} \quad （v > 4.9 \text{ m/s}） \tag{30}$$

ここで風速 v[m/s]は風量とチューブ断面積より算出する。式(25)を式(26)に代入して、$q_{l,n,i}$について解くと、

$$q_{l,n,i} = \frac{2\pi r_0 h \left(T_{a,i} - T_{st,n,i} \right)}{1 + 2\pi r_0 h \phi_0} \tag{31}$$

式(31)を式(27)に代入して、区分 iの空気温度（＝区分 iの出口温度） $T_{a,i}$について解くと、

$$T_{a,i} = \frac{aT_{st,n,i} + bT_{a,i-1}}{a + b} \tag{32}$$

ただし、$a = 2\pi r_0 h \Delta x$, $b = c\rho_a V (1 + 2\pi r_0 h \phi_0)$

ii）アースチューブ出口温度の計算

最初に、アースチューブ入口温度から基準温度（平均地中温度）を差し引いたものを区分 1 に対する入力空気温度 $T_{a,in}$（＝$DB_{oa,in}$）とする。

$$T_{a,0} = T_{a,in} - T_{ref} \tag{33}$$

次に、各区分（i=1, 2,…, N_x）に対して、順次式(32)の計算を行い、区分 N_xの空気温度 $T_{a,Nx}$に基準温度 T_{ref}を足したものを、アースチューブの出口温度 $T_{a,out}$（＝$DB_{oa,out}$）とする。また、各区分において、式(27)より表面熱流 $q_{l,n,i}$を算出した上で、式(24)により項別の温度成分 $T'_{s,n,i,k}$を更新する。

d) 土壌用の予備（助走）計算

アースチューブを含めた計算では、本来、土壌及の初期状態を計算するためにシステム全体で助走計算を行う必要があるが、土壌の大きな熱容量を考えると、アースチューブのみのためにシステム全体で数年にわたる長期の助走計算を行うことは現実的ではない。そのため簡易的に、計算開始後の 1 ステップ目においてアースチューブモジュール内部で単独で予備（助走）計算を行うこととしている。

計算方法としては、上記 c)Time Loop 処理と同じであるが、流入空気温度 $DB_{oa,in}$ は助走計算用に選択した 12 地域（旭川、札幌、盛岡、仙台、富山、前橋、東京、静岡、名古屋、大阪、鹿児島、那覇）のいずれかの外気温を用いる。また、風量については定格風量 maxVoa_in を用い、9:00～18:00 の間のみ導入する。計算時間間隔は 1h とする。予備（助走）計算の条件を表 1 に示す。

表 1　予備（助走）計算条件

項目	条件	備考
アースチューブの仕様	入力条件と同じ	土壌の物性、平均地中温度も同じ
流入空気温度	12 地域	助走計算用にスペック入力された地点の気象データを使用
流入空気風量	定格風量	助走計算用のスペック入力値を使用
運転時間	9～18 時（間欠運転）	シーケンス接続された SWC の値に従う
計算時間間隔	1h	指定された計算時間間隔に従う

引用文献

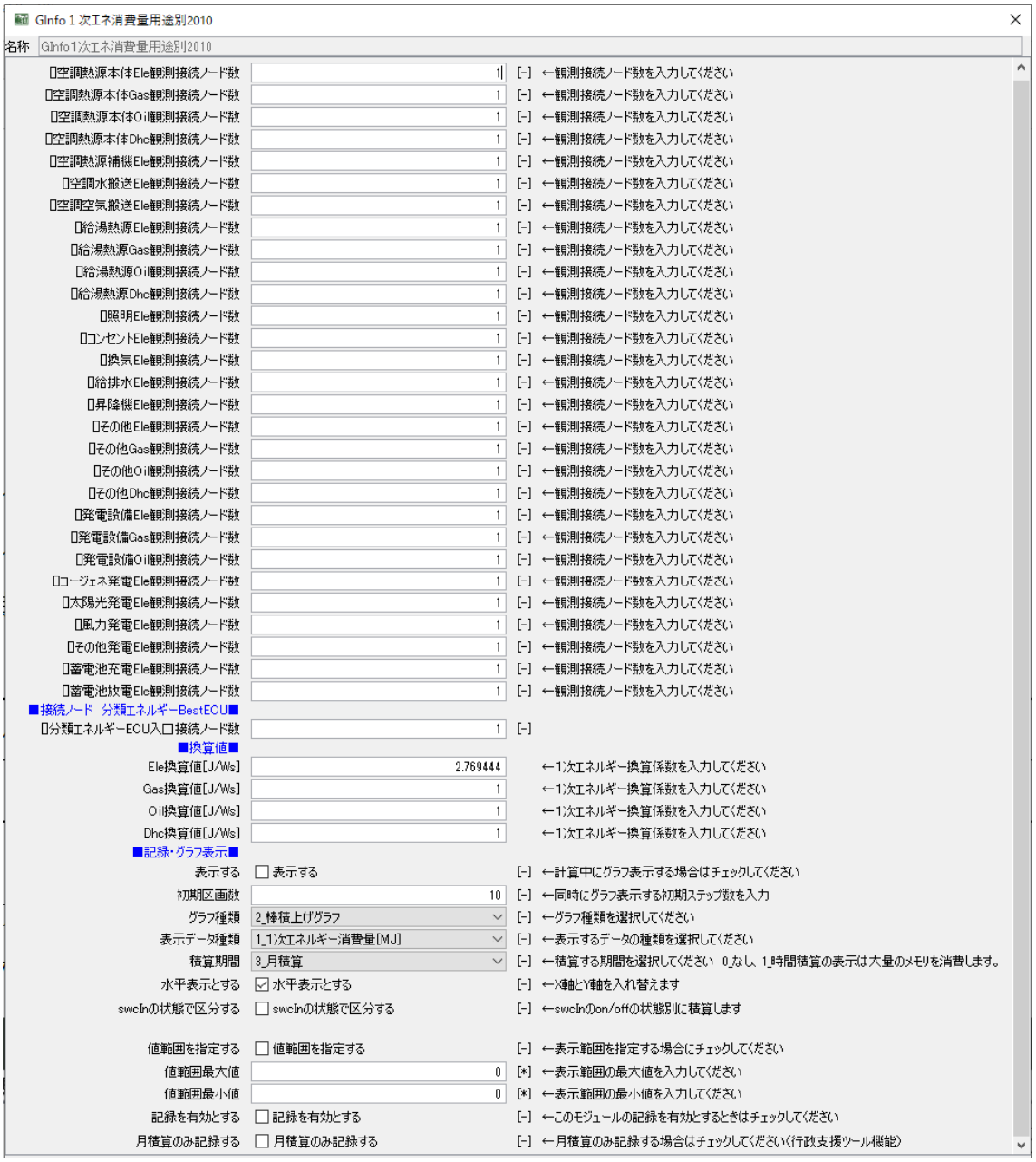
1） 社団法人 日本建築学会「見る・使う・学ぶ 環境建築」pp.70-73, 2011.5
2） 永田明寛：地中埋設管の熱応答に関する考察と近似式の提案, 日本建築学会大会学術講演梗概集 環境 II, pp.347-348, 2011.8

「Ginfo　1 次エネ消費量用途別 2010」（場所：設備 2015／グラフ　計測 2015）

モジュール名	Ginfo　1 次エネ消費量用途別 2010
クラス	GraphRealtimeConsumptionBarPrimaryEnergyUseModule20100101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、エネルギー消費先別の BestElectricity、BestGas、BestOil、BestDhc、

BestECUの媒体接続ノードを持ち、これらに接続されたエネルギー消費を分別集計処理し、記録するとともに、リアルタイムでグラフ表示する機能を持つモジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

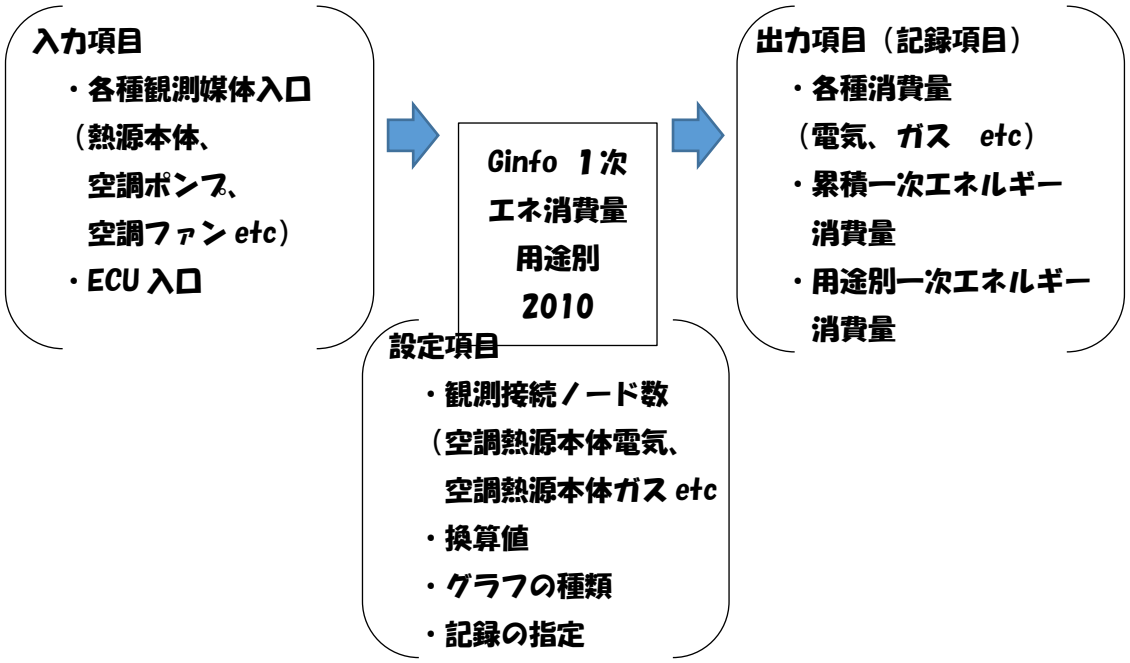


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	―	―	△	
1	[]空調熱源本体 Ele 観測接続ノード数	int	NumberOfHSmain_EleObs	1	[-]	―	―	◎	観測接続ノード数を入力
2	[]空調熱源本体 Gas 観測接続ノード数	int	NumberOfHSmain_GasObs	1	[-]	―	―	◎	〃
3	[]空調熱源本体 Oil 観測接続ノード数	int	NumberOfHSmain_OilObs	1	[-]	―	―	◎	〃
4	[]空調熱源本体 Dhc 観測接続ノード数	int	NumberOfHSmain_DhcObs	1	[-]	―	―	◎	〃
5	[]空調熱源補機 Ele 観測接続ノード数	int	NumberOfHSsub_EleObs	1	[-]	―	―	◎	〃
6	[]空調水搬送 Ele 観測接続ノード数	int	NumberOfACpump_EleObs	1	[-]	―	―	◎	〃
7	[]空調空気搬送 Ele 観測接続ノード数	int	NumberOfACfan_EleObs	1	[-]	―	―	◎	〃
8	[]給湯熱源 Ele 観測接続ノード数	int	NumberOfHWHS_EleObs	1	[-]	―	―	◎	〃
9	[]給湯熱源 Gas 観測接続ノード数	int	NumberOfHWHS_GasObs	1	[-]	―	―	◎	〃
10	[]給湯熱源 Oil 観測接続ノード数	int	NumberOfHWHS_OilObs	1	[-]	―	―	◎	〃
11	[]給湯熱源 Dhc 観測接続ノード数	int	NumberOfHWHS_DhcObs	1	[-]	―	―	◎	〃
12	[]照明 Ele 観測接続ノード数	int	NumberOfLighting_EleObs	1	[-]	―	―	◎	〃
13	[]コンセント Ele 観測接続ノード数	int	NumberOfConcent_EleObs	1	[-]	―	―	◎	〃
14	[]換気 Ele 観測接続ノード数	int	NumberOfVentilation_EleObs	1	[-]	―	―	◎	〃

15	[]給排水 Ele 観測接続ノード数	int	NumberOfWaterSupplyDrain_EleObs	1	[-]	ー	ー	◎	〃
16	[]昇降機 Ele 観測接続ノード数	int	NumberOfEV_EleObs	1	[-]	ー	ー	◎	〃
17	[]その他 Ele 観測接続ノード数	int	NumberOfOther_EleObs	1	[-]	ー	ー	◎	〃
18	[]その他 Gas 観測接続ノード数	int	NumberOfOther_GasObs	1	[-]	ー	ー	◎	〃
19	[]その他 Oil 観測接続ノード数	int	NumberOfOther_OilObs	1	[-]	ー	ー	◎	〃
20	[]その他 Dhc 観測接続ノード数	int	NumberOfOther_DhcObs	1	[-]	ー	ー	◎	〃
21	[]発電設備 Ele 観測接続ノード数	int	NumberOfCGS_eleObs	1	[-]	ー	ー	◎	〃
22	[]発電設備 Gas 観測接続ノード数	int	NumberOfCGS_gasObs	1	[-]	ー	ー	◎	〃
23	[]発電設備 Oil 観測接続ノード数	int	NumberOfCGS_oilObs	1	[-]	ー	ー	◎	〃
24	[]コージェネ発電 Ele 観測接続ノード数	int	NumberOfeleObsGenCGS	1	[-]	ー	ー	◎	〃
25	[]太陽光発電 Ele 観測接続ノード数	int	NumberOfeleObsGenSOL	1	[-]	ー	ー	◎	〃
26	[]風力発電 Ele 観測接続ノード数	int	NumberOfeleObsGenWIN	1	[-]	ー	ー	◎	〃
27	[]その他発電 Ele 観測接続ノード数	int	NumberOfeleObsGenXXX	1	[-]	ー	ー	◎	〃
28	[]蓄電池充電 Ele 観測接続ノード数	int	NumberOfBAT_eleObs	1	[-]	ー	ー	◎	〃
29	[]蓄電池放電 Ele 観測接続ノード数	int	NumberOfeleObsGenBAT	1	[-]	ー	ー	◎	〃
①	■接続ノード　分類エネルギーBestECU■								
1	[]分類エネルギーECU 入口接続ノード数	int	NumberOfecuIn	1	[-]	ー	ー	◎	ecu 接続ノード数と（エネルギーへの倍率）を入力

②	■換算値■								
1	Ele 換算値[J/Ws]	double	primaryEnergyHSmain_Ele	2.769444	[J/Ws]	ー	ー	○	１次エネルギー換算係数を入力
2	Gas 換算値[J/Ws]	double	primaryEnergyHSmain_Gas	1	[J/Ws]	ー	ー	△	〃
3	Oil 換算値[J/Ws]	double	primaryEnergyHSmain_Oil	1	[J/Ws]	ー	ー	△	〃
4	Dhc 換算値[J/Ws]	double	primaryEnergyHSmain_Dhc	1	[J/Ws]	ー	ー	△	〃
③	■記録・グラフ表示■								
1	グラフを表示する	boolean	Visible	FALSE(チェック無し)	[-]	ー	ー	○	計算中にグラフを表示するときはチェック
2	初期区画数	int	div	10	[-]	ー	ー	○	同時にグラフ表示する初期ステップ数を入力
3	グラフ種類	String 1_棒積上げ３Ｄグラフ、 2_棒積上げグラフ、 3_折れ線グラフ、 4_折れ線３Ｄグラフ、 5_棒３Ｄグラフ、 6_棒グラフ、 7_面グラフ、 8_Waterfall	graphType	2_棒積上げグラフ	[-]	ー	ー	○	このモジュールの記録を有効とするときはチェック
4	表示データ種類	String 1_１次エネルギー消費量[MJ]、 2_１次エネルギー消費	dataType	1_１次エネルギー消費量[MJ]	[-]	ー	ー		

		量（石油換算）[Mtoe]							
5	積算期間	String 0_なし、 1_時間積算、 2_日積算、 3_月積算、 4_年積算	sumHDMY	3_月積算	[-]	ー	ー		0_なし、1_時間積算の表示は大量のメモリを消費します。
6	水平表示とする	boolean	isBarH	TRUE(チェック有)	[-]	ー	ー		X 軸と Y 軸を入れ替える場合はチェック
7	swcIn の状態で区分する	boolean	isSeparateOnOff	FALSE(チェック無し)	[-]	ー	ー		swcIn の on/off の状態別に積算する場合はチェック
	値範囲を指定する	boolean	isRangeManual	FALSE(チェック無し)	[-]	ー	ー		表示範囲を指定する場合はチェック
	値範囲最大値	double	rangeMax	0	[*]	ー	ー		
	値範囲最小値	double	rangeMin	0	[*]	ー	ー		
	記録を有効とする	boolean	isRecord	FALSE(チェック無し)	[-]	ー	ー		このモジュールの記録を有効とするときはチェック
	月積算のみ記録する	boolean	isRecordMMonly	FALSE(チェック無し)	[-]	ー	ー		月積算のみを記録するときはチェック

（5）シーケンス接続（図1における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut区分	備考
1	記録	L2_recOut	REC	―	状態	空気	出口	
2	停止	L1_swcIn	swcIn	―	制御	On/Off信号	入口	
3	空調熱源本体Ele観察	L0_eleObsHSmain[0]	eleObsHSmain[]	―	状態	電気	観測	
4	空調熱源本体Gas観察	L0_gasObsHSmain[0]	gasObsHSmain[]	―	状態	ガス（燃料）	観測	
5	空調熱源本体Oil観察	L0_oilObsHSmain[0]	oilObsHSmain[]	―	状態	油（燃料）	観測	
7	空調熱源本体Dhc観察	L0_dhcObsHSmain[0]	dhcObsHSmain[]	―	状態	熱供給	観測	
8	空調熱源補機Ele観察	L0_eleObsHSsub[0]	eleObsHSsub[]	―	状態	電気	観測	
9	空調水搬送Ele観察	L0_eleObsACpump[0]	eleObsACpump[]	―	状態	電気	観測	
10	空調空気搬送Ele観察	L0_eleObsACfan[0]	eleObsACfan[]	―	状態	電気	観測	
11	給湯熱源Ele観察	L0_eleObsHWHS[0]	eleObsHWHS[]	―	状態	電気	観測	
12	給湯熱源Gas観察	L0_gasObsHWHS[0]	gasObsHWHS[]	―	状態	ガス（燃料）	観測	
13	給湯熱源Oil観察	L0_oilObsHWHS[0]	oilObsHWHS[]	―	状態	油（燃料）	観測	
14	給湯熱源Dhc観察	L0_dhcObsHWHS[0]	dhcObsHWHS[]	―	状態	熱供給	観測	
15	照明Ele観察	L0_eleObsLighting[0]	eleObsLighting[]	―	状態	電気	観測	
16	コンセントEle観察	L0_eleObsConcent[0]	eleObsConcent[]	―	状態	電気	観測	
17	換気Ele観察	L0_eleObsVentilation[0]	eleObsVentilation[]	―	状態	電気	観測	
18	給排水Ele観察	L0_eleObsWaterSupplyDrain[0]	eleObsWaterSupplyDrain[]	―	状態	電気	観測	
19	昇降機Ele観察	L0_eleObsEV[0]	eleObsEV[]	―	状態	電気	観測	
20	その他Ele観察	L0_eleObsOther[0]	eleObsOther[]	―	状態	電気	観測	
21	その他Gas観察	L0_gasObsOther[0]	gasObsOther[]	―	状態	ガス（燃料）	観測	

22	その他 Oil 観察	LO_oilObsOther[O]	oilObsOther[]	ー	状態	油（燃料）	観測	
23	その他 Dhc 観察	LO_dhcObsOther[O]	dhcObsOther[]	ー	状態	熱供給	観測	
24	発電設備用 ele 観察	LO_eleObsCGS[O]	eleObsCGS[]	ー	状態	電気	観測	
25	発電設備用 Gas 観察	LO_gasObsCGS[O]	gasObsCGS[]	ー	状態	ガス（燃料）	観測	
26	発電設備用 Oil 観察	LO_oilObsCGS[O]	oilObsCGS[]	ー	状態	油（燃料）	観測	
27	コージェネ発電 Ele 観察	LO_eleObsGenCGS[O]	eleObsGenCGS[]	ー	状態	電気	観測	
28	太陽光発電 Ele 観察	LO_eleObsGenSOL[]	eleObsGenSOL[]	ー	状態	電気	観測	
29	風力発電 Ele 観察	LO_eleObsGenWIN[]	eleObsGenWIN[]	ー	状態	電気	観測	
30	その他発電 Ele 観察	LO_eleObsGenXXX[]	eleObsGenXXX[]	ー	状態	電気	観測	
31	蓄電池充電 Ele 観察	LO_eleObsBAT[]	eleObsBAT[]	ー	状態	電気	観測	
32	蓄電池放電 Ele 観察	LO_eleObsGenBAT[]	eleObsGenBAT[]	ー	状態	電気	観測	
33	ECU 入口	LO_ecuIn[]	ecuIn[]	ー	状態	エネ消費区分	観測	

（6）記録項目

No.	項目	出力内容	単位	分類
1	一次エネルギー消費量累積電力	エネルギー	MJ	エネルギー
2	一次エネルギー消費量累積ガス	エネルギー	MJ	エネルギー
3	一次エネルギー消費量累積油	エネルギー	MJ	エネルギー
4	一次エネルギー消費量累積熱供給	エネルギー	MJ	エネルギー
5	一次エネルギー消費量累積合計	エネルギー	MJ	エネルギー
6	一次エネルギー消費量累積電力　空調熱源本体	エネルギー	MJ	エネルギー
7	一次エネルギー消費量累積ガス　空調熱源本体	エネルギー	MJ	エネルギー
8	一次エネルギー消費量累積油　空調熱源本体	エネルギー	MJ	エネルギー
9	一次エネルギー消費量累積熱供給　空調熱源本体	エネルギー	MJ	エネルギー
10	一次エネルギー消費量累積電力　空調熱源補機	エネルギー	MJ	エネルギー
11	一次エネルギー消費量累積電力　空調水搬送	エネルギー	MJ	エネルギー
12	一次エネルギー消費量累積電力　空調空気搬送	エネルギー	MJ	エネルギー
13	一次エネルギー消費量累積電力　給湯熱源	エネルギー	MJ	エネルギー
14	一次エネルギー消費量累積ガス　給湯熱源	エネルギー	MJ	エネルギー

15	一次エネルギー消費量　累積油　給湯熱源	エネルギー	MJ	エネルギー
16	一次エネルギー消費量　累積熱供給　給湯熱源	エネルギー	MJ	エネルギー
17	一次エネルギー消費量　累積電力　照明	エネルギー	MJ	エネルギー
18	一次エネルギー消費量　累積電力　コンセント	エネルギー	MJ	エネルギー
19	一次エネルギー消費量　累積電力　換気	エネルギー	MJ	エネルギー
20	一次エネルギー消費量　累積電力　給排水	エネルギー	MJ	エネルギー
21	一次エネルギー消費量　累積電力　昇降機	エネルギー	MJ	エネルギー
22	一次エネルギー消費量　累積電力　その他	エネルギー	MJ	エネルギー
23	一次エネルギー消費量　累積ガス　その他	エネルギー	MJ	エネルギー
24	一次エネルギー消費量　累積油　その他	エネルギー	MJ	エネルギー
25	一次エネルギー消費量　累積熱供給　その他	エネルギー	MJ	エネルギー
26	一次エネルギー消費量　累積電力　発電設備	エネルギー	MJ	エネルギー
27	一次エネルギー消費量　累積ガス　発電設備	エネルギー	MJ	エネルギー
28	一次エネルギー消費量　累積油　発電設備	エネルギー	MJ	エネルギー
29	一次エネルギー消費量　累積電力　コージェネ発電	エネルギー	MJ	エネルギー
30	一次エネルギー消費量　累積電力　太陽光発電	エネルギー	MJ	エネルギー
31	一次エネルギー消費量　累積電力　風力発電	エネルギー	MJ	エネルギー
32	一次エネルギー消費量	エネルギー	MJ	エネルギー

	累積電力　その他発電			
33	一次エネルギー消費量　累積電力　蓄電池充電	エネルギー	MJ	エネルギー
34	一次エネルギー消費量　累積電力　蓄電池放電	エネルギー	MJ	エネルギー
35	効率化設備 Ele 補正係数			
36	消費　電力	電気	W	
37	消費　ガス	ガス	W	
38	消費　油	油	W	
39	消費　熱供給	熱供給	W	
40	消費　合計		W	
41	消費　電力　空調熱源本体	電力	W	
42	消費　ガス　空調熱源本体	ガス	W	
43	消費　油　空調熱源本体	油	W	
44	消費　熱供給　空調熱源本体	熱供給	W	
45	消費　電力　空調熱源補機	電力	W	
46	消費　電力　空調水搬送	電力	W	
47	消費　電力　空調空気搬送	電力	W	
48	消費　電力　給湯熱源	電力	W	
49	消費　ガス　給湯熱源	ガス	W	
50	消費　油　給湯熱源	油	W	
51	消費　熱供給　給湯熱源	熱供給	W	
52	消費　電力　照明	電力	W	
53	消費　電力　コンセント	電力	W	
54	消費　電力　換気	電力	W	

55	消費　電力　給排水	電力	W	
56	消費　電力　昇降機	電力	W	
57	消費　電力　その他	電力	W	
58	消費　ガス　その他	ガス	W	
59	消費　油　その他	油	W	
60	消費　熱供給　その他	熱供給	W	
61	消費　電力　発電設備	電力	W	
62	消費　ガス　発電設備	ガス	W	
63	消費　油　発電設備	油	W	
64	消費　電力　コージェネ発電	電力	W	
65	消費　電力　太陽光発電	電力	W	
66	消費　電力　風力発電	電力	W	
67	消費　電力　その他発電	電力	W	
68	消費　電力　蓄電池充電	電力	W	
69	消費　電力　蓄電池放電	電力	W	
70				
71				

（7）計算フロー・計算内容

各種消費量（エネルギー、電気、ガス、油 等）を用途先別に分別集計し、グラフ表示する。一次エネルギーについては、各種エネルギー別（電気、ガス、油、DHC）の換算値を適用して集計する。

（8）データ範囲と適用範囲外の取扱い

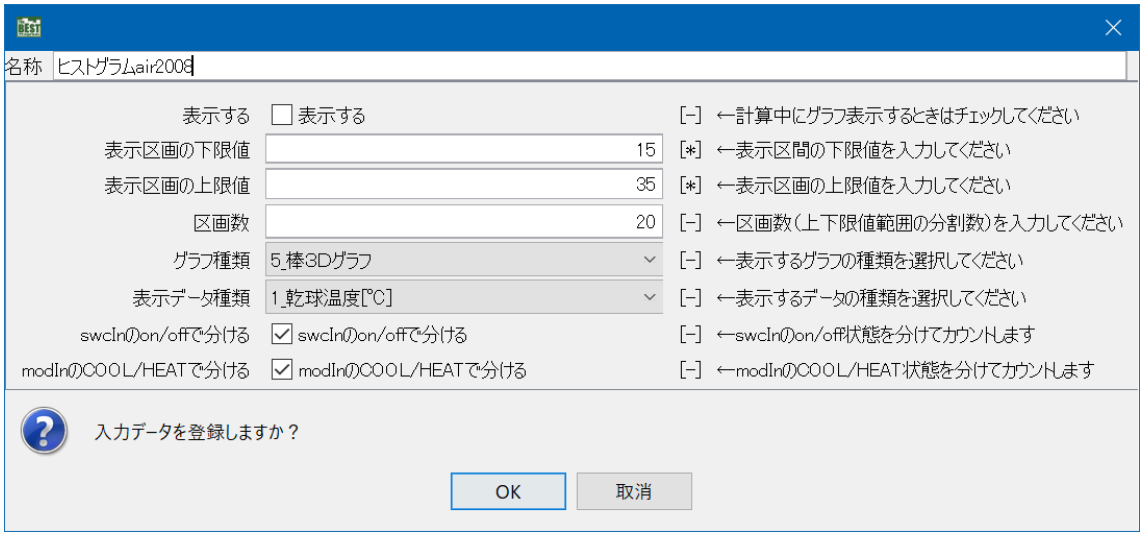
特になし。

「ヒストグラム air」（場所：設備 2015／グラフ計測 2015／）

モジュール名	ヒストグラム air
クラス	GraphRealtimeAirHistogramModule20080909 GraphRealtimeAirHistogramJFrame20080909

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、観察した BestAir 媒体の表示データ種類で指定したデータ（乾球温度など）について計算中にヒストグラムを作成するモジュールである。

空調機テンプレートなどに組み込まれたものがあり、表示するにチェックするだけで RA ヒストグラムを表示することができ、計算中に空調の良し悪しを判断することができる。

GraphRealtimeAirHistogramModule20080909 クラスがモジュールの本体で、実際にグラフを表示するのは GraphRealtimeAirHistogramJFrame20080909 クラスで行っている。

＊グラフ作成エンジンは JFreeChart を利用している。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

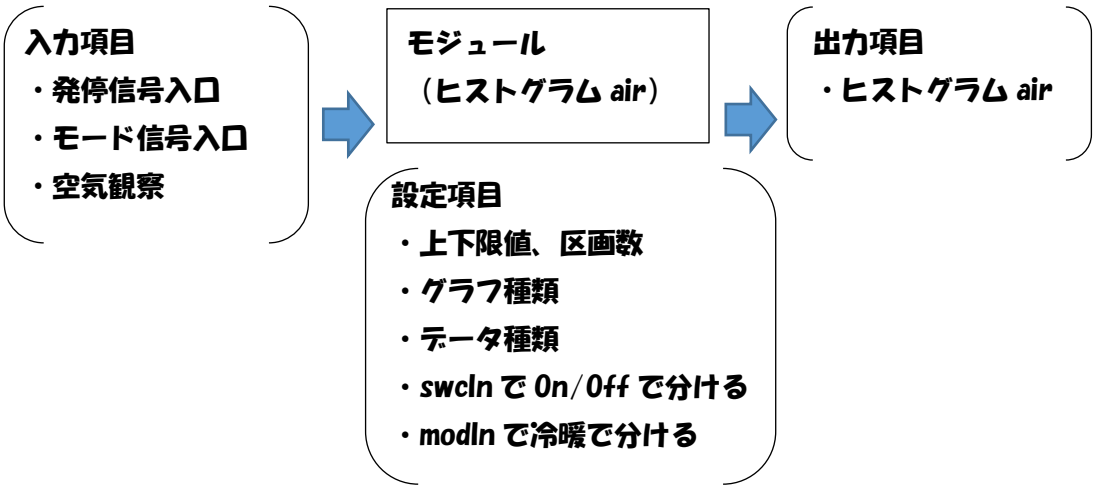


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]				
1	表示する	boolean	setVisible	FALSE	[-]				←計算中にグラフ表示するときはチェックしてください
2	表示区画の下限値	double	lowerLimit	15	[*]				←表示区間の下限値を入力してください
3	表示区画の上限値	double	upperLimit	35	[*]				←表示区画の上限値を入力してください
4	区画数	int	div	20	[-]				←区画数（上下限値範囲の分割数）を入力してください
5	グラフ種類	String	graphType	1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall	[-]				←表示するグラフの種類を選択してください
6	表示データ種類	String	dataType	1_乾球温度[℃]、2_絶対湿度[g/g']、3_質量風量[g/s]、4_湿球温度[℃]、5_相対湿度[%]、6_比エンタルピー[J/g]、7_露点温度[℃]	[-]				←表示するデータの種類を選択してください
7	swcIn の on/off で分ける	boolean	isswcIn	TRUE	[-]				←swcIn の on/off 状態を分けてカウントします
8	modIn の COOL/HEAT で分ける	boolean	ismodIn	TRUE	[-]				←modIn の COOL/HEAT 状態を分けてカウントします

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	発停信号入口	LO_swcIn	swcIn	-	制御	OnOff 信号	出口	OnOff 信号の入口
2	モード信号入口	LO_modIn	modIn	-	制御	mod 信号	入口	mod 信号の入口
3	空気観察	LO_airObs	airObs	-	状態	空気	観察	空気の観察
4								

（6）記録項目

No.	項目	出力内容	単位	分類
1				

このモジュールに記録項目はない。

（7）計算フロー・計算内容

・表示する

有効の時に、計算時にヒストグラムを表示する。

・表示区画の下限値、上限値、区画数

上限値と下限値の間を区画数で均等分割する。

これに下限値未満と上限値以上の区画を加えたもので表示する。

・グラフの種類

グラフの種類は次の８種類から選択する。

1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall

・swcln の on/off で分ける

例えば空調機への swcln を接続しておくと空調機の運転時と停止時とに分けて度数を集計する。

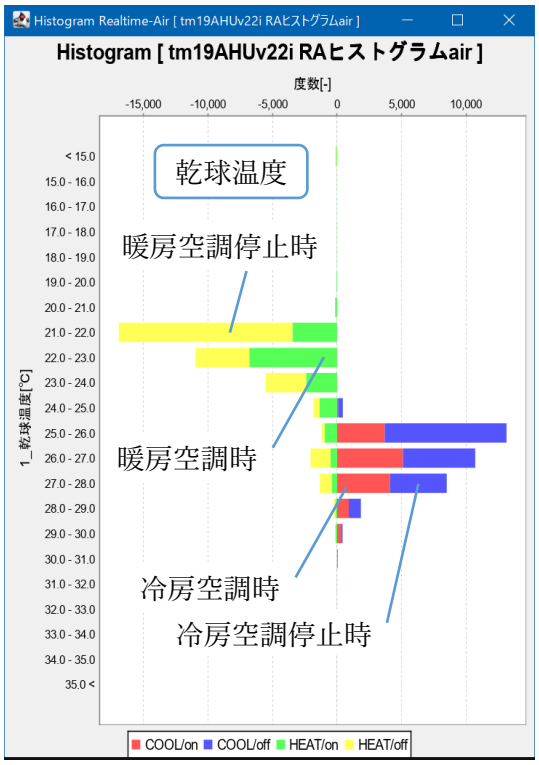
・modln の COOL/HEAT で分ける

例えば空調機への modln を接続しておくと冷房時と暖房時に分けて度数を集計する。

空調機の RA について乾球温度と相対湿度の棒積上げヒストグラムの表示例を示す。

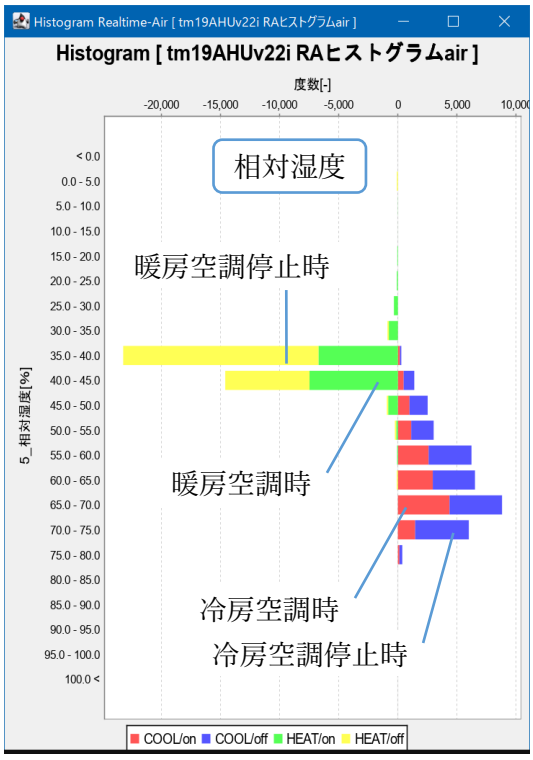
5 分間隔で年間計算（助走期間あり）したもので、度数は計算ステップ数である。

度数×計算時間間隔（5 分）÷60 で年間の時間数が求められる。



RA の乾球温度の状態を描画した例

下限値 15℃、上限値 35℃、20 区画



RA の相対湿度を描画した例

下限値 0 、上限値 100、20 区画

（８）データ範囲と範囲外の取扱い

特になし。

参考ソース

■GraphRealtimeAirHistogramModule20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.dk.test.TestCommand;

/**
 * @author HIROSHI NINOMIYA /20080826 /20080908
 * GraphRealtimeAirHistogramModule20080826 リアルタイムグラフ度数表示です
 *   BestAir  に接続する  表示データ種類として
 *     乾球温度、絶対湿度、相対湿度、質量流量、湿球温度、エンタルピー、露点温度
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 *
 * GraphRealtimeBarAitModule  を基に接続ノードを「観測/Obs」に変更
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 * isswcIn simodInを追加
 */
public class GraphRealtimeAirHistogramModule20080909 extends AbstractBestModule implements
      IBestMetaModule {

   private final String moduleName = "(GraphRealtimeAirHistogramModule20080909) ";

   //接続ノード
   //出入り口
   private final String S_NODE_airObs  = "L0_airObs"; //観測空気

   //制御ノード
   private final String C_NODE_swcIn = "L1_swcIn";
   private final String C_NODE_modIn = "L1_modIn";

   //記録ノード

   //仕様
   private final String SPEC_name      = "名称";      //外部定義項目
   private final String SPEC_Visible = "表示する"; //描画する  true、false
   private final String SPEC_lowerLimit  = "表示区画の下限値"; //表示する下限値
   private final String SPEC_upperLimit   = "表示区画の上限値";//表示する上限値
   private final String SPEC_div     = "区画数";   //片側区画数
   private final String SPEC_graphType = "グラフ種類";//グラフ種類
   private final String SPEC_dataType  = "表示データ種類";//表示するデータの種類
   //       乾球温度
   //       絶対湿度
   //       質量風量
   //       湿球温度
   //       相対湿度
   //       比エンタルピー
   //       露点温度
   private final String SPEC_isswcIn = "swcInのon/offで分ける";//on/offでカウントを分ける
   private final String SPEC_ismodIn = "modInのCOOL/HEATで分ける";//COOL・HEATでカウントを分ける
   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

   //仕様など
   private StringBuffer message = new StringBuffer();    //メッセージ
```

```
private String  name = null;  //機器名称
private boolean setVisible = true;  //描画するか　どうか
private double lowerLimit;//表示区画の下限値
private double upperLimit;//表示区画の上限値
private int div;     //区画数
private String graphType = null; //表示す売るグラフの種類
private String dataType = null;  //表示するデータの種類
private boolean isswcIn = false;//on/offでカウントを分ける
private boolean ismodIn = false;//COOL・HEATでカウントを分ける
private boolean isRecord = false;//記録を有効とする=true

//接続熱媒など
private BestAir airObs  = null;

private int swcIn;
private int modIn;

//グラフ描画データなど
private GraphRealtimeAirHistogramJFrame20080909 gAirHistogram = null;


@Override
public void setProfile(BestSpecs spec) {
   if(spec == null) {
      return;
   }

   Map<String, String> map = spec.getSpec();
   if(map == null) {
      return;
   }

   //名称を取得
   this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

   //Visibleを取得
   this.setVisible = spec.getSpecValue( this.SPEC_Visible, true );

   //lowerLimitを取得
   this.lowerLimit = spec.getSpecValue( this.SPEC_lowerLimit, 20. );

   //upperLimitを取得
   this.upperLimit = spec.getSpecValue( this.SPEC_upperLimit, 40. );

   //divを取得
   this.div = spec.getSpecValue( this.SPEC_div, 10 );

   //graphTypeを取得
   this.graphType = spec.getSpecValue( this.SPEC_graphType, "5_棒３Ｄグラフ" );

   //dataTypeを取得
   this.dataType = spec.getSpecValue( this.SPEC_dataType, "1_乾球温度[℃]" );

   //isswcIn
   this.isswcIn = spec.getSpecValue( this.SPEC_isswcIn, false );

   //ismodIn
   this.ismodIn = spec.getSpecValue( this.SPEC_ismodIn, false );

   //isRecordを取得
   this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

}


public void initialize(IBestStateMessage stateNodes,
      IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
      IBestDomainSchedule schedule) {
   //状態ノードを取得
   super.sm = stateNodes;

   //制御ノードを取得
```

```
    super.cm = commandNodes;

    //記録ノードを取得
    super.rm = recordNodes;


    //接続ノード　出口

    //接続ノード　入口チェック
    //airObs
    this.airObs = BestAir.bindnode( super.transferMapState, super.sm, this.S_NODE_airObs );

    //制御ノード

    //グラフの描画準備
    if( this.setVisible ){
        //
        gAirHistogram = new GraphRealtimeAirHistogramJFrame20080909( this.name,
              this.lowerLimit, this.upperLimit, this.div, this.graphType, this.dataType, this.isswcIn, this.ismodIn );
    }

}

public void outputs() {
    if(super.sm == null || super.cm == null ){
        return;
    }

    //
    this.airObs
    = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airObs));
    this.swcIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ));
    this.modIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn ));

    //ノードに設定

    //新しいデータを加える
    if( this.setVisible ){
        gAirHistogram.addData( this.swcIn, this.modIn, this.airObs );
    }

    //記録
    if( this.isRecord  && super.rm != null ){
        this.record();
    }

    message.setLength(0);
}

/**
 * 記録
 */
private void record(){
    if( CheckPrintModule.isPrintMessage ){
    }

    //if( CheckPrintModule.isPrintEnergy ){
    //}

    //if( CheckPrintModule.isPrintLoad ){
    //}

    //if( CheckPrintModule.isPrintStateOut ){
    //}

    //if( CheckPrintModule.isPrintStateMy ){
    //}

    //if( CheckPrintModule.isPrintStateIn ){
    //}
}
```

```
    public void update() {

    }


    public Object viewInternal(TestCommand cmd) {
        ArrayList<Object> result = new ArrayList<Object>();
        result.add(super.sm);
        result.add(super.cm);
        result.add(super.rm);

        //外部定義項目

        //記録ノードに設定する値
        if(airObs != null) {
            result.add(new Double(this.airObs.getTempDB()));      //温度
            result.add(new Double(this.airObs.getHumi()));        //絶対湿度
            result.add(new Double(this.airObs.getFlowRate()));    //風量
        }else{
            result.add(new Double(0.0));
            result.add(new Double(0.0));
            result.add(new Double(0.0));
        }

        result.add(super.transferMapCommand);
        result.add(super.transferMapRecord);
        result.add(super.transferMapState);
        return result;
    }
}
```

■GraphRealtimeAirHistogramJFrame20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.awt.BorderLayout;
import java.awt.Container;
import java.lang.reflect.InvocationTargetException;

import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

import jp.or.ibec.best.DO.BestAir;

/**
 * @author HIROSHI NINOMIYA /20080826 /20080908 /20080909
 * GraphRealtimeAirHistogramJFrame20080826 リアルタイムグラフ表示部分です
 *   BestAir  に接続する  表示データ種類として
 *     乾球温度、絶対湿度、相対湿度、質量流量、湿球温度、エンタルピー、露点温度
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 */
public class GraphRealtimeAirHistogramJFrame20080909 {

   private final String moduleName = "(GraphRealtimeAirHistogramJFrame20080909) ";

   private JFrame jFrame = null;

   private JFreeChart chart = null;

   //仕様など
   private String  name = null;  //機器名称
   private double lowerLimit;//表示区画の下限値
   private double upperLimit;//表示区画の上限値
   private double range;//区画の幅
   private int div;     //区画数
   private String graphType = null; //表示す売るグラフの種類
   private String dataType = null;  //表示するデータの種類
   private boolean isswcIn;
   private boolean ismodIn;

   //グラフ描画データなど
   private DefaultCategoryDataset dataset = null;
   private String categoryMin = null;
   private String categoryMax = null;
   private String[] category;

   private double categoryNumMin;
   private double categoryNumMax;
   private int count1Min;
   private int count1Max;
   private int[] count1;
   private int count2Min;
   private int count2Max;
   private int[] count2;
   private int count3Min;
   private int count3Max;
   private int[] count3;
   private int count4Min;
   private int count4Max;
   private int[] count4;
   double dataValue;
```

```
private boolean isReverse = false;//lowerLimit  と  upperLimitの値が逆転している=true

private BestAir air = new BestAir();


public GraphRealtimeAirHistogramJFrame20080909() {
   this( "GraphRealtimeAirHistogramJFrame", 20, 1, 20, "5_棒３Ｄグラフ", "1_乾球温度[℃]", true, true  );
}


public GraphRealtimeAirHistogramJFrame20080909( String name, double lowerLimit, double upperLimit,
      int div, String graphType, String dataType, boolean isswcIn, boolean ismodIn ) {
   this.name = name;
   this.lowerLimit = lowerLimit;
   this.upperLimit = upperLimit;
   this.div    = div;
   this.graphType = graphType;
   this.dataType  = dataType;
   this.isswcIn = isswcIn;
   this.ismodIn = ismodIn;

   createAndShowGUI();
}


public void setAir( BestAir air ){
   this.air = air;
}


 private void createAndShowGUI() {
     //Create and set up the window.
   this.jFrame = new JFrame();

   //this.jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   this.jFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

   this.jFrame.setTitle(  "Histogram Realtime-Air [ " +  this.name + " ]"  );

   Container contentpane = this.jFrame.getContentPane();

   Container panel = this.createChart();

   if( panel == null ){
      System.out.println( " panel == null ");
      System.exit(0);
   }else{
      System.out.println( " panel != null ");
   }

   contentpane.add( panel, BorderLayout.CENTER  );

   //グラフの描画準備

   this.jFrame.setSize(500,700);
     //this.jFrame.pack();
     this.jFrame.setVisible(true);
 }


 public static void main(String[] args) {
     //Schedule a job for the event-dispatching thread:
     //creating and showing this application's GUI.
   /*
   javax.swing.SwingUtilities.invokeLater( new Runnable() {
       public void run() {
           createAndShowGUI();
       }
   });
   */
 }
```

```
private JComponent createChart(){
   //
   dataset = createDataset();

   //
   if( this.graphType.equals( "8_Waterfall" )){
      this.chart = ChartFactory.createWaterfallChart(
            "Histogram [ "+this.name+" ]",    this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "1_棒積上げ３Ｄグラフ" )){
      this.chart = ChartFactory.createStackedBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "2_棒積上げグラフ" )){
      this.chart = ChartFactory.createStackedBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "3_折れ線グラフ" )){
      this.chart = ChartFactory.createLineChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "4_折れ線３Ｄグラフ" )){
      this.chart = ChartFactory.createLineChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "5_棒３Ｄグラフ" )){
      this.chart = ChartFactory.createBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "6_棒グラフ" )){
      this.chart = ChartFactory.createBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "7_面グラフ" )){
      this.chart = ChartFactory.createAreaChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else{
      System.out.println( this.moduleName + "graphType error!" );
   }

   ChartPanel cPanel = new ChartPanel( this.chart );

   return cPanel;
}


//row keys
private String series1 = null;
private String series2 = null;
private String series3 = null;
private String series4 = null;

private DefaultCategoryDataset createDataset(){
   //dataset
   this.dataset = new DefaultCategoryDataset();

   this.categoryNumMin = this.lowerLimit;
   this.categoryNumMax = this.upperLimit;
   this.range = ( this.upperLimit - this.lowerLimit ) / this.div;

   if( this.lowerLimit > this.upperLimit ){
      this.isReverse = true;
   }


   if( isswcIn ){
      if( ismodIn ){
         //row keys
         series1 = "COOL/on";
```

```
      series2 = "COOL/off";
      series3 = "HEAT/on";
      series4 = "HEAT/off";
   }else{
      //row keys
      series1 = "/on";
      series2 = "/off";
      series3 = "/on";
      series4 = "/off";
   }
}else{
   if( ismodIn ){
      //row keys
      series1 = "COOL/";
      series2 = "COOL/";
      series3 = "HEAT/";
      series4 = "HEAT/";
   }else{
      //row keys
      series1 = "/";
      series2 = "/";
      series3 = "/";
      series4 = "/";
   }
}

//column keys
if( this.isReverse ){
   this.categoryMin = "> "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" >";
}else{
   this.categoryMin = "< "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" <";
}

//count
count1Min = 0;
count1Max = 0;
count2Min = 0;
count2Max = 0;
count3Min = 0;
count3Max = 0;
count4Min = 0;
count4Max = 0;

dataset.addValue( count1Min, series1, categoryMin );
dataset.addValue( count2Min, series2, categoryMin );
dataset.addValue( count3Min, series3, categoryMin );
dataset.addValue( count4Min, series4, categoryMin );

this.category = new String[ this.div ];
this.count1 = new int[ this.div ];
this.count2 = new int[ this.div ];
this.count3 = new int[ this.div ];
this.count4 = new int[ this.div ];

//int k = 0;
for( int i= 0; i<this.div; i++ ){
   category[i] = ""+( this.categoryNumMin + this.range * ( i) )
   +" - "+( this.categoryNumMin + this.range * ( i + 1 ) );
   count1[i] = 0;
   count2[i] = 0;
   count3[i] = 0;
   count4[i] = 0;
   //
   dataset.addValue( count1[i], series1, category[i] );
   dataset.addValue( count2[i], series2, category[i] );
   dataset.addValue( count3[i], series3, category[i] );
   dataset.addValue( count4[i], series4, category[i] );
   //k++;
}
```

```
    dataset.addValue( count1Max, series1, categoryMax );
    dataset.addValue( count2Max, series2, categoryMax );
    dataset.addValue( count1Max, series3, categoryMax );
    dataset.addValue( count2Max, series4, categoryMax );

    return dataset;
}


public void addData( int swc, int mod, BestAir airP ){

    //System.out.println( moduleName +"addData()");
    //System.out.println( moduleName + "■■isEventDispatchThread() addData ="
    //+ SwingUtilities.isEventDispatchThread() );

    final boolean isOnOff;

    String str = null;
    String stc = null;
    double count = 0.1;

    this.air = airP;
    //        乾球温度
    //        絶対湿度
    //        質量風量
    //        湿球温度
    //        相対湿度
    //        比エンタルピー
    //        露点温度

    if( this.dataType.equals( "1_乾球温度[℃]")){
        this.dataValue = this.air.getTempDB();
    }else if(this.dataType.equals( "2_絶対湿度[g/g']")){
        this.dataValue = this.air.getHumi();
    }else if(this.dataType.equals( "3_質量風量[g/s]")){
        this.dataValue = this.air.getFlowRate();
    }else if(this.dataType.equals( "4_湿球温度[℃]")){
        this.dataValue = this.air.getTempWB();
    }else if(this.dataType.equals( "5_相対湿度[%]")){
        this.dataValue = this.air.getHumiR();
    }else if(this.dataType.equals( "6_比エンタルピー[J/g]")){
        this.dataValue = this.air.getEnthalpy();
    }else if(this.dataType.equals( "7_露点温度[℃]")){
        this.dataValue = this.air.getTempDP();
    }

    //
    if( isswcIn ){
        isOnOff = Airswc.isON( swc );
    }else{
        isOnOff = true;
    }

    if( ismodIn ){
    }else{
        mod = Airmod.onCOOL( mod );
    }

    if( isOnOff ){
        if( Airmod.isCOOL( mod )){
            //str = "COOL/on";
            str = this.series1;
            if( isReverse ){
                if( this.dataValue > categoryNumMin ){
                    count = ++count1Min;
                    stc = categoryMin;
                }else if( this.dataValue <= categoryNumMax ){
                    count = ++count1Max;
                    stc = categoryMax;
                }else{
                    for( int i = 0; i < div; i++ ){
                        if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
```

```
                    count = ++count1[ i ];
                    stc = category[ i ];
                    break;
                }
            }
        }
    }else{
        if( this.dataValue < categoryNumMin ){
            count = ++count1Min;
            stc = categoryMin;
        }else if( this.dataValue >= categoryNumMax ){
            count = ++count1Max;
            stc = categoryMax;
        }else{
            for( int i = 0; i < div; i++ ){
                if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                    count = ++count1[ i ];
                    stc = category[ i ];
                    break;
                }
            }
        }
    }

}else if( Airmod.isHEAT( mod )){
    //str = "HEAT/on";
    str = this.series3;
    if( isReverse ){
        if( this.dataValue > categoryNumMin ){
            count = --count3Min;
            stc = categoryMin;
        }else if( this.dataValue <= categoryNumMax ){
            count = --count3Max;
            stc = categoryMax;
        }else{
            for( int i = 0; i < div; i++ ){
                if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                    count = --count3[ i ];
                    stc = category[ i ];
                    break;
                }
            }
        }
    }else{
        if( this.dataValue < categoryNumMin ){
            count = --count3Min;
            stc = categoryMin;
        }else if( this.dataValue >= categoryNumMax ){
            count = --count3Max;
            stc = categoryMax;
        }else{
            for( int i = 0; i < div; i++ ){
                if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                    count = --count3[ i ];
                    stc = category[ i ];
                    break;
                }
            }
        }
    }

}
}else{
    if( Airmod.isCOOL( mod )){
        //str = "COOL/off";
        str = this.series2;
        if( isReverse ){
            if( this.dataValue > categoryNumMin ){
                count = ++count2Min;
                stc = categoryMin;
            }else if( this.dataValue <= categoryNumMax ){
                count = ++count2Max;
```

```
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                  count = ++count2[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }else{
         if( this.dataValue < categoryNumMin ){
            count = ++count2Min;
            stc = categoryMin;
         }else if( this.dataValue >= categoryNumMax ){
            count = ++count2Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                  count = ++count2[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }
   }else if( Airmod.isHEAT( mod )){
      //str = "HEAT/off";
      str = this.series4;
      if( isReverse ){
         if( this.dataValue > categoryNumMin ){
            count = --count4Min;
            stc = categoryMin;
         }else if( this.dataValue <= categoryNumMax ){
            count = --count4Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                  count = --count4[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }else{
         if( this.dataValue < categoryNumMin ){
            count = --count4Min;
            stc = categoryMin;
         }else if( this.dataValue >= categoryNumMax ){
            count = --count4Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                  count = --count4[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }
   }
}

final double fcount = count;
final String rowKey = str;
final String columnKey = stc;

final Runnable worker = new Runnable() {
   public void run(){
```

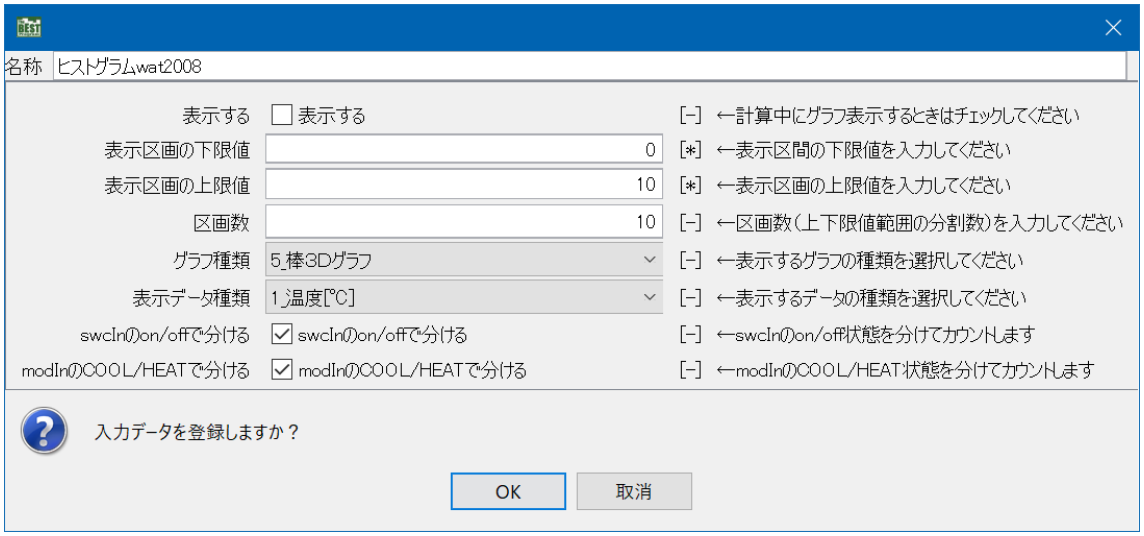
```
            //System.out.println( moduleName +"worker run()");
            //System.out.println( moduleName + "■■isEventDispatchThread() worker ="
            //+ SwingUtilities.isEventDispatchThread() );
            if( rowKey != null && columnKey != null ){
               dataset.setValue( fcount, rowKey, columnKey );
            }
         }
      };

      Thread runThread = new Thread(){
         public void run() {
            //System.out.println( moduleName +"worker runThread()");
            //System.out.println( moduleName +"■■isEventDispatchThread() runThread ="
            //+ SwingUtilities.isEventDispatchThread() );
            try {
               SwingUtilities.invokeAndWait( worker );
            } catch (InterruptedException e) {
               e.printStackTrace();
            } catch (InvocationTargetException e) {
               e.printStackTrace();
            }
         }
      };

      runThread.start();
   }
}
```

「ヒストグラム wat」（場所：設備 2015／グラフ計測 2015／）

モジュール名	ヒストグラム wat
クラス	GraphRealtimeWaterHistogramModule20080909 GraphRealtimeWaterHistogramJFrame20080909

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、観察した BestWater 媒体の表示データ種類で指定したデータ（温度など）について計算中にヒストグラムを作成するモジュールである。

熱源群テンプレートなどに組み込まれたものがあり、表示するにチェックするだけで熱源群出口のヒストグラムを表示することができ、計算中に熱源運転の良し悪しを判断することができる。

GraphRealtimeWaterHistogramModule20080909 クラスがモジュールの本体で、実際にグラフを表示するのは GraphRealtimeWaterHistogramJFrame20080909 クラスで行っている。

＊グラフ作成エンジンは JFreeChart を利用している。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

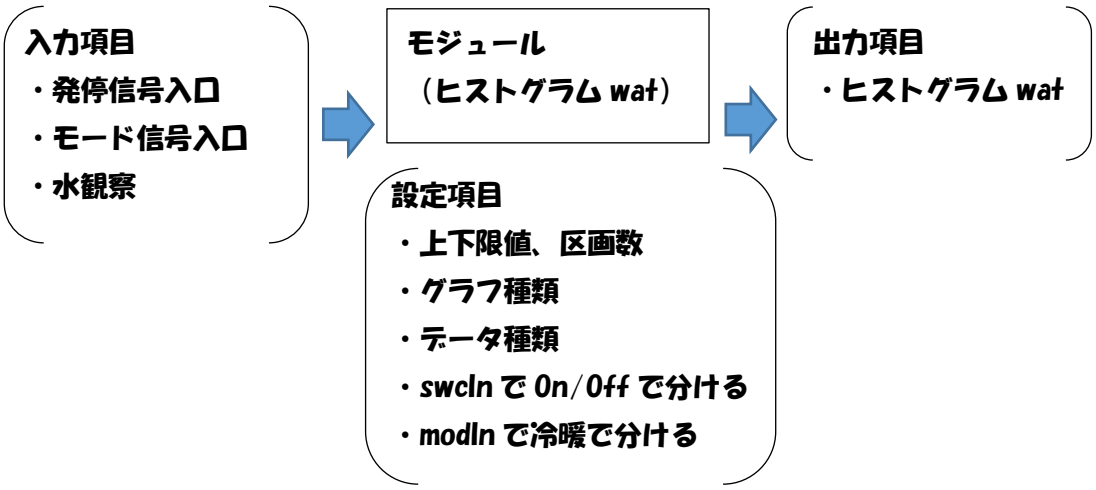


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図1における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]				
1	表示する	boolean	setVisible	FALSE	[-]				←計算中にグラフ表示するときはチェックしてください
2	表示区画の下限値	double	lowerLimit	0	[*]				←表示区間の下限値を入力してください
3	表示区画の上限値	double	upperLimit	10	[*]				←表示区画の上限値を入力してください
4	区画数	int	div	10	[-]				←区画数（上下限値範囲の分割数）を入力してください
5	グラフ種類	String	graphType	1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall	[-]				←表示するグラフの種類を選択してください
6	表示データ種類	String	dataType	1_温度[℃]、2_質量流量[g/s]	[-]				←表示するデータの種類を選択してください
7	swcIn の on/off で分ける	boolean	isswcIn	TRUE	[-]				←swcIn の on/off 状態を分けてカウントします
8	modIn の COOL/HEAT で分ける	boolean	ismodIn	TRUE	[-]				←modIn の COOL/HEAT 状態を分けてカウントします

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	発停信号入口	LO_swcIn	swcIn	-	制御	OnOff 信号	出口	OnOff 信号の入口
2	モード信号入口	LO_modIn	modIn	-	制御	mod 信号	入口	mod 信号の入口
3	空気観察	LO_watObs	watObs	-	状態	水	観察	水の観察
4								

（6）記録項目

No.	項目	出力内容	単位	分類
1				

このモジュールに記録項目はない。

（7）計算フロー・計算内容

・表示する

有効の時に、計算時にヒストグラムを表示する。

・表示区画の下限値、上限値、区画数

上限値と下限値の間を区画数で均等分割する。

これに下限値未満と上限値以上の区画を加えたもので表示する。

・グラフの種類

グラフの種類は次の８種類から選択する。

1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall

・swcln の on/off で分ける

例えば空調機への swcln を接続しておくと空調機の運転時と停止時とに分けて度数を集計する。

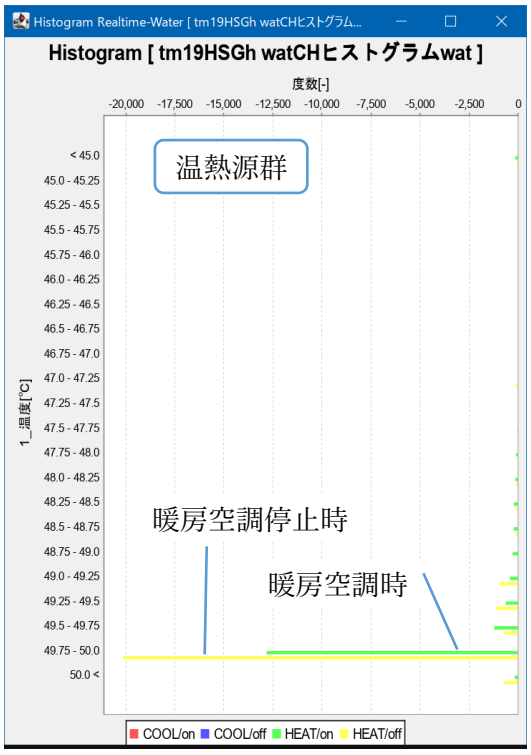
・modln の COOL/HEAT で分ける

例えば空調機への modln を接続しておくと冷房時と暖房時に分けて度数を集計する。

冷・温熱源群の出口の温度の棒ヒストグラム、棒３Ｄヒストグラムの表示例を示す。

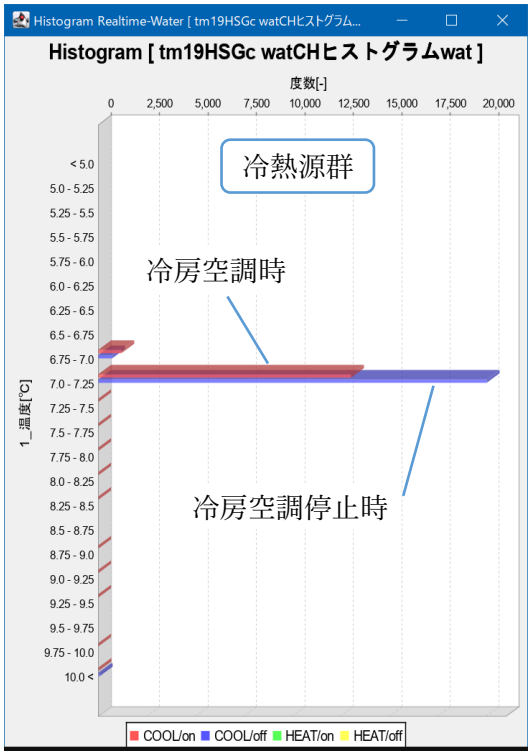
5 分間隔で年間計算（助走期間あり）したもので、度数は計算ステップ数である。

度数×計算時間間隔（5 分）÷60 で年間の時間数が求められる。



温熱源群出口水温の状態を描画した例

下限値 45℃、上限値 50℃、20 区画



冷熱源群出口水温を描画した例

下限値 5、上限値 10、20 区画

（８）データ範囲と範囲外の取扱い

特になし。

参考ソース

■GraphRealtimeWaterHistogramModule20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.Map;

import jp.or.ibec.best.DO.BestWater;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.dk.test.TestCommand;

/**
 * @author HIROSHI NINOMIYA /20080827 /20080908
 * GraphRealtimeWaterHistogramModule20080827 リアルタイムグラフ度数表示です
 *   BestWater  に接続する  表示データ種類として
 *     温度、質量流量
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 *
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 * isswcIn simodInを追加
 */
public class GraphRealtimeWaterHistogramModule20080909 extends AbstractBestModule implements
      IBestMetaModule {

   private final String moduleName = "(GraphRealtimeWaterHistogramModule20080909) ";

   //接続ノード
   //出入り口
   private final String S_NODE_watObs  = "L0_watObs"; //観測水

   //制御ノード
   private final String C_NODE_swcIn = "L1_swcIn";
   private final String C_NODE_modIn = "L1_modIn";

   //記録ノード

   //仕様
   private final String SPEC_name      = "名称";      //外部定義項目
   private final String SPEC_Visible = "表示する"; //描画する  true、false
   private final String SPEC_lowerLimit  = "表示区画の下限値"; //表示区画の下限値
   private final String SPEC_upperLimit  = "表示区画の上限値"; //表示区画の上限値
   private final String SPEC_div     = "区画数";   //区画数
   private final String SPEC_graphType = "グラフ種類";//グラフ種類
   private final String SPEC_dataType  = "表示データ種類";//表示するデータの種類
   //       温度
   //       質量流量
   private final String SPEC_isswcIn = "swcInのon/offで分ける";//on/offでカウントを分ける
   private final String SPEC_ismodIn = "modInのCOOL/HEATで分ける";//COOL・HEATでカウントを分ける
   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

   //仕様など
   private StringBuffer message = new StringBuffer();    //メッセージ
   private String  name = null;  //機器名称
   private boolean setVisible = true;  //描画するか  どうか
   private double lowerLimit;//表示区画の下限値
   private double upperLimit;//表示区画の上限値
   private int div;//区画数
   private String graphType = null; //表示す売るグラフの種類
```

```
private String dataType = null;  //表示するデータの種類
private boolean isswcIn = false;//on/offでカウントを分ける
private boolean ismodIn = false;//COOL・HEATでカウントを分ける
private boolean isRecord = false;//記録を有効とする=true

//接続熱媒など
private BestWater watObs  = null;

private int swcIn;
private int modIn;

//グラフ描画データなど
private GraphRealtimeWaterHistogramJFrame20080909 gWaterHistogram = null;


@Override
public void setProfile(BestSpecs spec) {
   if(spec == null) {
      return;
   }

   Map<String, String> map = spec.getSpec();
   if(map == null) {
      return;
   }

   //名称を取得
   this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

   //Visibleを取得
   this.setVisible = spec.getSpecValue( this.SPEC_Visible, true );

   //lowerLimitを取得
   this.lowerLimit = spec.getSpecValue( this.SPEC_lowerLimit, 0. );

   //upperLimitを取得
   this.upperLimit = spec.getSpecValue( this.SPEC_upperLimit, 10. );

   //divを取得
   this.div = spec.getSpecValue( this.SPEC_div, 10 );

   //graphTypeを取得
   this.graphType = spec.getSpecValue( this.SPEC_graphType, "5_棒３Ｄグラフ" );

   //dataTypeを取得
   this.dataType = spec.getSpecValue( this.SPEC_dataType, "1_温度[℃]" );

   //isswcIn
   this.isswcIn = spec.getSpecValue( this.SPEC_isswcIn, false );

   //ismodIn
   this.ismodIn = spec.getSpecValue( this.SPEC_ismodIn, false );

   //isRecordを取得
   this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

}


public void initialize(IBestStateMessage stateNodes,
      IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
      IBestDomainSchedule schedule) {
   //状態ノードを取得
   super.sm = stateNodes;

   //制御ノードを取得
   super.cm = commandNodes;

   //記録ノードを取得
   super.rm = recordNodes;
```

```
    //接続ノード　出口

    //接続ノード　入口チェック
    //airObs
    this.watObs = BestWater.bindnode( super.transferMapState, super.sm, this.S_NODE_watObs );

    //制御ノード

    //グラフの描画準備
    if( this.setVisible ){
        //
        gWaterHistogram = new GraphRealtimeWaterHistogramJFrame20080909( this.name,
              this.lowerLimit, this.upperLimit, this.div, this.graphType, this.dataType, this.isswcIn, this.ismodIn );
    }

}

public void outputs() {
    if(super.sm == null || super.cm == null ){
        return;
    }

    //
    this.watObs
    = (BestWater)super.sm.getState(super.getConnectionNode(this.S_NODE_watObs));
    this.swcIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ));
    this.modIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn ));

    //ノードに設定

    //新しいデータを加える
    if( this.setVisible ){
        gWaterHistogram.addData( this.swcIn, this.modIn, this.watObs );
    }

    //記録

    if( this.isRecord  && super.rm != null ){
        this.record();
    }

    message.setLength(0);
}

/**
 * 記録
 */
private void record(){
    if( CheckPrintModule.isPrintMessage ){
    }

    //if( CheckPrintModule.isPrintEnergy ){
    //}

    //if( CheckPrintModule.isPrintLoad ){
    //}

    //if( CheckPrintModule.isPrintStateOut ){
    //}

    //if( CheckPrintModule.isPrintStateMy ){
    //}

    //if( CheckPrintModule.isPrintStateIn ){
    //}
}

public void update() {

}
```

```
    public Object viewInternal(TestCommand cmd) {
        ArrayList<Object> result = new ArrayList<Object>();
        result.add(super.sm);
        result.add(super.cm);
        result.add(super.rm);

        //外部定義項目

        //記録ノードに設定する値
        if(watObs != null) {
            result.add(new Double(this.watObs.getTemp()));     //温度
            result.add(new Double(this.watObs.getFlowRate())); //質量流量
        }else{
            result.add(new Double(0.0));
            result.add(new Double(0.0));
        }

        result.add(super.transferMapCommand);
        result.add(super.transferMapRecord);
        result.add(super.transferMapState);
        return result;
    }
}
```

■GraphRealtimeWaterHistogramJFrame20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.awt.BorderLayout;
import java.awt.Container;
import java.lang.reflect.InvocationTargetException;

import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

import jp.or.ibec.best.DO.BestWater;

/**
 * @author HIROSHI NINOMIYA /20080827 /20080908 /20080909
 * GraphRealtimeWaterHistogramJFrame リアルタイムグラフ表示部分です
 *   BestWater  に接続する  表示データ種類として
 *     温度、質量流量
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 */
public class GraphRealtimeWaterHistogramJFrame20080909 {

   private final String moduleName = "(GraphRealtimeWaterHistogramJFrame20080909) ";

   private JFrame jFrame = null;

   private JFreeChart chart = null;

   //仕様など
   private String  name = null;  //機器名称
   private double lowerLimit;//表示区画の下限値
   private double upperLimit;//表示区画の上限値
   private double range;      //１区画の幅
   private int div;     //区画数
   private String graphType = null; //表示す売るグラフの種類
   private String dataType = null;  //表示するデータの種類
   private boolean isswcIn;
   private boolean ismodIn;

   //グラフ描画データなど
   private DefaultCategoryDataset dataset = null;
   private String categoryMin = null;
   private String categoryMax = null;
   private String[] category;

   private double categoryNumMin;
   private double categoryNumMax;
   private int count1Min;
   private int count1Max;
   private int[] count1;
   private int count2Min;
   private int count2Max;
   private int[] count2;
   private int count3Min;
   private int count3Max;
   private int[] count3;
   private int count4Min;
   private int count4Max;
   private int[] count4;
   double dataValue;
```

```
private boolean isReverse = false;//lowerLimit  とupperLimitが逆転している=true

private BestWater water = new BestWater();


public GraphRealtimeWaterHistogramJFrame20080909() {
   this( "GraphRealtimeWaterHistogramJFrame", 0, 10, 10, "5_棒３Ｄグラフ", "1_温度[℃]", true, true  );
}


public GraphRealtimeWaterHistogramJFrame20080909( String name, double lowerLimit, double upperLimit,
      int div, String graphType, String dataType, boolean isswcIn, boolean ismodIn ) {
   this.name = name;
   this.lowerLimit = lowerLimit;
   this.upperLimit = upperLimit;
   this.div    = div;
   this.graphType = graphType;
   this.dataType  = dataType;
   this.isswcIn = isswcIn;
   this.ismodIn = ismodIn;

   createAndShowGUI();
}


public void setWater( BestWater water ){
   this.water = water;
}


 private void createAndShowGUI() {
     //Create and set up the window.
   this.jFrame = new JFrame();

   //this.jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   this.jFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

   this.jFrame.setTitle(  "Histogram Realtime-Water [ " +  this.name + " ]"  );

   Container contentpane = this.jFrame.getContentPane();

   Container panel = this.createChart();

   if( panel == null ){
      System.out.println( " panel == null ");
      System.exit(0);
   }else{
      System.out.println( " panel != null ");
   }

   contentpane.add( panel, BorderLayout.CENTER  );

   //グラフの描画準備

   this.jFrame.setSize(500,700);
     //this.jFrame.pack();
     this.jFrame.setVisible(true);
 }


 public static void main(String[] args) {
     //Schedule a job for the event-dispatching thread:
     //creating and showing this application's GUI.
   /*
   javax.swing.SwingUtilities.invokeLater( new Runnable() {
       public void run() {
           createAndShowGUI();
       }
   });
   */
 }
```

```
private JComponent createChart(){
   //
   dataset = createDataset();

   //
   if( this.graphType.equals( "8_Waterfall" )){
      this.chart = ChartFactory.createWaterfallChart(
            "Histogram [ "+this.name+" ]",    this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "1_棒積上げ３Ｄグラフ" )){
      this.chart = ChartFactory.createStackedBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "2_棒積上げグラフ" )){
      this.chart = ChartFactory.createStackedBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "3_折れ線グラフ" )){
      this.chart = ChartFactory.createLineChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "4_折れ線３Ｄグラフ" )){
      this.chart = ChartFactory.createLineChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "5_棒３Ｄグラフ" )){
      this.chart = ChartFactory.createBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "6_棒グラフ" )){
      this.chart = ChartFactory.createBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "7_面グラフ" )){
      this.chart = ChartFactory.createAreaChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else{
      System.out.println( this.moduleName + "graphType error!" );
   }

   ChartPanel cPanel = new ChartPanel( this.chart );

   return cPanel;
}


//row keys
private String series1 = null;
private String series2 = null;
private String series3 = null;
private String series4 = null;

private DefaultCategoryDataset createDataset(){
   //dataset
   this.dataset = new DefaultCategoryDataset();

   this.categoryNumMin = this.lowerLimit;
   this.categoryNumMax = this.upperLimit;
   this.range = ( this.upperLimit - this.lowerLimit ) / this.div;

   if( this.lowerLimit > this.upperLimit ){
      this.isReverse = true;
   }


   if( isswcIn ){
      if( ismodIn ){
         //row keys
         series1 = "COOL/on";
```

```
      series2 = "COOL/off";
      series3 = "HEAT/on";
      series4 = "HEAT/off";
   }else{
      //row keys
      series1 = "/on";
      series2 = "/off";
      series3 = "/on";
      series4 = "/off";
   }
}else{
   if( ismodIn ){
      //row keys
      series1 = "COOL/";
      series2 = "COOL/";
      series3 = "HEAT/";
      series4 = "HEAT/";
   }else{
      //row keys
      series1 = "/";
      series2 = "/";
      series3 = "/";
      series4 = "/";
   }
}

//column keys
if( this.isReverse ){
   this.categoryMin = "> "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" >";
}else{
   this.categoryMin = "< "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" <";
}

//count
count1Min = 0;
count1Max = 0;
count2Min = 0;
count2Max = 0;
count3Min = 0;
count3Max = 0;
count4Min = 0;
count4Max = 0;

dataset.addValue( count1Min, series1, categoryMin );
dataset.addValue( count2Min, series2, categoryMin );
dataset.addValue( count3Min, series3, categoryMin );
dataset.addValue( count4Min, series4, categoryMin );

this.category = new String[ this.div ];
this.count1 = new int[ this.div ];
this.count2 = new int[ this.div ];
this.count3 = new int[ this.div ];
this.count4 = new int[ this.div ];

//int k = 0;
for( int i= 0; i<this.div; i++ ){
   category[i] = ""+( this.categoryNumMin + this.range * ( i) )
   +" - "+( this.categoryNumMin + this.range * ( i + 1 ) );
   count1[i] = 0;
   count2[i] = 0;
   count3[i] = 0;
   count4[i] = 0;
   //
   dataset.addValue( count1[i], series1, category[i] );
   dataset.addValue( count2[i], series2, category[i] );
   dataset.addValue( count3[i], series3, category[i] );
   dataset.addValue( count4[i], series4, category[i] );
   //k++;
}
```

```
    dataset.addValue( count1Max, series1, categoryMax );
    dataset.addValue( count2Max, series2, categoryMax );
    dataset.addValue( count1Max, series3, categoryMax );
    dataset.addValue( count2Max, series4, categoryMax );

    return dataset;
}


public void addData( int swc, int mod, BestWater waterP ){

    //System.out.println( moduleName +"addData()");
    //System.out.println( moduleName + "■■isEventDispatchThread() addData ="
    //+ SwingUtilities.isEventDispatchThread() );

    final boolean isOnOff;

    String str = null;
    String stc = null;
    double count = 0.1;

    this.water = waterP;
    //        温度
    //        質量流量

    if( this.dataType.equals( "1_温度[℃]")){
       this.dataValue = this.water.getTemp();
    }else if(this.dataType.equals( "2_質量流量[g/s]")){
       this.dataValue = this.water.getFlowRate();
    }


    //
    if( isswcIn ){
       isOnOff = Airswc.isON( swc );
    }else{
       isOnOff = true;
    }

    if( ismodIn ){
    }else{
       mod = Airmod.onCOOL( mod );
    }

    if( isOnOff ){
       if( Airmod.isCOOL( mod )){
          //str = "COOL/on";
          str = this.series1;
          if( isReverse ){
             if( this.dataValue > categoryNumMin ){
                count = ++count1Min;
                stc = categoryMin;
             }else if( this.dataValue <= categoryNumMax ){
                count = ++count1Max;
                stc = categoryMax;
             }else{
                for( int i = 0; i < div; i++ ){
                   if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                      count = ++count1[ i ];
                      stc = category[ i ];
                      break;
                   }
                }
             }
          }else{
             if( this.dataValue < categoryNumMin ){
                count = ++count1Min;
                stc = categoryMin;
             }else if( this.dataValue >= categoryNumMax ){
                count = ++count1Max;
                stc = categoryMax;
             }else{
```

```
            for( int i = 0; i < div; i++ ){
               if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                  count = ++count1[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }

   }else if( Airmod.isHEAT( mod )){
      //str = "HEAT/on";
      str = this.series3;
      if( isReverse ){
         if( this.dataValue > categoryNumMin ){
            count = --count3Min;
            stc = categoryMin;
         }else if( this.dataValue <= categoryNumMax ){
            count = --count3Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                  count = --count3[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }else{
         if( this.dataValue < categoryNumMin ){
            count = --count3Min;
            stc = categoryMin;
         }else if( this.dataValue >= categoryNumMax ){
            count = --count3Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                  count = --count3[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }

   }
}else{
   if( Airmod.isCOOL( mod )){
      //str = "COOL/off";
      str = this.series2;
      if( isReverse ){
         if( this.dataValue > categoryNumMin ){
            count = ++count2Min;
            stc = categoryMin;
         }else if( this.dataValue <= categoryNumMax ){
            count = ++count2Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                  count = ++count2[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }else{
         if( this.dataValue < categoryNumMin ){
            count = ++count2Min;
            stc = categoryMin;
```

```
            }else if( this.dataValue >= categoryNumMax ){
               count = ++count2Max;
               stc = categoryMax;
            }else{
               for( int i = 0; i < div; i++ ){
                  if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                     count = ++count2[ i ];
                     stc = category[ i ];
                     break;
                  }
               }
            }
         }
      }else if( Airmod.isHEAT( mod )){
         //str = "HEAT/off";
         str = this.series4;
         if( isReverse ){
            if( this.dataValue > categoryNumMin ){
               count = --count4Min;
               stc = categoryMin;
            }else if( this.dataValue <= categoryNumMax ){
               count = --count4Max;
               stc = categoryMax;
            }else{
               for( int i = 0; i < div; i++ ){
                  if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                     count = --count4[ i ];
                     stc = category[ i ];
                     break;
                  }
               }
            }
         }else{
            if( this.dataValue < categoryNumMin ){
               count = --count4Min;
               stc = categoryMin;
            }else if( this.dataValue >= categoryNumMax ){
               count = --count4Max;
               stc = categoryMax;
            }else{
               for( int i = 0; i < div; i++ ){
                  if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                     count = --count4[ i ];
                     stc = category[ i ];
                     break;
                  }
               }
            }
         }
      }
   }

   final double fcount = count;
   final String rowKey = str;
   final String columnKey = stc;

   final Runnable worker = new Runnable() {
      public void run(){
         //System.out.println( moduleName +"worker run()");
         //System.out.println( moduleName + "■■isEventDispatchThread() worker ="
         //+ SwingUtilities.isEventDispatchThread() );
         if( rowKey != null && columnKey != null ){
            dataset.setValue( fcount, rowKey, columnKey );
         }
      }
   };

   Thread runThread = new Thread(){
      public void run() {
         //System.out.println( moduleName +"worker runThread()");
         //System.out.println( moduleName +"■■isEventDispatchThread() runThread ="
         //+ SwingUtilities.isEventDispatchThread() );
```

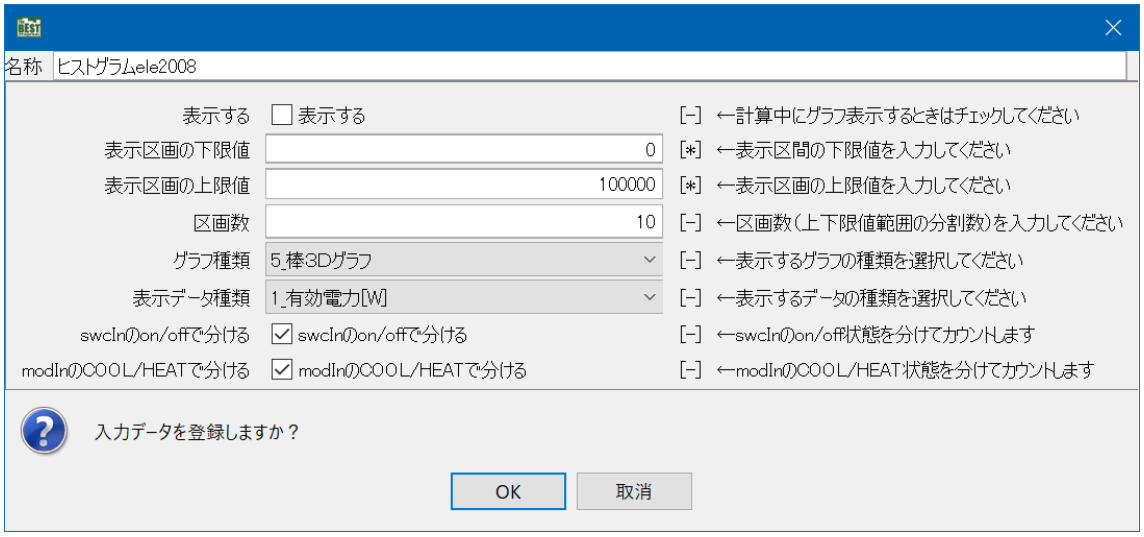
```
            try {
               SwingUtilities.invokeAndWait( worker );
            } catch (InterruptedException e) {
               e.printStackTrace();
            } catch (InvocationTargetException e) {
               e.printStackTrace();
            }
         }
      };

      runThread.start();
   }
}
```

「ヒストグラム ele」（場所：設備 2015／グラフ計測 2015／）

モジュール名	ヒストグラム ele
クラス	GraphRealtimeElectricityHistogramModule20080909 GraphRealtimeElectricityHistogramJFrame20080909

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、観察した BestElectricity 媒体の表示データ種類で指定したデータ（有効電力、無効電力、周波数）について計算中にヒストグラムを作成するモジュールである。

本モジュールの L0_eleObs をヒストグラムを表示したいモジュールの L0_elelOut、L0_eleIn、あるいは L0_eleObs ノードへ接続し、表示するにチェックするだけでその値のヒストグラムを表示することができ、計算中に良し悪しを判断することができる。

GraphRealtimeElectricityHistogramModul20080909 クラスがモジュールの本体で、実際にグラフを表示するのは GraphRealtimeElectricityHistogramJFrame20080909 クラスで行っている。

＊グラフ作成エンジンは JFreeChart を利用している。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

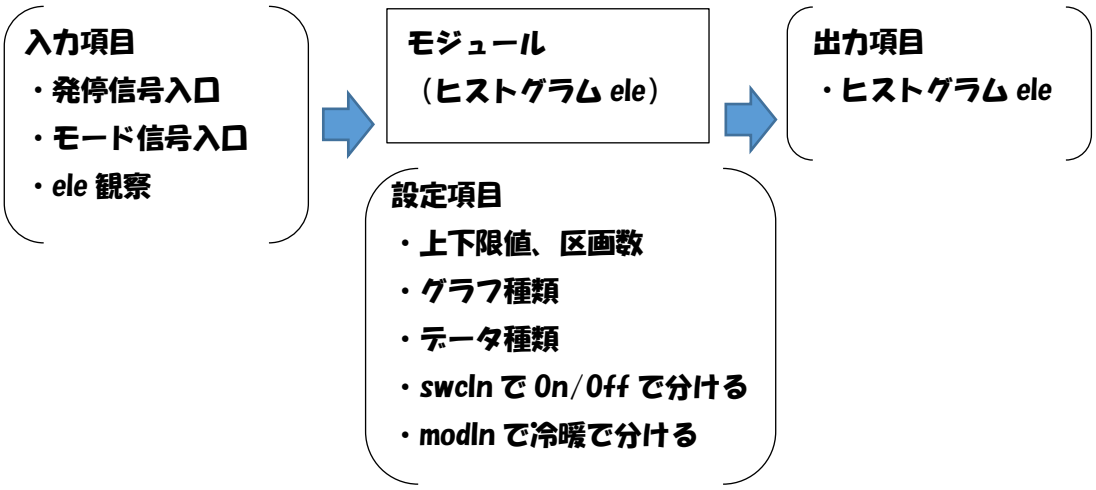


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図1における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]				
1	表示する	boolean	setVisible	FALSE	[-]				←計算中にグラフ表示するときはチェックしてください
2	表示区画の下限値	double	lowerLimit	0	[*]				←表示区間の下限値を入力してください
3	表示区画の上限値	double	upperLimit	40	[*]				←表示区画の上限値を入力してください
4	区画数	int	div	10	[-]				←区画数（上下限値範囲の分割数）を入力してください
5	グラフ種類	String	graphType	1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall	[-]				←表示するグラフの種類を選択してください
6	表示データ種類	String	dataType	1_有効電力[W]、2_無効電力[var]、3_周波数[Hz]	[-]				←表示するデータの種類を選択してください
7	swcIn の on/off で分ける	boolean	isswcIn	TRUE	[-]				←swcIn の on/off 状態を分けてカウントします
8	modIn の COOL/HEAT で分ける	boolean	ismodIn	TRUE	[-]				←modIn の COOL/HEAT 状態を分けてカウントします

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	発停信号入口	LO_swcIn	swcIn	-	制御	OnOff 信号	出口	OnOff 信号の入口
2	モード信号入口	LO_modIn	modIn	-	制御	mod 信号	入口	mod 信号の入口
3	ele 観察	LO_eleObs	eleObs	-	状態	BestElectricity	観察	ele の観察
4								

（6）記録項目

No.	項目	出力内容	単位	分類
1				

このモジュールに記録項目はない。

（7）計算フロー・計算内容

・表示する

有効の時に、計算時にヒストグラムを表示する。

・表示区画の下限値、上限値、区画数

上限値と下限値の間を区画数で均等分割する。

これに下限値未満と上限値以上の区画を加えたもので表示する。

・グラフの種類

グラフの種類は次の８種類から選択する。

1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall

・swcln の on/off で分ける

例えば空調機への swcln を接続しておくと空調機の運転時と停止時とに分けて度数を集計する。

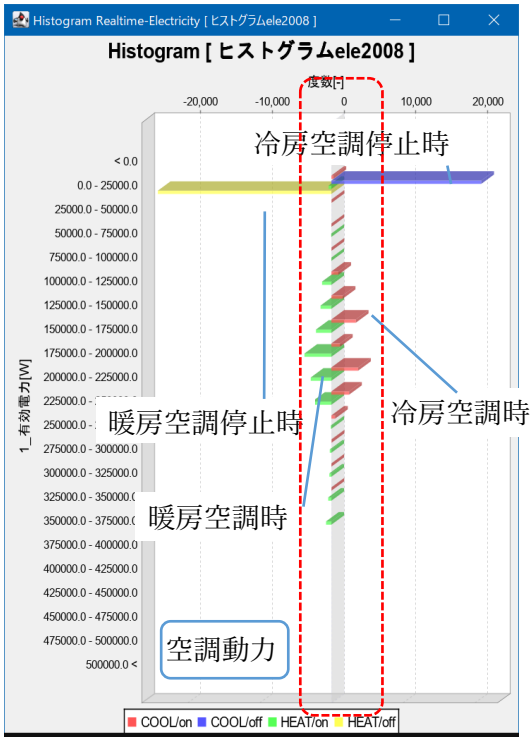
・modln の COOL/HEAT で分ける

例えば空調機への modln を接続しておくと冷房時と暖房時に分けて度数を集計する。

空調動力の実行電力の棒３Ｄヒストグラムの表示例を示す。

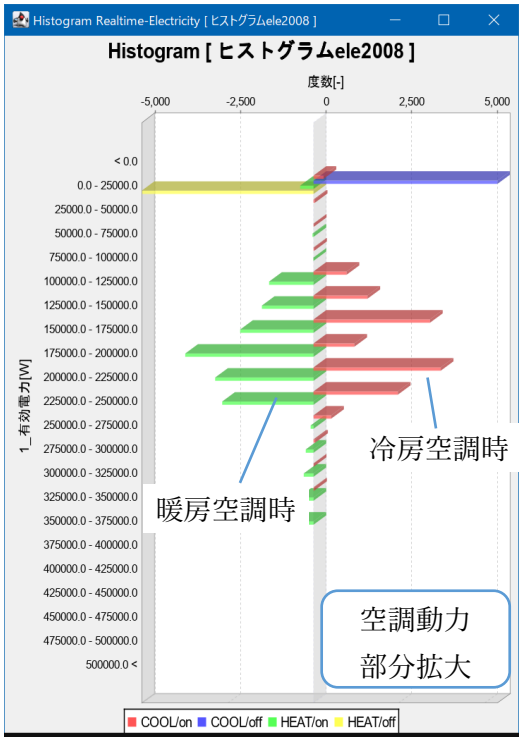
5 分間隔で年間計算（助走期間あり）したもので、度数は計算ステップ数である。

度数×計算時間間隔（5 分）÷60 で年間の時間数が求められる。



空調動力の状態を描画した例

下限値 0、上限値 5000000、20 区画



空調動力を部分拡大描画した例

下限値 0、上限値 500000、20 区画

（８）データ範囲と範囲外の取扱い

特になし。

参考ソース

■GraphRealtimeElectricityHistogramModule20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.Map;

import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.dk.test.TestCommand;

/**
 * @author HIROSHI NINOMIYA /20080827 /20080908
 * GraphRealtimeElectricityHistogramModule20080827 リアルタイムグラフ度数表示です
 *   BestWater  に接続する  表示データ種類として
 *     有効電力、無効電力、周波数
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 *
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 * isswcIn simodInを追加
 */
public class GraphRealtimeElectricityHistogramModule20080909 extends AbstractBestModule implements
      IBestMetaModule {

   private final String moduleName = "(GraphRealtimeElectricityHistogramModule20080909) ";

   //接続ノード
   //出入り口
   private final String S_NODE_eleObs  = "L0_eleObs"; //観測電力

   //制御ノード
   private final String C_NODE_swcIn = "L1_swcIn";
   private final String C_NODE_modIn = "L1_modIn";

   //記録ノード

   //仕様
   private final String SPEC_name      = "名称";      //外部定義項目
   private final String SPEC_Visible = "表示する"; //描画する  true、false
   private final String SPEC_lowerLimit = "表示区画の下限値";  //表示区画の下限値
   private final String SPEC_upperLimit = "表示区画の上限値";  //表示区画の上限値
   private final String SPEC_div     = "区画数";   //区画数
   private final String SPEC_graphType = "グラフ種類";//グラフ種類
   private final String SPEC_dataType  = "表示データ種類";//表示するデータの種類
   //       有効電力
   //       無効電力
   //       周波数
   private final String SPEC_isswcIn = "swcInのon/offで分ける";//on/offでカウントを分ける
   private final String SPEC_ismodIn = "modInのCOOL/HEATで分ける";//COOL・HEATでカウントを分ける
   private boolean isswcIn = false;//on/offでカウントを分ける
   private boolean ismodIn = false;//COOL・HEATでカウントを分ける
   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

   //仕様など
   private StringBuffer message = new StringBuffer();    //メッセージ
   private String  name = null;  //機器名称
   private boolean setVisible = true;  //描画するか  どうか
   private double lowerLimit;//表示区画の下限値
```

```
private double upperLimit;//表示区画の上限値
private int div;     //片側区画数
private String graphType = null; //表示す売るグラフの種類
private String dataType = null;  //表示するデータの種類
private boolean isRecord = false;//記録を有効とする=true

//接続熱媒など
private BestElectricity eleObs  = null;

private int swcIn;
private int modIn;

//グラフ描画データなど
private GraphRealtimeElectricityHistogramJFrame20080909 gElectricityHistogram = null;


@Override
public void setProfile(BestSpecs spec) {
   if(spec == null) {
      return;
   }

   Map<String, String> map = spec.getSpec();
   if(map == null) {
      return;
   }

   //名称を取得
   if(null != map.get(this.SPEC_name)) {
      this.name = (String)map.get(this.SPEC_name);
   }else{
      System.out.println(this.moduleName + "(W)名称がありません");
      this.name = "InstrumentAir";
   }

   //Visibleを取得
   if(null != map.get(this.SPEC_Visible)) {
      this.setVisible = Boolean.parseBoolean( map.get(this.SPEC_Visible) );
   }else{
      System.out.println(this.moduleName + "(W)描画指定がありません");
      this.setVisible = true;
   }

   //lowerLimitを取得
   if(null != map.get(this.SPEC_lowerLimit)) {
      this.lowerLimit = Double.parseDouble( map.get(this.SPEC_lowerLimit) );
   }else{
      System.out.println(this.moduleName + "(W)表示区画の下限値がありません->=0");
      this.lowerLimit = 0;
   }


   //upperLimitを取得
   if(null != map.get(this.SPEC_upperLimit)) {
      this.upperLimit = Double.parseDouble( map.get(this.SPEC_upperLimit) );
   }else{
      System.out.println(this.moduleName + "(W)表示区画の上限値がありません->=100000");
      this.upperLimit = 1000000;
   }

   //divを取得
   if(null != map.get(this.SPEC_div)) {
      this.div = Integer.parseInt( map.get(this.SPEC_div) );
   }else{
      System.out.println(this.moduleName + "(W)区画数がありません->=10");
      this.div = 10;
   }

   //graphTypeを取得
   if(null != map.get(this.SPEC_graphType)) {
      this.graphType = map.get(this.SPEC_graphType);
   }else{
```

```
        System.out.println(this.moduleName + "(W)グラフ種類がありません");
        this.graphType = "5_棒３Ｄグラフ";
    }

    //dataTypeを取得
    if(null != map.get(this.SPEC_dataType)) {
        this.dataType = map.get(this.SPEC_dataType);
    }else{
        System.out.println(this.moduleName + "(W)表示するデータ種類がありません");
        this.dataType = "1_有効電力[W]";
    }

    //isswcIn
    if(null != map.get(this.SPEC_isswcIn)) {
        this.isswcIn = Boolean.parseBoolean( map.get(this.SPEC_isswcIn) );
    }else{
        System.out.println(this.moduleName + "(W)isswcIn指定がありません->=false");
        this.isswcIn = false;
    }

    //ismodIn
    if(null != map.get(this.SPEC_ismodIn)) {
        this.ismodIn = Boolean.parseBoolean( map.get(this.SPEC_ismodIn) );
    }else{
        System.out.println(this.moduleName + "(W)ismodIn指定がありません->=false");
        this.ismodIn = false;
    }

    //isRecordを取得
    if(null != map.get(this.SPEC_isRecord)) {
        this.isRecord = Boolean.parseBoolean( map.get(this.SPEC_isRecord) );
    }else{
        System.out.println(this.moduleName + "(W)記録指定がありません->=false");
        this.isRecord = false;
    }
}


public void initialize(IBestStateMessage stateNodes,
        IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
        IBestDomainSchedule schedule) {
    //状態ノードを取得
    super.sm = stateNodes;

    //制御ノードを取得
    super.cm = commandNodes;

    //記録ノードを取得
    super.rm = recordNodes;


    //接続ノード　出口

    //接続ノード　入口チェック
    //eleObs
    if( super.sm.getState( super.getConnectionNode( this.S_NODE_eleObs )) != null ){
        this.eleObs
        = (BestElectricity) super.sm.getState( super.getConnectionNode( this.S_NODE_eleObs));
    }else{
        this.eleObs = new BestElectricity();
        super.sm.setState( super.getConnectionNode( this.S_NODE_eleObs ), this.eleObs );
        message.append( "(W)観測電力の接続なし→作成");
        System.out.println( this.moduleName + ">>Warning<< eleObs is null !!" );
    }


    //制御ノード

    //グラフの描画準備
    if( this.setVisible ){
        //
        gElectricityHistogram = new GraphRealtimeElectricityHistogramJFrame20080909( this.name,
```

```
            this.lowerLimit, this.upperLimit, this.div, this.graphType, this.dataType, this.isswcIn, this.ismodIn );
    }

}

public void outputs() {
    if(super.sm == null || super.cm == null ){
        return;
    }

    //
    this.eleObs
    = (BestElectricity)super.sm.getState(super.getConnectionNode(this.S_NODE_eleObs));
    this.swcIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ));
    this.modIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn ));

    //ノードに設定

    //新しいデータを加える
    if( this.setVisible ){
        gElectricityHistogram.addData( this.swcIn, this.modIn, this.eleObs );
    }

    //記録

    if( this.isRecord  && super.rm != null ){
        this.record();
    }

    message.setLength(0);
}

/**
 * 記録
 */
private void record(){
    if( CheckPrintModule.isPrintMessage ){
    }

    //if( CheckPrintModule.isPrintEnergy ){
    //}

    //if( CheckPrintModule.isPrintLoad ){
    //}

    //if( CheckPrintModule.isPrintStateOut ){
    //}

    //if( CheckPrintModule.isPrintStateMy ){
    //}

    //if( CheckPrintModule.isPrintStateIn ){
    //}
}



public void update() {

}


public Object viewInternal(TestCommand cmd) {
    ArrayList<Object> result = new ArrayList<Object>();
    result.add(super.sm);
    result.add(super.cm);
    result.add(super.rm);

    //外部定義項目

    //記録ノードに設定する値
    if(eleObs != null) {
```

```
            result.add(new Double(this.eleObs.getActivePower())); //有効電力
            result.add(new Double(this.eleObs.getReactivePower()));//無効電力
            result.add(new Double(this.eleObs.getFrequency()));   //周波数
        }else{
            result.add(new Double(0.0));
            result.add(new Double(0.0));
            result.add(new Double(0.0));
        }

        result.add(super.transferMapCommand);
        result.add(super.transferMapRecord);
        result.add(super.transferMapState);
        return result;
    }
}
```

■GraphRealtimeValHistogramJFrame20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.awt.BorderLayout;
import java.awt.Container;
import java.lang.reflect.InvocationTargetException;

import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

import jp.or.ibec.best.DO.BestElectricity;

/**
 * @author HIROSHI NINOMIYA /20080827 /20080908 /20080909
 * GraphRealtimeElectricityHistogramJFrame20080827 リアルタイムグラフ表示部分です
 *   BestAir  に接続する  表示データ種類として
 *     有効電力、無効電力、周波数
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 */
public class GraphRealtimeElectricityHistogramJFrame20080909 {

   private final String moduleName = "(GraphRealtimeElectricityHistogramJFrame20080909) ";

   private JFrame jFrame = null;

   private JFreeChart chart = null;

   //仕様など
   private String  name = null;  //機器名称
   private double lowerLimit;//表示区画の下限値
   private double upperLimit;//表示区画の上限値
   private double range;//区画幅
   private int div;//区画数
   private String graphType = null; //表示す売るグラフの種類
   private String dataType = null;  //表示するデータの種類
   private boolean isswcIn;
   private boolean ismodIn;

   //グラフ描画データなど
   private DefaultCategoryDataset dataset = null;
   private String categoryMin = null;
   private String categoryMax = null;
   private String[] category;

   private double categoryNumMin;
   private double categoryNumMax;
   private int count1Min;
   private int count1Max;
   private int[] count1;
   private int count2Min;
   private int count2Max;
   private int[] count2;
   private int count3Min;
   private int count3Max;
   private int[] count3;
   private int count4Min;
   private int count4Max;
   private int[] count4;
   double dataValue;
```

```
private boolean isReverse = false;//lowerLimit  とupperLimitが逆転している=true

private BestElectricity ele = new BestElectricity();


public GraphRealtimeElectricityHistogramJFrame20080909() {
   this( "GraphRealtimeElectricityHistogramJFrame", 0, 1000000, 10, "5_棒３Ｄグラフ", "1_有効電力[W]", true, true  );
}


public GraphRealtimeElectricityHistogramJFrame20080909( String name, double lowerLimit, double upperLimit,
      int div, String graphType, String dataType, boolean isswcIn, boolean ismodIn ) {
   this.name = name;
   this.lowerLimit = lowerLimit;
   this.upperLimit = upperLimit;
   this.div    = div;
   this.graphType = graphType;
   this.dataType  = dataType;
   this.isswcIn = isswcIn;
   this.ismodIn = ismodIn;

   createAndShowGUI();
}


public void setElectricity( BestElectricity ele ){
   this.ele = ele;
}


 private void createAndShowGUI() {
     //Create and set up the window.
   this.jFrame = new JFrame();

   //this.jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   this.jFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

   this.jFrame.setTitle(  "Histogram Realtime-Electricity [ " +  this.name + " ]"  );

   Container contentpane = this.jFrame.getContentPane();

   Container panel = this.createChart();

   if( panel == null ){
      System.out.println( " panel == null ");
      System.exit(0);
   }else{
      System.out.println( " panel != null ");
   }

   contentpane.add( panel, BorderLayout.CENTER  );

   //グラフの描画準備

   this.jFrame.setSize(500,700);
     //this.jFrame.pack();
     this.jFrame.setVisible(true);
 }


 public static void main(String[] args) {
     //Schedule a job for the event-dispatching thread:
     //creating and showing this application's GUI.
   /*
   javax.swing.SwingUtilities.invokeLater( new Runnable() {
       public void run() {
           createAndShowGUI();
       }
   });
   */
 }
```

```
private JComponent createChart(){
   //
   dataset = createDataset();

   //
   if( this.graphType.equals( "8_Waterfall" )){
      this.chart = ChartFactory.createWaterfallChart(
            "Histogram [ "+this.name+" ]",    this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "1_棒積上げ３Ｄグラフ" )){
      this.chart = ChartFactory.createStackedBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "2_棒積上げグラフ" )){
      this.chart = ChartFactory.createStackedBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "3_折れ線グラフ" )){
      this.chart = ChartFactory.createLineChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "4_折れ線３Ｄグラフ" )){
      this.chart = ChartFactory.createLineChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "5_棒３Ｄグラフ" )){
      this.chart = ChartFactory.createBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "6_棒グラフ" )){
      this.chart = ChartFactory.createBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "7_面グラフ" )){
      this.chart = ChartFactory.createAreaChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else{
      System.out.println( this.moduleName + "graphType error!" );
   }

   ChartPanel cPanel = new ChartPanel( this.chart );

   return cPanel;
}


//row keys
private String series1 = null;
private String series2 = null;
private String series3 = null;
private String series4 = null;

private DefaultCategoryDataset createDataset(){
   //dataset
   this.dataset = new DefaultCategoryDataset();

   this.categoryNumMin = this.lowerLimit;
   this.categoryNumMax = this.upperLimit;
   this.range = ( this.upperLimit - this.lowerLimit ) / this.div;

   if( this.lowerLimit > this.upperLimit ){
      this.isReverse = true;
   }


   if( isswcIn ){
      if( ismodIn ){
         //row keys
         series1 = "COOL/on";
```

```
      series2 = "COOL/off";
      series3 = "HEAT/on";
      series4 = "HEAT/off";
   }else{
      //row keys
      series1 = "/on";
      series2 = "/off";
      series3 = "/on";
      series4 = "/off";
   }
}else{
   if( ismodIn ){
      //row keys
      series1 = "COOL/";
      series2 = "COOL/";
      series3 = "HEAT/";
      series4 = "HEAT/";
   }else{
      //row keys
      series1 = "/";
      series2 = "/";
      series3 = "/";
      series4 = "/";
   }
}

//column keys
if( this.isReverse ){
   this.categoryMin = "> "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" >";
}else{
   this.categoryMin = "< "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" <";
}

//count
count1Min = 0;
count1Max = 0;
count2Min = 0;
count2Max = 0;
count3Min = 0;
count3Max = 0;
count4Min = 0;
count4Max = 0;

dataset.addValue( count1Min, series1, categoryMin );
dataset.addValue( count2Min, series2, categoryMin );
dataset.addValue( count3Min, series3, categoryMin );
dataset.addValue( count4Min, series4, categoryMin );

this.category = new String[ this.div ];
this.count1 = new int[ this.div ];
this.count2 = new int[ this.div ];
this.count3 = new int[ this.div ];
this.count4 = new int[ this.div ];

//int k = 0;
for( int i= 0; i<this.div; i++ ){
   category[i] = ""+( this.categoryNumMin + this.range * ( i) )
   +" - "+( this.categoryNumMin + this.range * ( i + 1 ) );
   count1[i] = 0;
   count2[i] = 0;
   count3[i] = 0;
   count4[i] = 0;
   //
   dataset.addValue( count1[i], series1, category[i] );
   dataset.addValue( count2[i], series2, category[i] );
   dataset.addValue( count3[i], series3, category[i] );
   dataset.addValue( count4[i], series4, category[i] );
   //k++;
}
```

```
    dataset.addValue( count1Max, series1, categoryMax );
    dataset.addValue( count2Max, series2, categoryMax );
    dataset.addValue( count1Max, series3, categoryMax );
    dataset.addValue( count2Max, series4, categoryMax );

    return dataset;
}


public void addData( int swc, int mod, BestElectricity eleP ){

    //System.out.println( moduleName +"addData()");
    //System.out.println( moduleName + "■■isEventDispatchThread() addData ="
    //+ SwingUtilities.isEventDispatchThread() );

    final boolean isOnOff;

    String str = null;
    String stc = null;
    double count = 0.1;

    this.ele = eleP;
    //        有効電力
    //        無効電力
    //        周波数

    if( this.dataType.equals( "1_有効電力[W]")){
        this.dataValue = this.ele.getActivePower();
    }else if(this.dataType.equals( "2_無効電力[var]")){
        this.dataValue = this.ele.getReactivePower();
    }else if(this.dataType.equals( "3_周波数[Hz]")){
        this.dataValue = this.ele.getFrequency();
    }


    //
    if( isswcIn ){
        isOnOff = Airswc.isON( swc );
    }else{
        isOnOff = true;
    }

    if( ismodIn ){
    }else{
        mod = Airmod.onCOOL( mod );
    }

    if( isOnOff ){
        if( Airmod.isCOOL( mod )){
            //str = "COOL/on";
            str = this.series1;
            if( isReverse ){
                if( this.dataValue > categoryNumMin ){
                    count = ++count1Min;
                    stc = categoryMin;
                }else if( this.dataValue <= categoryNumMax ){
                    count = ++count1Max;
                    stc = categoryMax;
                }else{
                    for( int i = 0; i < div; i++ ){
                        if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                            count = ++count1[ i ];
                            stc = category[ i ];
                            break;
                        }
                    }
                }
            }else{
                if( this.dataValue < categoryNumMin ){
                    count = ++count1Min;
                    stc = categoryMin;
                }else if( this.dataValue >= categoryNumMax ){
```

```
            count = ++count1Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                  count = ++count1[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }

   }else if( Airmod.isHEAT( mod )){
      //str = "HEAT/on";
      str = this.series3;
      if( isReverse ){
         if( this.dataValue > categoryNumMin ){
            count = --count3Min;
            stc = categoryMin;
         }else if( this.dataValue <= categoryNumMax ){
            count = --count3Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                  count = --count3[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }else{
         if( this.dataValue < categoryNumMin ){
            count = --count3Min;
            stc = categoryMin;
         }else if( this.dataValue >= categoryNumMax ){
            count = --count3Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                  count = --count3[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }

   }
}else{
   if( Airmod.isCOOL( mod )){
      //str = "COOL/off";
      str = this.series2;
      if( isReverse ){
         if( this.dataValue > categoryNumMin ){
            count = ++count2Min;
            stc = categoryMin;
         }else if( this.dataValue <= categoryNumMax ){
            count = ++count2Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                  count = ++count2[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }else{
```

```
            if( this.dataValue < categoryNumMin ){
               count = ++count2Min;
               stc = categoryMin;
            }else if( this.dataValue >= categoryNumMax ){
               count = ++count2Max;
               stc = categoryMax;
            }else{
               for( int i = 0; i < div; i++ ){
                  if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                     count = ++count2[ i ];
                     stc = category[ i ];
                     break;
                  }
               }
            }
         }
      }else if( Airmod.isHEAT( mod )){
         //str = "HEAT/off";
         str = this.series4;
         if( isReverse ){
            if( this.dataValue > categoryNumMin ){
               count = --count4Min;
               stc = categoryMin;
            }else if( this.dataValue <= categoryNumMax ){
               count = --count4Max;
               stc = categoryMax;
            }else{
               for( int i = 0; i < div; i++ ){
                  if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                     count = --count4[ i ];
                     stc = category[ i ];
                     break;
                  }
               }
            }
         }else{
            if( this.dataValue < categoryNumMin ){
               count = --count4Min;
               stc = categoryMin;
            }else if( this.dataValue >= categoryNumMax ){
               count = --count4Max;
               stc = categoryMax;
            }else{
               for( int i = 0; i < div; i++ ){
                  if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                     count = --count4[ i ];
                     stc = category[ i ];
                     break;
                  }
               }
            }
         }
      }
   }

   final double fcount = count;
   final String rowKey = str;
   final String columnKey = stc;

   final Runnable worker = new Runnable() {
      public void run(){
         //System.out.println( moduleName +"worker run()");
         //System.out.println( moduleName + "■■isEventDispatchThread() worker ="
         //+ SwingUtilities.isEventDispatchThread() );
         if( rowKey != null && columnKey != null ){
            dataset.setValue( fcount, rowKey, columnKey );
         }
      }
   };

   Thread runThread = new Thread(){
      public void run() {
```

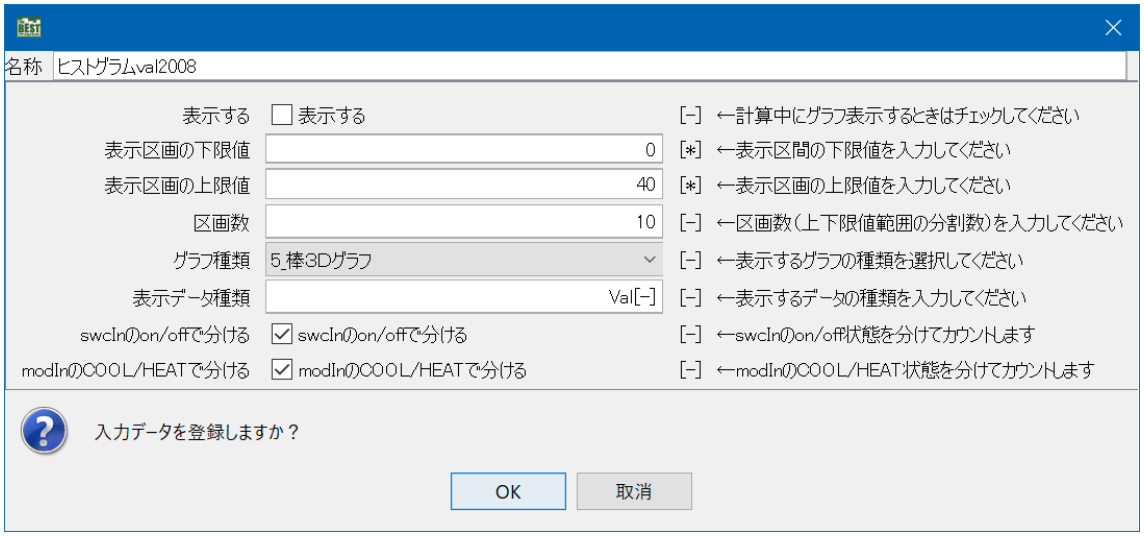
```
            //System.out.println( moduleName +"worker runThread()");
            //System.out.println( moduleName +"■■isEventDispatchThread() runThread ="
            //+ SwingUtilities.isEventDispatchThread() );
            try {
               SwingUtilities.invokeAndWait( worker );
            } catch (InterruptedException e) {
               e.printStackTrace();
            } catch (InvocationTargetException e) {
               e.printStackTrace();
            }
         }
      };

      runThread.start();
   }
}
```

「ヒストグラム val」（場所：設備 2015／グラフ計測 2015／）

モジュール名	ヒストグラム wat
クラス	GraphRealtimeValHistogramModule20080909 GraphRealtimeValHistogramJFrame20080909

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、観察した BestValue 媒体の表示データ種類で指定したデータ（値）について計算中にヒストグラムを作成するモジュールである。

本モジュールの L0_valObs をヒストグラムを表示したいモジュールの L0_valOut、L0_valIn、あるいは L0_valObs ノードへ接続し、表示するにチェックするだけでその値のヒストグラムを表示することができ、計算中に良し悪しを判断することができる。

GraphRealtimeValHistogramModul20080909 クラスがモジュールの本体で、実際にグラフを表示するのは GraphRealtimeValHistogramJFrame20080909 クラスで行っている。

＊グラフ作成エンジンは JFreeChart を利用している。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]				
1	表示する	boolean	setVisible	FALSE	[-]				←計算中にグラフ表示するときはチェックしてください
2	表示区画の下限値	double	lowerLimit	0	[*]				←表示区間の下限値を入力してください
3	表示区画の上限値	double	upperLimit	40	[*]				←表示区画の上限値を入力してください
4	区画数	int	div	10	[-]				←区画数（上下限値範囲の分割数）を入力してください
5	グラフ種類	String	graphType	1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall	[-]				←表示するグラフの種類を選択してください
6	表示データ種類	String	dataType	Val[-]	[-]				←表示するデータの種類を選択してください
7	swcIn の on/off で分ける	boolean	isswcIn	TRUE	[-]				←swcIn の on/off 状態を分けてカウントします
8	modIn の COOL/HEAT で分ける	boolean	ismodIn	TRUE	[-]				←modIn の COOL/HEAT 状態を分けてカウントします

（5）シーケンス接続（図1における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	発停信号入口	LO_swcIn	swcIn	-	制御	OnOff 信号	出口	OnOff 信号の入口
2	モード信号入口	LO_modIn	modIn	-	制御	mod 信号	入口	mod 信号の入口
3	値（val）観察	LO_valObs	valObs	-	状態	値	観察	値（val）の観察
4								

（6）記録項目

No.	項目	出力内容	単位	分類
1				

このモジュールに記録項目はない。

（7）計算フロー・計算内容

・表示する

有効の時に、計算時にヒストグラムを表示する。

・表示区画の下限値、上限値、区画数

上限値と下限値の間を区画数で均等分割する。

これに下限値未満と上限値以上の区画を加えたもので表示する。

・グラフの種類

グラフの種類は次の８種類から選択する。

1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall

・swcln の on/off で分ける

例えば空調機への swcln を接続しておくと空調機の運転時と停止時とに分けて度数を集計する。

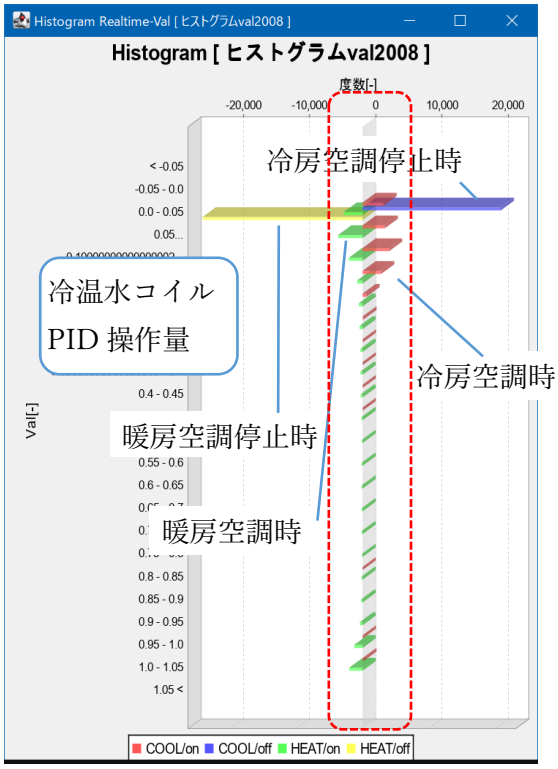
・modln の COOL/HEAT で分ける

例えば空調機への modln を接続しておくと冷房時と暖房時に分けて度数を集計する。

冷・温熱源群の出口の温度の棒ヒストグラム、棒３Ｄヒストグラムの表示例を示す。

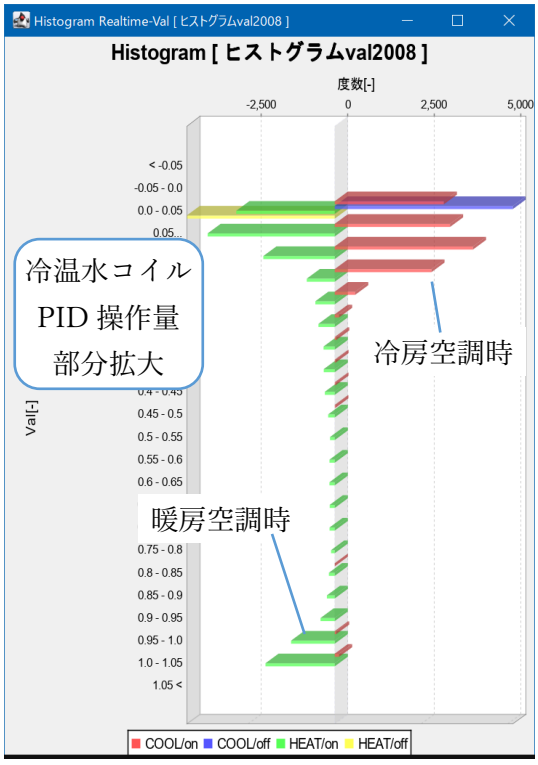
5 分間隔で年間計算（助走期間あり）したもので、度数は計算ステップ数である。

度数×計算時間間隔（5 分）÷60 で年間の時間数が求められる。



コイル用 PID 操作量の状態を描画した例

下限値-0.05、上限値 1.05、20 区画



コイル用 PID 操作量を部分拡大描画した例

下限値-0.05、上限値 1.05、20 区画

（８）データ範囲と範囲外の取扱い

特になし。

参考ソース

■GraphRealtimeValHistogramModule20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.Map;

import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.dk.test.TestCommand;

/**
 * @author HIROSHI NINOMIYA /20080827 /20080908
 * GraphRealtimeValHistogramModule20080827 リアルタイムグラフ度数表示です
 *   Val Double  に接続する
 *   表示データ種類としてユーザー指定データ
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 *
 * GraphRealtimeBarAirModule  を基に接続ノードを「観測/Obs」に変更
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 * isswcIn simodInを追加
 */
public class GraphRealtimeValHistogramModule20080909 extends AbstractBestModule implements
      IBestMetaModule {

   private final String moduleName = "(GraphRealtimeValHistogramModule20080909) ";

   //接続ノード
   //出入り口
   private final String S_NODE_valObs  = "LO_valObs"; //観測Val

   //制御ノード
   private final String C_NODE_swcIn = "L1_swcIn";
   private final String C_NODE_modIn = "L1_modIn";

   //記録ノード

   //仕様
   private final String SPEC_name      = "名称";      //外部定義項目
   private final String SPEC_Visible = "表示する"; //描画する  true、false
   private final String SPEC_lowerLimit  = "表示区画の下限値"; //表示区画の下限値
   private final String SPEC_upperLimit   = "表示区画の上限値";   //表示区画の上限値
   private final String SPEC_div     = "区画数";   //区画数
   private final String SPEC_graphType = "グラフ種類";//グラフ種類
   private final String SPEC_dataType  = "表示データ種類";//表示するデータの種類
   private final String SPEC_isswcIn = "swcInのon/offで分ける";//on/offでカウントを分ける
   private final String SPEC_ismodIn = "modInのCOOL/HEATで分ける";//COOL・HEATでカウントを分ける
   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

   //仕様など
   private StringBuffer message = new StringBuffer();    //メッセージ
   private String  name = null;  //機器名称
   private boolean setVisible = true;  //描画するか  どうか
   private double lowerLimit;    //表示区画の下限値
   private double upperLimit;    //表示区画の上限値
   private int div;     //区画数
   private String graphType = null; //表示す売るグラフの種類
   private String dataType = null;  //表示するデータの種類
```

```
private boolean isswcIn = false;//on/offでカウントを分ける
private boolean ismodIn = false;//COOL・HEATでカウントを分ける
private boolean isRecord = false;//記録を有効とする=true

//接続熱媒など
private BestValue valObs  = null;

private int swcIn;
private int modIn;

//グラフ描画データなど
private GraphRealtimeValHistogramJFrame20080909 gValHistogram = null;


@Override
public void setProfile(BestSpecs spec) {
   if(spec == null) {
      return;
   }

   Map<String, String> map = spec.getSpec();
   if(map == null) {
      return;
   }

   //名称を取得
   if(null != map.get(this.SPEC_name)) {
      this.name = (String)map.get(this.SPEC_name);
   }else{
      System.out.println(this.moduleName + "(W)名称がありません");
      this.name = "InstrumentAir";
   }

   //Visibleを取得
   if(null != map.get(this.SPEC_Visible)) {
      this.setVisible = Boolean.parseBoolean( map.get(this.SPEC_Visible) );
   }else{
      System.out.println(this.moduleName + "(W)描画指定がありません");
      this.setVisible = true;
   }

   //lowerLimitを取得
   if(null != map.get(this.SPEC_lowerLimit)) {
      this.lowerLimit = Double.parseDouble( map.get(this.SPEC_lowerLimit) );
   }else{
      System.out.println(this.moduleName + "(W)表示区画の下限値がありません->=0");
      this.lowerLimit = 0;
   }


   //upperLimitを取得
   if(null != map.get(this.SPEC_upperLimit)) {
      this.upperLimit = Double.parseDouble( map.get(this.SPEC_upperLimit) );
   }else{
      System.out.println(this.moduleName + "(W)表示区画の上限値がありません->=40");
      this.upperLimit = 40;
   }

   //divを取得
   if(null != map.get(this.SPEC_div)) {
      this.div = Integer.parseInt( map.get(this.SPEC_div) );
   }else{
      System.out.println(this.moduleName + "(W)区画数がありません->=10");
      this.div = 10;
   }

   //graphTypeを取得
   if(null != map.get(this.SPEC_graphType)) {
      this.graphType = map.get(this.SPEC_graphType);
   }else{
      System.out.println(this.moduleName + "(W)グラフ種類がありません");
      this.graphType = "5_棒３Ｄグラフ";
```

```
    }

    //dataTypeを取得
    if(null != map.get(this.SPEC_dataType)) {
       this.dataType = map.get(this.SPEC_dataType);
    }else{
       System.out.println(this.moduleName + "(W)表示するデータ種類がありません");
       this.dataType = "Val[-]";
    }

    //isswcIn
    if(null != map.get(this.SPEC_isswcIn)) {
       this.isswcIn = Boolean.parseBoolean( map.get(this.SPEC_isswcIn) );
    }else{
       System.out.println(this.moduleName + "(W)isswcIn指定がありません->=false");
       this.isswcIn = false;
    }

    //ismodIn
    if(null != map.get(this.SPEC_ismodIn)) {
       this.ismodIn = Boolean.parseBoolean( map.get(this.SPEC_ismodIn) );
    }else{
       System.out.println(this.moduleName + "(W)ismodIn指定がありません->=false");
       this.ismodIn = false;
    }

    //isRecordを取得
    if(null != map.get(this.SPEC_isRecord)) {
       this.isRecord = Boolean.parseBoolean( map.get(this.SPEC_isRecord) );
    }else{
       System.out.println(this.moduleName + "(W)記録指定がありません->=false");
       this.isRecord = false;
    }
}


public void initialize(IBestStateMessage stateNodes,
      IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
      IBestDomainSchedule schedule) {
    //状態ノードを取得
    super.sm = stateNodes;

    //制御ノードを取得
    super.cm = commandNodes;

    //記録ノードを取得
    super.rm = recordNodes;


    //接続ノード　出口

    //接続ノード　入口チェック
    //airObs
    if( super.sm.getState( super.getConnectionNode( this.S_NODE_valObs )) != null ){
       this.valObs
       = (BestValue) super.sm.getState( super.getConnectionNode( this.S_NODE_valObs));
    }else{
       this.valObs = new BestValue( );
       super.sm.setState( super.getConnectionNode( this.S_NODE_valObs ), this.valObs );
       message.append( "(W)観測Valの接続なし→作成");
       System.out.println( this.moduleName + ">>Warning<< valObs is null !!" );
    }


    //制御ノード

    //グラフの描画準備
    if( this.setVisible ){
       //
       gValHistogram = new GraphRealtimeValHistogramJFrame20080909( this.name,
             this.lowerLimit, this.upperLimit, this.div, this.graphType, this.dataType, this.isswcIn, this.ismodIn );
    }
```

```
}

public void outputs() {
   if(super.sm == null || super.cm == null ){
      return;
   }

   //
   this.valObs
   = (BestValue)super.sm.getState(super.getConnectionNode(this.S_NODE_valObs));
   this.swcIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ));
   this.modIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn ));

   //ノードに設定

   //新しいデータを加える
   if( this.setVisible ){
      gValHistogram.addData( this.swcIn, this.modIn, this.valObs.getValue() );
   }

   //記録

   if( this.isRecord  && super.rm != null ){
      this.record();
   }

   message.setLength(0);
}

/**
 * 記録
 */
private void record(){
   if( CheckPrintModule.isPrintMessage ){
   }

   //if( CheckPrintModule.isPrintEnergy ){
   //}

   //if( CheckPrintModule.isPrintLoad ){
   //}

   //if( CheckPrintModule.isPrintStateOut ){
   //}

   //if( CheckPrintModule.isPrintStateMy ){
   //}

   //if( CheckPrintModule.isPrintStateIn ){
   //}
}


public void update() {

}


public Object viewInternal(TestCommand cmd) {
   ArrayList<Object> result = new ArrayList<Object>();
   result.add(super.sm);
   result.add(super.cm);
   result.add(super.rm);

   //外部定義項目

   //記録ノードに設定する値
   if(valObs != null) {
      result.add(new Double(this.valObs.getValue()));    //
   }else{
      result.add(new Double(0.0));
```

```
        }

        result.add(super.transferMapCommand);
        result.add(super.transferMapRecord);
        result.add(super.transferMapState);
        return result;
    }
}
```

■GraphRealtimeValHistogramJFrame20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.awt.BorderLayout;
import java.awt.Container;
import java.lang.reflect.InvocationTargetException;

import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;


/**
 * @author HIROSHI NINOMIYA /20080827 /20080908 /20080909
 * GraphRealtimeValHistogramJFrame20080827 リアルタイムグラフ表示部分です
 *   Val   (Double)   に接続する
 * 表示データ種類  Val *
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 */
public class GraphRealtimeValHistogramJFrame20080909 {

   private final String moduleName = "(GraphRealtimeValHistogramJFrame20080909) ";

   private JFrame jFrame = null;

   private JFreeChart chart = null;

   //仕様など
   private String  name = null;  //機器名称
   private double lowerLimit;//表示区画の下限値
   private double upperLimit;//表示区画の上限値
   private double range;      //１区画の幅
   private int div;     //区画数
   private String graphType = null; //表示す売るグラフの種類
   private String dataType = null;  //表示するデータの種類
   private boolean isswcIn;
   private boolean ismodIn;

   //グラフ描画データなど
   private DefaultCategoryDataset dataset = null;
   private String categoryMin = null;
   private String categoryMax = null;
   private String[] category;

   private double categoryNumMin;
   private double categoryNumMax;
   private int count1Min;
   private int count1Max;
   private int[] count1;
   private int count2Min;
   private int count2Max;
   private int[] count2;
   private int count3Min;
   private int count3Max;
   private int[] count3;
   private int count4Min;
   private int count4Max;
   private int[] count4;
   double dataValue;
```

```
private boolean isReverse = false;//lowerLimit  とupperLimitが逆転している=true

private Double val = new Double( 0.);


public GraphRealtimeValHistogramJFrame20080909() {
   this( "GraphRealtimeValHistogramJFrame", 0, 40, 20, "5_棒３Ｄグラフ", "Val[*]", true, true  );
}


public GraphRealtimeValHistogramJFrame20080909( String name, double lowerLimit, double upperLimit,
      int div, String graphType, String dataType, boolean isswcIn, boolean ismodIn ) {
   this.name = name;
   this.lowerLimit = lowerLimit;
   this.upperLimit = upperLimit;
   this.div    = div;
   this.graphType = graphType;
   this.dataType  = dataType;
   this.isswcIn = isswcIn;
   this.ismodIn = ismodIn;

   createAndShowGUI();
}


public void setVal( Double val ){
   this.val = val;
}


 private void createAndShowGUI() {
     //Create and set up the window.
   this.jFrame = new JFrame();

   //this.jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   this.jFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

   this.jFrame.setTitle(  "Histogram Realtime-Val [ " +  this.name + " ]"  );

   Container contentpane = this.jFrame.getContentPane();

   Container panel = this.createChart();

   if( panel == null ){
      System.out.println( " panel == null ");
      System.exit(0);
   }else{
      System.out.println( " panel != null ");
   }

   contentpane.add( panel, BorderLayout.CENTER  );

   //グラフの描画準備

   this.jFrame.setSize(500,700);
     //this.jFrame.pack();
     this.jFrame.setVisible(true);
 }


 public static void main(String[] args) {
     //Schedule a job for the event-dispatching thread:
     //creating and showing this application's GUI.
   /*
   javax.swing.SwingUtilities.invokeLater( new Runnable() {
       public void run() {
           createAndShowGUI();
       }
   });
   */
 }
```

```
private JComponent createChart(){
   //
   dataset = createDataset();

   //
   if( this.graphType.equals( "8_Waterfall" )){
      this.chart = ChartFactory.createWaterfallChart(
            "Histogram [ "+this.name+" ]",   this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "1_棒積上げ３Ｄグラフ" )){
      this.chart = ChartFactory.createStackedBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "2_棒積上げグラフ" )){
      this.chart = ChartFactory.createStackedBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "3_折れ線グラフ" )){
      this.chart = ChartFactory.createLineChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "4_折れ線３Ｄグラフ" )){
      this.chart = ChartFactory.createLineChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "5_棒３Ｄグラフ" )){
      this.chart = ChartFactory.createBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "6_棒グラフ" )){
      this.chart = ChartFactory.createBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "7_面グラフ" )){
      this.chart = ChartFactory.createAreaChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else{
      System.out.println( this.moduleName + "graphType error!" );
   }

   ChartPanel cPanel = new ChartPanel( this.chart );

   return cPanel;
}


//row keys
private String series1 = null;
private String series2 = null;
private String series3 = null;
private String series4 = null;

private DefaultCategoryDataset createDataset(){
   //dataset
   this.dataset = new DefaultCategoryDataset();

   this.categoryNumMin = this.lowerLimit;
   this.categoryNumMax = this.upperLimit;
   this.range = ( this.upperLimit - this.lowerLimit ) / this.div;

   if( this.lowerLimit > this.upperLimit ){
      this.isReverse = true;
   }


   if( isswcIn ){
      if( ismodIn ){
         //row keys
         series1 = "COOL/on";
         series2 = "COOL/off";
```

```
      series3 = "HEAT/on";
      series4 = "HEAT/off";
   }else{
      //row keys
      series1 = "/on";
      series2 = "/off";
      series3 = "/on";
      series4 = "/off";
   }
}else{
   if( ismodIn ){
      //row keys
      series1 = "COOL/";
      series2 = "COOL/";
      series3 = "HEAT/";
      series4 = "HEAT/";
   }else{
      //row keys
      series1 = "/";
      series2 = "/";
      series3 = "/";
      series4 = "/";
   }
}

//column keys
if( this.isReverse ){
   this.categoryMin = "> "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" >";
}else{
   this.categoryMin = "< "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" <";
}

//count
count1Min = 0;
count1Max = 0;
count2Min = 0;
count2Max = 0;
count3Min = 0;
count3Max = 0;
count4Min = 0;
count4Max = 0;

dataset.addValue( count1Min, series1, categoryMin );
dataset.addValue( count2Min, series2, categoryMin );
dataset.addValue( count3Min, series3, categoryMin );
dataset.addValue( count4Min, series4, categoryMin );

this.category = new String[ this.div ];
this.count1 = new int[ this.div ];
this.count2 = new int[ this.div ];
this.count3 = new int[ this.div ];
this.count4 = new int[ this.div ];

//int k = 0;
for( int i= 0; i<this.div; i++ ){
   category[i] = ""+( this.categoryNumMin + this.range * ( i) )
   +" - "+( this.categoryNumMin + this.range * ( i + 1 ) );
   count1[i] = 0;
   count2[i] = 0;
   count3[i] = 0;
   count4[i] = 0;
   //
   dataset.addValue( count1[i], series1, category[i] );
   dataset.addValue( count2[i], series2, category[i] );
   dataset.addValue( count3[i], series3, category[i] );
   dataset.addValue( count4[i], series4, category[i] );
   //k++;
}

dataset.addValue( count1Max, series1, categoryMax );
```

```
    dataset.addValue( count2Max, series2, categoryMax );
    dataset.addValue( count1Max, series3, categoryMax );
    dataset.addValue( count2Max, series4, categoryMax );

    return dataset;
}


public void addData( int swc, int mod, Double valP ){

    //System.out.println( moduleName +"addData()");
    //System.out.println( moduleName + "■■isEventDispatchThread() addData ="
    //+ SwingUtilities.isEventDispatchThread() );

    final boolean isOnOff;

    String str = null;
    String stc = null;
    double count = 0.1;

    this.val = valP;
    // val

    this.dataValue = this.val.doubleValue();

    //
    if( isswcIn ){
        isOnOff = Airswc.isON( swc );
    }else{
        isOnOff = true;
    }

    if( ismodIn ){
    }else{
        mod = Airmod.onCOOL( mod );
    }

    if( isOnOff ){
        if( Airmod.isCOOL( mod )){
            //str = "COOL/on";
            str = this.series1;
            if( isReverse ){
                if( this.dataValue > categoryNumMin ){
                    count = ++count1Min;
                    stc = categoryMin;
                }else if( this.dataValue <= categoryNumMax ){
                    count = ++count1Max;
                    stc = categoryMax;
                }else{
                    for( int i = 0; i < div; i++ ){
                        if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                            count = ++count1[ i ];
                            stc = category[ i ];
                            break;
                        }
                    }
                }
            }else{
                if( this.dataValue < categoryNumMin ){
                    count = ++count1Min;
                    stc = categoryMin;
                }else if( this.dataValue >= categoryNumMax ){
                    count = ++count1Max;
                    stc = categoryMax;
                }else{
                    for( int i = 0; i < div; i++ ){
                        if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                            count = ++count1[ i ];
                            stc = category[ i ];
                            break;
                        }
                    }
```

```
            }
        }

    }else if( Airmod.isHEAT( mod )){
        //str = "HEAT/on";
        str = this.series3;
        if( isReverse ){
            if( this.dataValue > categoryNumMin ){
                count = --count3Min;
                stc = categoryMin;
            }else if( this.dataValue <= categoryNumMax ){
                count = --count3Max;
                stc = categoryMax;
            }else{
                for( int i = 0; i < div; i++ ){
                    if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                        count = --count3[ i ];
                        stc = category[ i ];
                        break;
                    }
                }
            }
        }else{
            if( this.dataValue < categoryNumMin ){
                count = --count3Min;
                stc = categoryMin;
            }else if( this.dataValue >= categoryNumMax ){
                count = --count3Max;
                stc = categoryMax;
            }else{
                for( int i = 0; i < div; i++ ){
                    if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                        count = --count3[ i ];
                        stc = category[ i ];
                        break;
                    }
                }
            }
        }

    }
}else{
    if( Airmod.isCOOL( mod )){
        //str = "COOL/off";
        str = this.series2;
        if( isReverse ){
            if( this.dataValue > categoryNumMin ){
                count = ++count2Min;
                stc = categoryMin;
            }else if( this.dataValue <= categoryNumMax ){
                count = ++count2Max;
                stc = categoryMax;
            }else{
                for( int i = 0; i < div; i++ ){
                    if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                        count = ++count2[ i ];
                        stc = category[ i ];
                        break;
                    }
                }
            }
        }else{
            if( this.dataValue < categoryNumMin ){
                count = ++count2Min;
                stc = categoryMin;
            }else if( this.dataValue >= categoryNumMax ){
                count = ++count2Max;
                stc = categoryMax;
            }else{
                for( int i = 0; i < div; i++ ){
                    if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                        count = ++count2[ i ];
```

```
                    stc = category[ i ];
                    break;
                }
            }
        }
    }
}else if( Airmod.isHEAT( mod )){
    //str = "HEAT/off";
    str = this.series4;
    if( isReverse ){
        if( this.dataValue > categoryNumMin ){
            count = --count4Min;
            stc = categoryMin;
        }else if( this.dataValue <= categoryNumMax ){
            count = --count4Max;
            stc = categoryMax;
        }else{
            for( int i = 0; i < div; i++ ){
                if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                    count = --count4[ i ];
                    stc = category[ i ];
                    break;
                }
            }
        }
    }else{
        if( this.dataValue < categoryNumMin ){
            count = --count4Min;
            stc = categoryMin;
        }else if( this.dataValue >= categoryNumMax ){
            count = --count4Max;
            stc = categoryMax;
        }else{
            for( int i = 0; i < div; i++ ){
                if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                    count = --count4[ i ];
                    stc = category[ i ];
                    break;
                }
            }
        }
    }
}

final double fcount = count;
final String rowKey = str;
final String columnKey = stc;

final Runnable worker = new Runnable() {
    public void run(){
        //System.out.println( moduleName +"worker run()");
        //System.out.println( moduleName + "■■isEventDispatchThread() worker ="
        //+ SwingUtilities.isEventDispatchThread() );
        if( rowKey != null && columnKey != null ){
            dataset.setValue( fcount, rowKey, columnKey );
        }
    }
};

Thread runThread = new Thread(){
    public void run() {
        //System.out.println( moduleName +"worker runThread()");
        //System.out.println( moduleName +"■■isEventDispatchThread() runThread ="
        //+ SwingUtilities.isEventDispatchThread() );
        try {
            SwingUtilities.invokeAndWait( worker );
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (InvocationTargetException e) {
            e.printStackTrace();
        }
```

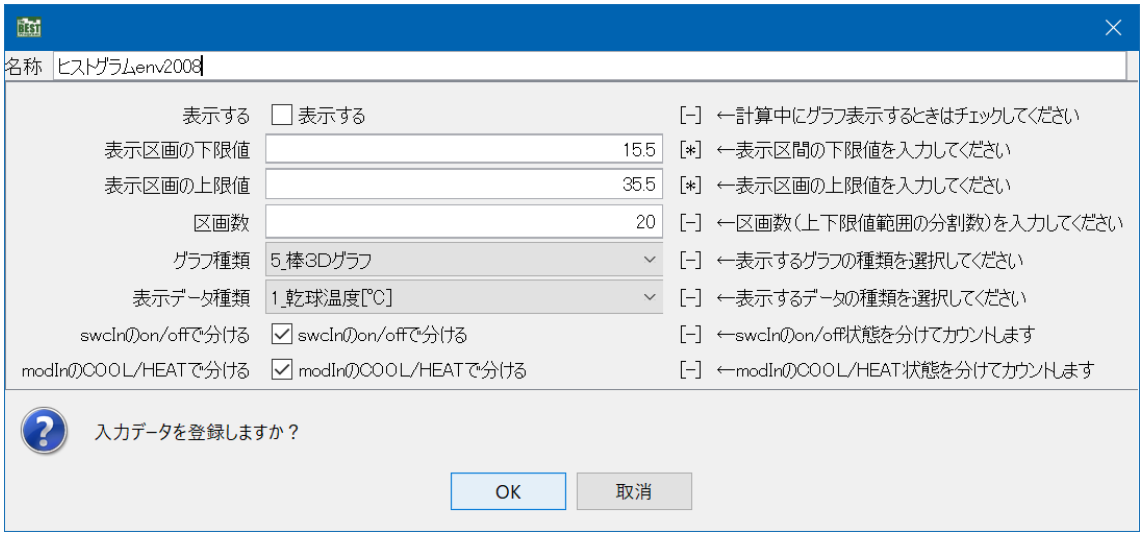
```
            }
        };

        runThread.start();
    }
}
```

「ヒストグラム env」（場所：設備 2015／グラフ計測 2015／）

モジュール名	ヒストグラム air
クラス	GraphRealtimeZoneEnvHistogramModule20080909 GraphRealtimeZoneEnvHistogramJFrame20080909

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、観察した BestAir 媒体の表示データ種類で指定したデータ（乾球温度など）について計算中にヒストグラムを作成するモジュールである。

室の env 媒体を発生させるために Zenv 接続 2015 モジュールあるいは ZSys 接続 201310 と併用し、L0_envOut ノードを観察接続する。表示するにチェックするだけでゾーンの環境（乾球温度、絶対湿度]、相対湿度、PMV、作用温度）状態をヒストグラム表示することができ、計算中に空調の良し悪しを判断することができる。

GraphRealtimeZoneEnvHistogramModule20080909 クラスがモジュールの本体で、実際にグラフを表示するのは GraphRealtimeZoneEnvHistogramJFrame20080909 クラスで行っている。

＊グラフ作成エンジンは JFreeChart を利用している。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]				
1	表示する	boolean	setVisible	FALSE	[-]				←計算中にグラフ表示するときはチェックしてください
2	表示区画の下限値	double	lowerLimit	15.5	[*]				←表示区間の下限値を入力してください
3	表示区画の上限値	double	upperLimit	35.5	[*]				←表示区画の上限値を入力してください
4	区画数	int	div	20	[-]				←区画数（上下限値範囲の分割数）を入力してください
5	グラフ種類	String	graphType	1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall	[-]				←表示するグラフの種類を選択してください
6	表示データ種類	String	dataType	1_乾球温度[℃]、2_絶対湿度[g/g']、3_相対湿度[%]、4_PMV[-]、5_作用温度[℃]	[-]				←表示するデータの種類を選択してください
7	swcIn の on/off で分ける	boolean	isswcIn	TRUE	[-]				←swcIn の on/off 状態を分けてカウントします
8	modIn の COOL/HEAT で分ける	boolean	ismodIn	TRUE	[-]				←modIn の COOL/HEAT 状態を分けてカウントします

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	発停信号入口	LO_swcIn	swcIn	-	制御	OnOff 信号	出口	OnOff 信号の入口
2	モード信号入口	LO_modIn	modIn	-	制御	mod 信号	入口	mod 信号の入口
3	空気観察	LO_envObs	envObs	-	状態	室環境	観察	室環境の観察
4								

（6）記録項目

No.	項目	出力内容	単位	分類
1				

このモジュールに記録項目はない。

（7）計算フロー・計算内容

・表示する

有効の時に、計算時にヒストグラムを表示する。

・表示区画の下限値、上限値、区画数

上限値と下限値の間を区画数で均等分割する。

これに下限値未満と上限値以上の区画を加えたもので表示する。

・グラフの種類

グラフの種類は次の８種類から選択する。

1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall

・swcln の on/off で分ける

例えば空調機への swcln を接続しておくと空調機の運転時と停止時とに分けて度数を集計する。

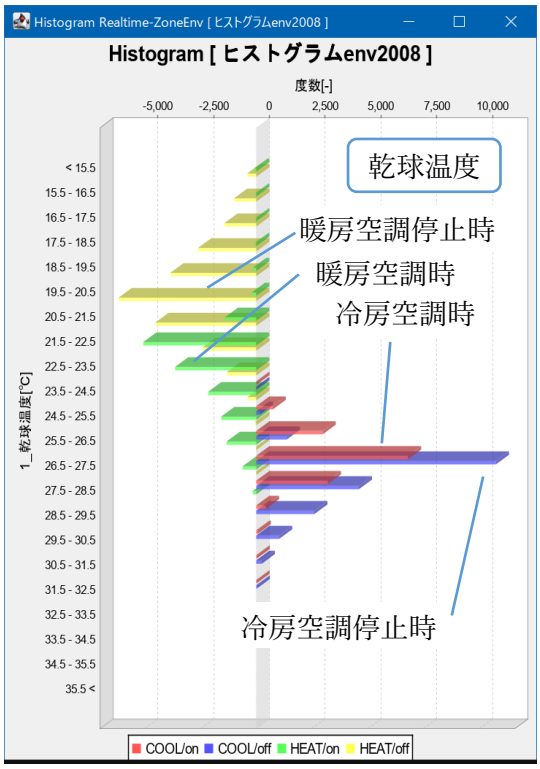
・modln の COOL/HEAT で分ける

例えば空調機への modln を接続しておくと冷房時と暖房時に分けて度数を集計する。

ゾーンの env について乾球温度と PMV の棒 3D ヒストグラムの表示例を示す。

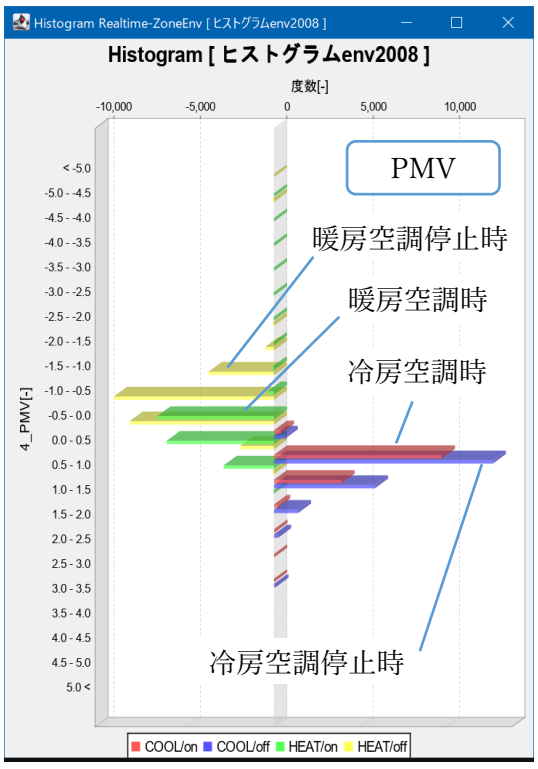
5 分間隔で年間計算（助走期間あり）したもので、度数は計算ステップ数である。

度数×計算時間間隔（5 分）÷60 で年間の時間数が求められる。



室の env の乾球温度の状態を描画した例

下限値 15.5℃、上限値 35.5℃、20 区画



室の env の PMV を描画した例

下限値-5、上限値+5、20 区画

（８）データ範囲と範囲外の取扱い

特になし。

参考ソース

■GraphRealtimeZoneEnvHistogramModule20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.Map;

import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.building.spaces.ZoneEnv;

/**
 * @author HIROSHI NINOMIYA /20080827 /20080908 /20080909
 * GraphRealtimeZoneEnvHistogramModule リアルタイムグラフ度数表示です
 *   ZoneEnv  に接続する  表示データ種類として
 *   ZoneEnv  に接続する  乾球温度、絶対湿度、相対湿度、PMV,作用温度
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 *
 * GraphRealtimeBarAirModule  を基に接続ノードを「観測/Obs」に変更
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 * isswcIn simodInを追加
 */
public class GraphRealtimeZoneEnvHistogramModule20080909 extends AbstractBestModule implements
      IBestMetaModule {

   private final String moduleName = "(GraphRealtimeZoneEnvHistogramModule20080909) ";

   //接続ノード
   //出入り口
   private final String S_NODE_envObs  = "L0_envObs"; //観測ZoneEnv

   //制御ノード
   private final String C_NODE_swcIn = "L1_swcIn";
   private final String C_NODE_modIn = "L1_modIn";

   //記録ノード

   //仕様
   private final String SPEC_name      = "名称";      //外部定義項目
   private final String SPEC_Visible = "表示する"; //描画する  true、false
   private final String SPEC_lowerLimit = "表示区画の下限値";//表示区画の下限値
   private final String SPEC_upperLimit = "表示区画の上限値";//表示区画の上限値
   private final String SPEC_div     = "区画数";   //区画数
   private final String SPEC_graphType = "グラフ種類";//グラフ種類
   private final String SPEC_dataType  = "表示データ種類";//表示するデータの種類
   // *   ZoneEnv  に接続する  乾球温度、絶対湿度、相対湿度、PMV,作用温度
   private final String SPEC_isswcIn = "swcInのon/offで分ける";//on/offでカウントを分ける
   private final String SPEC_ismodIn = "modInのCOOL/HEATで分ける";//COOL・HEATでカウントを分ける
   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

   //仕様など
   private StringBuffer message = new StringBuffer();    //メッセージ
   private String  name = null;  //機器名称
   private boolean setVisible = true;  //描画するか  どうか
   private double lowerLimit; //表示区画の下限値
   private double upperLimit; //表示区画の上限値
   private int div;     //区画数
   private String graphType = null; //表示す売るグラフの種類
```

```
private String dataType = null;  //表示するデータの種類
private boolean isswcIn = false;//on/offでカウントを分ける
private boolean ismodIn = false;//COOL・HEATでカウントを分ける
private boolean isRecord = false;//記録を有効とする=true

//接続熱媒など
private ZoneEnv envObs  = null;

private int swcIn;
private int modIn;

//グラフ描画データなど
private GraphRealtimeZoneEnvHistogramJFrame20080909 gEnvHistogram = null;


@Override
public void setProfile(BestSpecs spec) {
   if(spec == null) {
      return;
   }

   Map<String, String> map = spec.getSpec();
   if(map == null) {
      return;
   }

   //名称を取得
   if(null != map.get(this.SPEC_name)) {
      this.name = (String)map.get(this.SPEC_name);
   }else{
      System.out.println(this.moduleName + "(W)名称がありません");
      this.name = "InstrumentAir";
   }

   //Visibleを取得
   if(null != map.get(this.SPEC_Visible)) {
      this.setVisible = Boolean.parseBoolean( map.get(this.SPEC_Visible) );
   }else{
      System.out.println(this.moduleName + "(W)描画指定がありません");
      this.setVisible = true;
   }

   //lowerLimitを取得
   if(null != map.get(this.SPEC_lowerLimit)) {
      this.lowerLimit = Double.parseDouble( map.get(this.SPEC_lowerLimit) );
   }else{
      System.out.println(this.moduleName + "(W)表示区画の下限値がありません->=15");
      this.lowerLimit = 15;
   }


   //upperLimitを取得
   if(null != map.get(this.SPEC_upperLimit)) {
      this.upperLimit = Double.parseDouble( map.get(this.SPEC_upperLimit) );
   }else{
      System.out.println(this.moduleName + "(W)表示区画の上限値がありません->=35");
      this.upperLimit = 35;
   }

   //divを取得
   if(null != map.get(this.SPEC_div)) {
      this.div = Integer.parseInt( map.get(this.SPEC_div) );
   }else{
      System.out.println(this.moduleName + "(W)区画数がありません->=20");
      this.div = 20;
   }

   //graphTypeを取得
   if(null != map.get(this.SPEC_graphType)) {
      this.graphType = map.get(this.SPEC_graphType);
   }else{
      System.out.println(this.moduleName + "(W)グラフ種類がありません");
```

```
            this.graphType = "5_棒３Ｄグラフ";
        }

        //dataTypeを取得
        if(null != map.get(this.SPEC_dataType)) {
            this.dataType = map.get(this.SPEC_dataType);
        }else{
            System.out.println(this.moduleName + "(W)表示するデータ種類がありません");
            this.dataType = "1_乾球温度[℃]";
        }

        //isswcIn
        if(null != map.get(this.SPEC_isswcIn)) {
            this.isswcIn = Boolean.parseBoolean( map.get(this.SPEC_isswcIn) );
        }else{
            System.out.println(this.moduleName + "(W)isswcIn指定がありません->=false");
            this.isswcIn = false;
        }

        //ismodIn
        if(null != map.get(this.SPEC_ismodIn)) {
            this.ismodIn = Boolean.parseBoolean( map.get(this.SPEC_ismodIn) );
        }else{
            System.out.println(this.moduleName + "(W)ismodIn指定がありません->=false");
            this.ismodIn = false;
        }

        //isRecordを取得
        if(null != map.get(this.SPEC_isRecord)) {
            this.isRecord = Boolean.parseBoolean( map.get(this.SPEC_isRecord) );
        }else{
            System.out.println(this.moduleName + "(W)記録指定がありません->=false");
            this.isRecord = false;
        }
    }


    public void initialize(IBestStateMessage stateNodes,
            IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
            IBestDomainSchedule schedule) {
        //状態ノードを取得
        super.sm = stateNodes;

        //制御ノードを取得
        super.cm = commandNodes;

        //記録ノードを取得
        super.rm = recordNodes;


        //接続ノード　出口

        //接続ノード　入口チェック
        //envObs
        if( super.sm.getState( super.getConnectionNode( this.S_NODE_envObs )) != null ){
            this.envObs
            = (ZoneEnv) super.sm.getState( super.getConnectionNode( this.S_NODE_envObs));
        }else{
            this.envObs = new ZoneEnv();
            super.sm.setState( super.getConnectionNode( this.S_NODE_envObs ), this.envObs );
            message.append( "(W)観測ZoneEnvの接続なし→作成");
            System.out.println( this.moduleName + ">>Warning<< envObs is null !!" );
        }


        //制御ノード

        //グラフの描画準備
        if( this.setVisible ){
            //
            gEnvHistogram = new GraphRealtimeZoneEnvHistogramJFrame20080909( this.name,
                    this.lowerLimit, this.upperLimit, this.div, this.graphType, this.dataType, this.isswcIn, this.ismodIn );
```

```
    }

}

public void outputs() {
    if(super.sm == null || super.cm == null ){
        return;
    }

    //
    this.envObs
    = (ZoneEnv)super.sm.getState(super.getConnectionNode(this.S_NODE_envObs));
    this.swcIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ));
    this.modIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn ));

    //ノードに設定

    //新しいデータを加える
    if( this.setVisible ){
        gEnvHistogram.addData( this.swcIn, this.modIn, this.envObs );
    }

    //記録

    if( this.isRecord  && super.rm != null ){
        this.record();
    }

    message.setLength(0);
}

/**
 * 記録
 */
private void record(){
    if( CheckPrintModule.isPrintMessage ){
    }

    //if( CheckPrintModule.isPrintEnergy ){
    //}

    //if( CheckPrintModule.isPrintLoad ){
    //}

    //if( CheckPrintModule.isPrintStateOut ){
    //}

    //if( CheckPrintModule.isPrintStateMy ){
    //}

    //if( CheckPrintModule.isPrintStateIn ){
    //}
}



public void update() {

}


public Object viewInternal(TestCommand cmd) {
    ArrayList<Object> result = new ArrayList<Object>();
    result.add(super.sm);
    result.add(super.cm);
    result.add(super.rm);

    //外部定義項目

    //記録ノードに設定する値
    if(envObs != null) {
        result.add(new Double(this.envObs.getTa())); //温度
```

```
            result.add(new Double(this.envObs.getXa())); //絶対湿度
            result.add(new Double(this.envObs.getRh())); //相対湿度
            result.add(new Double(this.envObs.getPMV()));   //PMV
            result.add(new Double(this.envObs.getOT())); //作用温度
        }else{
            result.add(new Double(0.0));
            result.add(new Double(0.0));
            result.add(new Double(0.0));
            result.add(new Double(0.0));
            result.add(new Double(0.0));
        }

        result.add(super.transferMapCommand);
        result.add(super.transferMapRecord);
        result.add(super.transferMapState);
        return result;
    }
}
```

■GraphRealtimeZoneEnvHistogramJFrame20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.awt.BorderLayout;
import java.awt.Container;
import java.lang.reflect.InvocationTargetException;

import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

import jp.or.ibec.best.domain.building.spaces.ZoneEnv;

/**
 * @author HIROSHI NINOMIYA /20080827 /20080908 /20080909
 * GraphRealtimeZoneEnvHistogramJFrame リアルタイムグラフ表示部分です
 *   ZoneEnv  に接続する  表示データ種類として
 *   ZoneEnv  に接続する  乾球温度、絶対湿度、相対湿度、PMV,作用温度
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 */
public class GraphRealtimeZoneEnvHistogramJFrame20080909 {

   private final String moduleName = "(GraphRealtimeZoneEnvHistogramJFrame20080909) ";

   private JFrame jFrame = null;

   private JFreeChart chart = null;

   //仕様など
   private String  name = null;  //機器名称
   private double lowerLimit;//表示区画の下限値
   private double upperLimit;//表示区画の上限値
   private double range;//１区画の幅
   private int div;     //片側区画数
   private String graphType = null; //表示す売るグラフの種類
   private String dataType = null;  //表示するデータの種類
   private boolean isswcIn;
   private boolean ismodIn;

   //グラフ描画データなど
   private DefaultCategoryDataset dataset = null;
   private String categoryMin = null;
   private String categoryMax = null;
   private String[] category;

   private double categoryNumMin;
   private double categoryNumMax;
   private int count1Min;
   private int count1Max;
   private int[] count1;
   private int count2Min;
   private int count2Max;
   private int[] count2;
   private int count3Min;
   private int count3Max;
   private int[] count3;
   private int count4Min;
   private int count4Max;
   private int[] count4;
   double dataValue;
```

```
private boolean isReverse = false;//lowerLimit  とupperLimitが逆転している=true

private ZoneEnv env = new ZoneEnv();


public GraphRealtimeZoneEnvHistogramJFrame20080909() {
   this( "GraphRealtimeZoneEnvHistogramJFrame", 15, 35, 20, "5_棒３Ｄグラフ", "1_乾球温度[℃]", true, true );
}


public GraphRealtimeZoneEnvHistogramJFrame20080909( String name, double lowerLimit, double upperLimit,
      int div, String graphType, String dataType, boolean isswcIn, boolean ismodIn ) {
   this.name = name;
   this.lowerLimit = lowerLimit;
   this.upperLimit = upperLimit;
   this.div    = div;
   this.graphType = graphType;
   this.dataType  = dataType;
   this.isswcIn = isswcIn;
   this.ismodIn = ismodIn;

   createAndShowGUI();
}


public void setEnv( ZoneEnv env ){
   this.env = env;
}


 private void createAndShowGUI() {
     //Create and set up the window.
   this.jFrame = new JFrame();

   //this.jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   this.jFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

   this.jFrame.setTitle(  "Histogram Realtime-ZoneEnv [ " +  this.name + " ]"  );

   Container contentpane = this.jFrame.getContentPane();

   Container panel = this.createChart();

   if( panel == null ){
      System.out.println( " panel == null ");
      System.exit(0);
   }else{
      System.out.println( " panel != null ");
   }

   contentpane.add( panel, BorderLayout.CENTER  );

   //グラフの描画準備

   this.jFrame.setSize(500,700);
     //this.jFrame.pack();
     this.jFrame.setVisible(true);
 }


 public static void main(String[] args) {
     //Schedule a job for the event-dispatching thread:
     //creating and showing this application's GUI.
   /*
   javax.swing.SwingUtilities.invokeLater( new Runnable() {
       public void run() {
           createAndShowGUI();
       }
   });
   */
 }
```

```
private JComponent createChart(){
   //
   dataset = createDataset();

   //
   if( this.graphType.equals( "8_Waterfall" )){
      this.chart = ChartFactory.createWaterfallChart(
            "Histogram [ "+this.name+" ]",    this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "1_棒積上げ３Ｄグラフ" )){
      this.chart = ChartFactory.createStackedBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "2_棒積上げグラフ" )){
      this.chart = ChartFactory.createStackedBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "3_折れ線グラフ" )){
      this.chart = ChartFactory.createLineChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "4_折れ線３Ｄグラフ" )){
      this.chart = ChartFactory.createLineChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "5_棒３Ｄグラフ" )){
      this.chart = ChartFactory.createBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "6_棒グラフ" )){
      this.chart = ChartFactory.createBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "7_面グラフ" )){
      this.chart = ChartFactory.createAreaChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else{
      System.out.println( this.moduleName + "graphType error!" );
   }

   ChartPanel cPanel = new ChartPanel( this.chart );

   return cPanel;
}

//row keys
private String series1 = null;
private String series2 = null;
private String series3 = null;
private String series4 = null;

private DefaultCategoryDataset createDataset(){
   //dataset
   this.dataset = new DefaultCategoryDataset();

   this.categoryNumMin = this.lowerLimit;
   this.categoryNumMax = this.upperLimit;
   this.range = ( this.upperLimit - this.lowerLimit ) / this.div;

   if( this.lowerLimit > this.upperLimit ){
      this.isReverse = true;
   }


   if( isswcIn ){
      if( ismodIn ){
         //row keys
         series1 = "COOL/on";
         series2 = "COOL/off";
```

```
      series3 = "HEAT/on";
      series4 = "HEAT/off";
   }else{
      //row keys
      series1 = "/on";
      series2 = "/off";
      series3 = "/on";
      series4 = "/off";
   }
}else{
   if( ismodIn ){
      //row keys
      series1 = "COOL/";
      series2 = "COOL/";
      series3 = "HEAT/";
      series4 = "HEAT/";
   }else{
      //row keys
      series1 = "/";
      series2 = "/";
      series3 = "/";
      series4 = "/";
   }
}

//column keys
if( this.isReverse ){
   this.categoryMin = "> "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" >";
}else{
   this.categoryMin = "< "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" <";
}

//count
count1Min = 0;
count1Max = 0;
count2Min = 0;
count2Max = 0;
count3Min = 0;
count3Max = 0;
count4Min = 0;
count4Max = 0;

dataset.addValue( count1Min, series1, categoryMin );
dataset.addValue( count2Min, series2, categoryMin );
dataset.addValue( count3Min, series3, categoryMin );
dataset.addValue( count4Min, series4, categoryMin );

this.category = new String[ this.div ];
this.count1 = new int[ this.div ];
this.count2 = new int[ this.div ];
this.count3 = new int[ this.div ];
this.count4 = new int[ this.div ];

//int k = 0;
for( int i= 0; i<this.div; i++ ){
   category[i] = ""+( this.categoryNumMin + this.range * ( i) )
   +" - "+( this.categoryNumMin + this.range * ( i + 1 ) );
   count1[i] = 0;
   count2[i] = 0;
   count3[i] = 0;
   count4[i] = 0;
   //
   dataset.addValue( count1[i], series1, category[i] );
   dataset.addValue( count2[i], series2, category[i] );
   dataset.addValue( count3[i], series3, category[i] );
   dataset.addValue( count4[i], series4, category[i] );
   //k++;
}

dataset.addValue( count1Max, series1, categoryMax );
```

```
        dataset.addValue( count2Max, series2, categoryMax );
        dataset.addValue( count1Max, series3, categoryMax );
        dataset.addValue( count2Max, series4, categoryMax );

        return dataset;
    }


    public void addData( int swc, int mod, ZoneEnv envP ){

        //System.out.println( moduleName +"addData()");
        //System.out.println( moduleName + "■■isEventDispatchThread() addData ="
        //+ SwingUtilities.isEventDispatchThread() );

        final boolean isOnOff;

        String str = null;
        String stc = null;
        double count = 0.1;

        this.env = envP;
        //       乾球温度、絶対湿度、相対湿度、PMV、作用温度のトレンドグラフ

        if( this.dataType.equals( "1_乾球温度[℃]")){
            this.dataValue = this.env.getTa();
        }else if(this.dataType.equals( "2_絶対湿度[g/g']")){
            this.dataValue = this.env.getXa();
        }else if(this.dataType.equals( "3_相対湿度[%]")){
            this.dataValue = this.env.getRh();
        }else if(this.dataType.equals( "4_PMV[-]")){
            this.dataValue = this.env.getPMV();
        }else if(this.dataType.equals( "5_作用温度[℃]")){
            this.dataValue = this.env.getOT();
        }else{
            this.dataValue = this.env.getTa();
        }

        //
        if( isswcIn ){
            isOnOff = Airswc.isON( swc );
        }else{
            isOnOff = true;
        }

        if( ismodIn ){
        }else{
            mod = Airmod.onCOOL( mod );
        }

        if( isOnOff ){
            if( Airmod.isCOOL( mod )){
                //str = "COOL/on";
                str = this.series1;
                if( isReverse ){
                    if( this.dataValue > categoryNumMin ){
                        count = ++count1Min;
                        stc = categoryMin;
                    }else if( this.dataValue <= categoryNumMax ){
                        count = ++count1Max;
                        stc = categoryMax;
                    }else{
                        for( int i = 0; i < div; i++ ){
                            if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                                count = ++count1[ i ];
                                stc = category[ i ];
                                break;
                            }
                        }
                    }
                }else{
                    if( this.dataValue < categoryNumMin ){
                        count = ++count1Min;
```

```
            stc = categoryMin;
         }else if( this.dataValue >= categoryNumMax ){
            count = ++count1Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                  count = ++count1[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }

   }else if( Airmod.isHEAT( mod )){
      //str = "HEAT/on";
      str = this.series3;
      if( isReverse ){
         if( this.dataValue > categoryNumMin ){
            count = --count3Min;
            stc = categoryMin;
         }else if( this.dataValue <= categoryNumMax ){
            count = --count3Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                  count = --count3[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }else{
         if( this.dataValue < categoryNumMin ){
            count = --count3Min;
            stc = categoryMin;
         }else if( this.dataValue >= categoryNumMax ){
            count = --count3Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                  count = --count3[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }

   }
}else{
   if( Airmod.isCOOL( mod )){
      //str = "COOL/off";
      str = this.series2;
      if( isReverse ){
         if( this.dataValue > categoryNumMin ){
            count = ++count2Min;
            stc = categoryMin;
         }else if( this.dataValue <= categoryNumMax ){
            count = ++count2Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                  count = ++count2[ i ];
                  stc = category[ i ];
                  break;
               }
            }
```

```
            }
        }else{
            if( this.dataValue < categoryNumMin ){
                count = ++count2Min;
                stc = categoryMin;
            }else if( this.dataValue >= categoryNumMax ){
                count = ++count2Max;
                stc = categoryMax;
            }else{
                for( int i = 0; i < div; i++ ){
                    if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                        count = ++count2[ i ];
                        stc = category[ i ];
                        break;
                    }
                }
            }
        }
    }else if( Airmod.isHEAT( mod )){
        //str = "HEAT/off";
        str = this.series4;
        if( isReverse ){
            if( this.dataValue > categoryNumMin ){
                count = --count4Min;
                stc = categoryMin;
            }else if( this.dataValue <= categoryNumMax ){
                count = --count4Max;
                stc = categoryMax;
            }else{
                for( int i = 0; i < div; i++ ){
                    if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                        count = --count4[ i ];
                        stc = category[ i ];
                        break;
                    }
                }
            }
        }else{
            if( this.dataValue < categoryNumMin ){
                count = --count4Min;
                stc = categoryMin;
            }else if( this.dataValue >= categoryNumMax ){
                count = --count4Max;
                stc = categoryMax;
            }else{
                for( int i = 0; i < div; i++ ){
                    if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                        count = --count4[ i ];
                        stc = category[ i ];
                        break;
                    }
                }
            }
        }
    }
}

final double fcount = count;
final String rowKey = str;
final String columnKey = stc;

final Runnable worker = new Runnable() {
    public void run(){
        //System.out.println( moduleName +"worker run()");
        //System.out.println( moduleName + "■■isEventDispatchThread() worker ="
        //+ SwingUtilities.isEventDispatchThread() );
        if( rowKey != null && columnKey != null ){
            dataset.setValue( fcount, rowKey, columnKey );
        }
    }
};
```

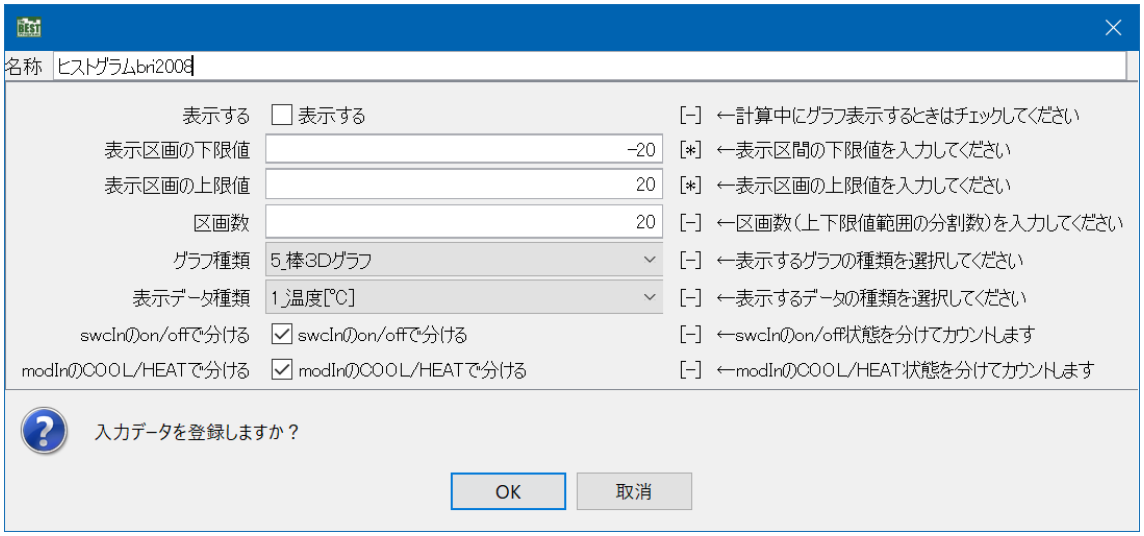
```
        Thread runThread = new Thread(){
            public void run() {
                //System.out.println( moduleName +"worker runThread()");
                //System.out.println( moduleName +"■■isEventDispatchThread() runThread ="
                //+ SwingUtilities.isEventDispatchThread() );
                try {
                    SwingUtilities.invokeAndWait( worker );
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } catch (InvocationTargetException e) {
                    e.printStackTrace();
                }
            }
        };

        runThread.start();
    }
}
```

「ヒストグラム bri」（場所：設備 2015／グラフ計測 2015／）

モジュール名	ヒストグラム wat
クラス	GraphRealtimeBrineHistogramModule20080909 GraphRealtimeBrineHistogramJFrame20080909

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、観察した BestBrine 媒体の表示データ種類で指定したデータ（温度など）について計算中にヒストグラムを作成するモジュールである。

表示したいモジュールの L0_briOut、L0_briIn または L0_briObs に本モジュールの L0_briObs ノードを接続し、表示するにチェックするとヒストグラムを表示することができ、計算中に良し悪しを判断することができる。

GraphRealtimeBrineHistogramModule20080909 クラスがモジュールの本体で、実際にグラフを表示するのは GraphRealtimeBrineHistogramJFrame20080909 クラスで行っている。

＊グラフ作成エンジンは JFreeChart を利用している。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図1における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]				
1	表示する	boolean	setVisible	FALSE	[-]				←計算中にグラフ表示するときはチェックしてください
2	表示区画の下限値	Double	lowerLimit	-20	[*]				←表示区間の下限値を入力してください
3	表示区画の上限値	Double	upperLimit	20	[*]				←表示区画の上限値を入力してください
4	区画数	int	div	15	[-]				←区画数（上下限値範囲の分割数）を入力してください
5	グラフ種類	String	graphType	1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall	[-]				←表示するグラフの種類を選択してください
6	表示データ種類	String	dataType	1_温度[℃]、2_質量流量[g/s]	[-]				←表示するデータの種類を選択してください
7	swcIn の on/off で分ける	boolean	isswcIn	TRUE	[-]				←swcIn の on/off 状態を分けてカウントします
8	modIn の COOL/HEAT で分ける	boolean	ismodIn	TRUE	[-]				←modIn の COOL/HEAT 状態を分けてカウントします

（5）シーケンス接続（図1における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	発停信号入口	LO_swcIn	swcIn	-	制御	OnOff 信号	出口	OnOff 信号の入口
2	モード信号入口	LO_modIn	modIn	-	制御	mod 信号	入口	mod 信号の入口
3	ブライン観察	LO_briObs	briObs	-	状態	ブライン	観察	ブラインの観察
4								

（6）記録項目

No.	項目	出力内容	単位	分類
1				

このモジュールに記録項目はない。

（7）計算フロー・計算内容

・表示する

有効の時に、計算時にヒストグラムを表示する。

・表示区画の下限値、上限値、区画数

上限値と下限値の間を区画数で均等分割する。

これに下限値未満と上限値以上の区画を加えたもので表示する。

・グラフの種類

グラフの種類は次の８種類から選択する。

1_棒積上げ３Ｄグラフ、2_棒積上げグラフ、3_折れ線グラフ、4_折れ線３Ｄグラフ、5_棒３Ｄグラフ、6_棒グラフ、7_面グラフ、8_Waterfall

・swcln の on/off で分ける

例えば熱源への swcln を接続しておくと空調機の運転時と停止時とに分けて度数を集計する。

・modln の COOL/HEAT で分ける

例えば熱源への modln を接続しておくと冷房時と暖房時に分けて度数を集計する。

表示例は省略する。

表示形態はヒストグラム wat と同様である。

（8）データ範囲と範囲外の取扱い

特になし。

参考ソース

■GraphRealtimeBrineHistogramModule20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.Map;

import jp.or.ibec.best.DO.BestBrine;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;
import jp.or.ibec.best.dk.test.TestCommand;

/**
 * @author HIROSHI NINOMIYA /20080827 /20080908
 * GraphRealtimeBrineHistogramModule20080827 リアルタイムグラフ度数表示です
 *   BestBrine  に接続する  表示データ種類として
 *     温度、質量流量
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 *
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 * isswcIn simodInを追加
 */
public class GraphRealtimeBrineHistogramModule20080909 extends AbstractBestModule implements
      IBestMetaModule {

   private final String moduleName = "(GraphRealtimeBrineHistogramModule20080909) ";

   //接続ノード
   //出入り口
   private final String S_NODE_briObs  = "L0_briObs"; //観測Brine

   //制御ノード
   private final String C_NODE_swcIn = "L1_swcIn";
   private final String C_NODE_modIn = "L1_modIn";

   //記録ノード

   //仕様
   private final String SPEC_name      = "名称";      //外部定義項目
   private final String SPEC_Visible = "表示する"; //描画する  true、false
   private final String SPEC_lowerLimit  = "表示区画の下限値"; //表示区画の下限値
   private final String SPEC_upperLimit  = "表示区画の上限値"; //表示区画の上限値
   private final String SPEC_div     = "区画数";   //区画数
   private final String SPEC_graphType = "グラフ種類";//グラフ種類
   private final String SPEC_dataType  = "表示データ種類";//表示するデータの種類
   //       温度
   //       質量流量
   private final String SPEC_isswcIn = "swcInのon/offで分ける";//on/offでカウントを分ける
   private final String SPEC_ismodIn = "modInのCOOL/HEATで分ける";//COOL・HEATでカウントを分ける
   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

   //仕様など
   private StringBuffer message = new StringBuffer();    //メッセージ
   private String  name = null;  //機器名称
   private boolean setVisible = true;  //描画するか  どうか
   private double lowerLimit;//表示区画の下限値
   private double upperLimit;//表示区画の上限値
   private int div;//区画数
   private String graphType = null; //表示す売るグラフの種類
```

```
private String dataType = null;  //表示するデータの種類
private boolean isswcIn = false;//on/offでカウントを分ける
private boolean ismodIn = false;//COOL・HEATでカウントを分ける
private boolean isRecord = false;//記録を有効とする=true

//接続熱媒など
private BestBrine briObs  = null;

private int swcIn;
private int modIn;

//グラフ描画データなど
private GraphRealtimeBrineHistogramJFrame20080909 gBrineHistogram = null;


@Override
public void setProfile(BestSpecs spec) {
   if(spec == null) {
      return;
   }

   Map<String, String> map = spec.getSpec();
   if(map == null) {
      return;
   }

   //名称を取得
   this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

   //Visibleを取得
   this.setVisible = spec.getSpecValue( this.SPEC_Visible, true );

   //lowerLimitを取得
   this.lowerLimit = spec.getSpecValue( this.SPEC_lowerLimit, 0. );

   //upperLimitを取得
   this.upperLimit = spec.getSpecValue( this.SPEC_upperLimit, 10. );

   //divを取得
   this.div = spec.getSpecValue( this.SPEC_div, 10 );

   //graphTypeを取得
   this.graphType = spec.getSpecValue( this.SPEC_graphType, "5_棒３Ｄグラフ" );

   //dataTypeを取得
   this.dataType = spec.getSpecValue( this.SPEC_dataType, "1_温度[℃]" );

   //isswcIn
   this.isswcIn = spec.getSpecValue( this.SPEC_isswcIn, false );

   //ismodIn
   this.ismodIn = spec.getSpecValue( this.SPEC_ismodIn, false );

   //isRecordを取得
   this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

}


public void initialize(IBestStateMessage stateNodes,
      IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
      IBestDomainSchedule schedule) {
   //状態ノードを取得
   super.sm = stateNodes;

   //制御ノードを取得
   super.cm = commandNodes;

   //記録ノードを取得
   super.rm = recordNodes;
```

```
    //接続ノード　出口

    //接続ノード　入口チェック
    //briObs
    if( super.sm.getState( super.getConnectionNode( this.S_NODE_briObs )) != null ){
       this.briObs
       = (BestBrine) super.sm.getState( super.getConnectionNode( this.S_NODE_briObs));
    }else{
       this.briObs = new BestBrine();
       super.sm.setState( super.getConnectionNode( this.S_NODE_briObs ), this.briObs );
       message.append( "(W)観測Brineの接続なし→作成");
       System.out.println( this.moduleName + ">>Warning<< briObs is null !!" );
    }


    //制御ノード

    //グラフの描画準備
    if( this.setVisible ){
       //
       gBrineHistogram = new GraphRealtimeBrineHistogramJFrame20080909( this.name,
             this.lowerLimit, this.upperLimit, this.div, this.graphType, this.dataType, this.isswcIn, this.ismodIn );
    }

 }

 public void outputs() {
    if(super.sm == null || super.cm == null ){
       return;
    }

    //
    this.briObs
    = (BestBrine)super.sm.getState(super.getConnectionNode(this.S_NODE_briObs));
    this.swcIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ));
    this.modIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn ));

    //ノードに設定

    //新しいデータを加える
    if( this.setVisible ){
       gBrineHistogram.addData( this.swcIn, this.modIn, this.briObs );
    }

    //記録

    if( this.isRecord  && super.rm != null ){
       this.record();
    }

    message.setLength(0);
 }

 /**
  * 記録
  */
 private void record(){
    if( CheckPrintModule.isPrintMessage ){
    }

    //if( CheckPrintModule.isPrintEnergy ){
    //}

    //if( CheckPrintModule.isPrintLoad ){
    //}

    //if( CheckPrintModule.isPrintStateOut ){
    //}

    //if( CheckPrintModule.isPrintStateMy ){
    //}
```

```
        //if( CheckPrintModule.isPrintStateIn ){
        //}
    }

    public void update() {

    }


    public Object viewInternal(TestCommand cmd) {
        ArrayList<Object> result = new ArrayList<Object>();
        result.add(super.sm);
        result.add(super.cm);
        result.add(super.rm);

        //外部定義項目

        //記録ノードに設定する値
        if(briObs != null) {
            result.add(new Double(this.briObs.getTemp()));     //温度
            result.add(new Double(this.briObs.getFlowRate())); //質量流量
        }else{
            result.add(new Double(0.0));
            result.add(new Double(0.0));
        }

        result.add(super.transferMapCommand);
        result.add(super.transferMapRecord);
        result.add(super.transferMapState);
        return result;
    }
}
```

■GraphRealtimeBrineHistogramJFrame20080909

```
package jp.or.ibec.best.domain.sample.air;

import java.awt.BorderLayout;
import java.awt.Container;
import java.lang.reflect.InvocationTargetException;

import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;

import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.data.category.DefaultCategoryDataset;

import jp.or.ibec.best.DO.BestBrine;

/**
 * @author HIROSHI NINOMIYA /20080827 /20080908 /20080909
 * GraphRealtimeBrineHistogramJFrame リアルタイムグラフ表示部分です
 *   BestBrine  に接続する  表示データ種類として
 *     温度、質量流量
 *   のヒストグラムを作成
 *   グラフ種類は下記のとおり
 *     棒、棒３Ｄ、棒積上げ、棒積上げ３Ｄ、折れ線、折れ線３Ｄ、面、Waterfall
 *   表示中心値、区画幅、片側区画数、グラフ種類、表示データ種類（上記）を指定できる
 * 区画の上下限を指定する方法に変更
 * 記録の有効無効判断を追加
 */
public class GraphRealtimeBrineHistogramJFrame20080909 {

   private final String moduleName = "(GraphRealtimeBrineHistogramJFrame20080909) ";

   private JFrame jFrame = null;

   private JFreeChart chart = null;

   //仕様など
   private String  name = null;  //機器名称
   private double lowerLimit;//表示区画の下限値
   private double upperLimit;//表示区画の上限値
   private double range;      //１区画の幅
   private int div;     //区画数
   private String graphType = null; //表示す売るグラフの種類
   private String dataType = null;  //表示するデータの種類
   private boolean isswcIn;
   private boolean ismodIn;

   //グラフ描画データなど
   private DefaultCategoryDataset dataset = null;
   private String categoryMin = null;
   private String categoryMax = null;
   private String[] category;

   private double categoryNumMin;
   private double categoryNumMax;
   private int count1Min;
   private int count1Max;
   private int[] count1;
   private int count2Min;
   private int count2Max;
   private int[] count2;
   private int count3Min;
   private int count3Max;
   private int[] count3;
   private int count4Min;
   private int count4Max;
   private int[] count4;
   double dataValue;
```

```
private boolean isReverse = false;//lowerLimit  とupperLimitが逆転している=true

private BestBrine brine = new BestBrine();


public GraphRealtimeBrineHistogramJFrame20080909() {
   this( "GraphRealtimeBrineHistogramJFrame", 0, 10, 10, "5_棒３Ｄグラフ", "1_温度[℃]", true, true  );
}


public GraphRealtimeBrineHistogramJFrame20080909( String name, double lowerLimit, double upperLimit,
      int div, String graphType, String dataType, boolean isswcIn, boolean ismodIn ) {
   this.name = name;
   this.lowerLimit = lowerLimit;
   this.upperLimit = upperLimit;
   this.div    = div;
   this.graphType = graphType;
   this.dataType  = dataType;
   this.isswcIn = isswcIn;
   this.ismodIn = ismodIn;

   createAndShowGUI();
}


public void setWater( BestBrine brine ){
   this.brine = brine;
}


 private void createAndShowGUI() {
     //Create and set up the window.
   this.jFrame = new JFrame();

   //this.jFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   this.jFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

   this.jFrame.setTitle(  "Histogram Realtime-Brine [ " +  this.name + " ]"  );

   Container contentpane = this.jFrame.getContentPane();

   Container panel = this.createChart();

   if( panel == null ){
      System.out.println( " panel == null ");
      System.exit(0);
   }else{
      System.out.println( " panel != null ");
   }

   contentpane.add( panel, BorderLayout.CENTER  );

   //グラフの描画準備

   this.jFrame.setSize(500,700);
     //this.jFrame.pack();
     this.jFrame.setVisible(true);
 }


 public static void main(String[] args) {
     //Schedule a job for the event-dispatching thread:
     //creating and showing this application's GUI.
   /*
   javax.swing.SwingUtilities.invokeLater( new Runnable() {
       public void run() {
           createAndShowGUI();
       }
   });
   */
 }
```

```
private JComponent createChart(){
   //
   dataset = createDataset();

   //
   if( this.graphType.equals( "8_Waterfall" )){
      this.chart = ChartFactory.createWaterfallChart(
            "Histogram [ "+this.name+" ]",    this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "1_棒積上げ３Ｄグラフ" )){
      this.chart = ChartFactory.createStackedBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "2_棒積上げグラフ" )){
      this.chart = ChartFactory.createStackedBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "3_折れ線グラフ" )){
      this.chart = ChartFactory.createLineChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "4_折れ線３Ｄグラフ" )){
      this.chart = ChartFactory.createLineChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "5_棒３Ｄグラフ" )){
      this.chart = ChartFactory.createBarChart3D(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "6_棒グラフ" )){
      this.chart = ChartFactory.createBarChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else if( this.graphType.equals( "7_面グラフ" )){
      this.chart = ChartFactory.createAreaChart(
            "Histogram [ "+this.name+" ]", this.dataType, "度数[-]",
            dataset, PlotOrientation.HORIZONTAL, true, true, false);
   }else{
      System.out.println( this.moduleName + "graphType error!" );
   }

   ChartPanel cPanel = new ChartPanel( this.chart );

   return cPanel;
}


//row keys
private String series1 = null;
private String series2 = null;
private String series3 = null;
private String series4 = null;

private DefaultCategoryDataset createDataset(){
   //dataset
   this.dataset = new DefaultCategoryDataset();

   this.categoryNumMin = this.lowerLimit;
   this.categoryNumMax = this.upperLimit;
   this.range = ( this.upperLimit - this.lowerLimit ) / this.div;

   if( this.lowerLimit > this.upperLimit ){
      this.isReverse = true;
   }


   if( isswcIn ){
      if( ismodIn ){
         //row keys
         series1 = "COOL/on";
```

```
      series2 = "COOL/off";
      series3 = "HEAT/on";
      series4 = "HEAT/off";
   }else{
      //row keys
      series1 = "/on";
      series2 = "/off";
      series3 = "/on";
      series4 = "/off";
   }
}else{
   if( ismodIn ){
      //row keys
      series1 = "COOL/";
      series2 = "COOL/";
      series3 = "HEAT/";
      series4 = "HEAT/";
   }else{
      //row keys
      series1 = "/";
      series2 = "/";
      series3 = "/";
      series4 = "/";
   }
}

//column keys
if( this.isReverse ){
   this.categoryMin = "> "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" >";
}else{
   this.categoryMin = "< "+this.categoryNumMin;
   this.categoryMax = this.categoryNumMax+" <";
}

//count
count1Min = 0;
count1Max = 0;
count2Min = 0;
count2Max = 0;
count3Min = 0;
count3Max = 0;
count4Min = 0;
count4Max = 0;

dataset.addValue( count1Min, series1, categoryMin );
dataset.addValue( count2Min, series2, categoryMin );
dataset.addValue( count3Min, series3, categoryMin );
dataset.addValue( count4Min, series4, categoryMin );

this.category = new String[ this.div ];
this.count1 = new int[ this.div ];
this.count2 = new int[ this.div ];
this.count3 = new int[ this.div ];
this.count4 = new int[ this.div ];

//int k = 0;
for( int i= 0; i<this.div; i++ ){
   category[i] = ""+( this.categoryNumMin + this.range * ( i) )
   +" - "+( this.categoryNumMin + this.range * ( i + 1 ) );
   count1[i] = 0;
   count2[i] = 0;
   count3[i] = 0;
   count4[i] = 0;
   //
   dataset.addValue( count1[i], series1, category[i] );
   dataset.addValue( count2[i], series2, category[i] );
   dataset.addValue( count3[i], series3, category[i] );
   dataset.addValue( count4[i], series4, category[i] );
   //k++;
}
```

```
    dataset.addValue( count1Max, series1, categoryMax );
    dataset.addValue( count2Max, series2, categoryMax );
    dataset.addValue( count1Max, series3, categoryMax );
    dataset.addValue( count2Max, series4, categoryMax );

    return dataset;
}


public void addData( int swc, int mod, BestBrine brineP ){

    //System.out.println( moduleName +"addData()");
    //System.out.println( moduleName + "■■isEventDispatchThread() addData ="
    //+ SwingUtilities.isEventDispatchThread() );

    final boolean isOnOff;

    String str = null;
    String stc = null;
    double count = 0.1;

    this.brine = brineP;
    //        温度
    //        質量流量

    if( this.dataType.equals( "1_温度[℃]")){
       this.dataValue = this.brine.getTemp();
    }else if(this.dataType.equals( "2_質量流量[g/s]")){
       this.dataValue = this.brine.getFlowRate();
    }


    //
    if( isswcIn ){
       isOnOff = Airswc.isON( swc );
    }else{
       isOnOff = true;
    }

    if( ismodIn ){
    }else{
       mod = Airmod.onCOOL( mod );
    }

    if( isOnOff ){
       if( Airmod.isCOOL( mod )){
          //str = "COOL/on";
          str = this.series1;
          if( isReverse ){
             if( this.dataValue > categoryNumMin ){
                count = ++count1Min;
                stc = categoryMin;
             }else if( this.dataValue <= categoryNumMax ){
                count = ++count1Max;
                stc = categoryMax;
             }else{
                for( int i = 0; i < div; i++ ){
                   if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                      count = ++count1[ i ];
                      stc = category[ i ];
                      break;
                   }
                }
             }
          }else{
             if( this.dataValue < categoryNumMin ){
                count = ++count1Min;
                stc = categoryMin;
             }else if( this.dataValue >= categoryNumMax ){
                count = ++count1Max;
                stc = categoryMax;
             }else{
```

```
            for( int i = 0; i < div; i++ ){
               if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                  count = ++count1[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }

   }else if( Airmod.isHEAT( mod )){
      //str = "HEAT/on";
      str = this.series3;
      if( isReverse ){
         if( this.dataValue > categoryNumMin ){
            count = --count3Min;
            stc = categoryMin;
         }else if( this.dataValue <= categoryNumMax ){
            count = --count3Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                  count = --count3[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }else{
         if( this.dataValue < categoryNumMin ){
            count = --count3Min;
            stc = categoryMin;
         }else if( this.dataValue >= categoryNumMax ){
            count = --count3Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                  count = --count3[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }

   }
}else{
   if( Airmod.isCOOL( mod )){
      //str = "COOL/off";
      str = this.series2;
      if( isReverse ){
         if( this.dataValue > categoryNumMin ){
            count = ++count2Min;
            stc = categoryMin;
         }else if( this.dataValue <= categoryNumMax ){
            count = ++count2Max;
            stc = categoryMax;
         }else{
            for( int i = 0; i < div; i++ ){
               if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                  count = ++count2[ i ];
                  stc = category[ i ];
                  break;
               }
            }
         }
      }else{
         if( this.dataValue < categoryNumMin ){
            count = ++count2Min;
            stc = categoryMin;
```

```
            }else if( this.dataValue >= categoryNumMax ){
               count = ++count2Max;
               stc = categoryMax;
            }else{
               for( int i = 0; i < div; i++ ){
                  if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                     count = ++count2[ i ];
                     stc = category[ i ];
                     break;
                  }
               }
            }
         }
      }else if( Airmod.isHEAT( mod )){
         //str = "HEAT/off";
         str = this.series4;
         if( isReverse ){
            if( this.dataValue > categoryNumMin ){
               count = --count4Min;
               stc = categoryMin;
            }else if( this.dataValue <= categoryNumMax ){
               count = --count4Max;
               stc = categoryMax;
            }else{
               for( int i = 0; i < div; i++ ){
                  if( this.dataValue > categoryNumMin + range * ( i+1 ) ){
                     count = --count4[ i ];
                     stc = category[ i ];
                     break;
                  }
               }
            }
         }else{
            if( this.dataValue < categoryNumMin ){
               count = --count4Min;
               stc = categoryMin;
            }else if( this.dataValue >= categoryNumMax ){
               count = --count4Max;
               stc = categoryMax;
            }else{
               for( int i = 0; i < div; i++ ){
                  if( this.dataValue < categoryNumMin + range * ( i+1 ) ){
                     count = --count4[ i ];
                     stc = category[ i ];
                     break;
                  }
               }
            }
         }
      }
   }

   final double fcount = count;
   final String rowKey = str;
   final String columnKey = stc;

   final Runnable worker = new Runnable() {
      public void run(){
         //System.out.println( moduleName +"worker run()");
         //System.out.println( moduleName + "■■isEventDispatchThread() worker ="
         //+ SwingUtilities.isEventDispatchThread() );
         if( rowKey != null && columnKey != null ){
            dataset.setValue( fcount, rowKey, columnKey );
         }
      }
   };

   Thread runThread = new Thread(){
      public void run() {
         //System.out.println( moduleName +"worker runThread()");
         //System.out.println( moduleName +"■■isEventDispatchThread() runThread ="
         //+ SwingUtilities.isEventDispatchThread() );
```

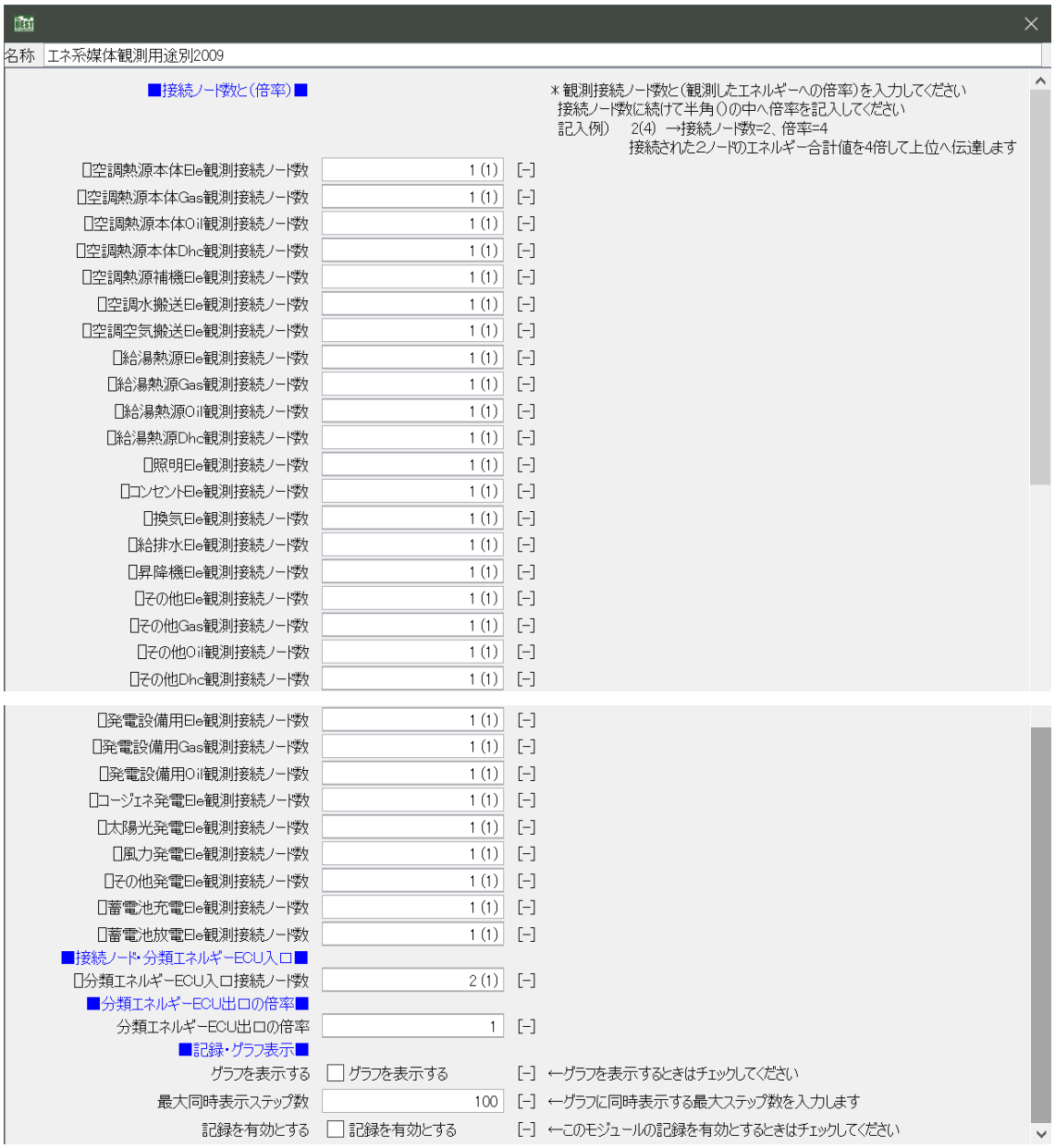
```
            try {
               SwingUtilities.invokeAndWait( worker );
            } catch (InterruptedException e) {
               e.printStackTrace();
            } catch (InvocationTargetException e) {
               e.printStackTrace();
            }
         }
      };

      runThread.start();
   }
}
```

「エネ系媒体観測用途別 2009」（場所：設備 2015／グラフ 計測 2015／）

モジュール名	エネ系媒体観測用途別 2009
クラス	MediumObsSumForECUModule20090808

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、エネルギー消費先別の BestElectricity、BestGas、BestOil、BestDhc、

BestECU の媒体接続ノードを持ち、これらに接続されたエネルギー消費を分別集計処理して管理・記録する機能を持つモジュールである。

分別集計処理したデータは、BestECU 媒体でまとめて管理し、L0_ecuOut 接続ノードから外部へ渡すことができる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

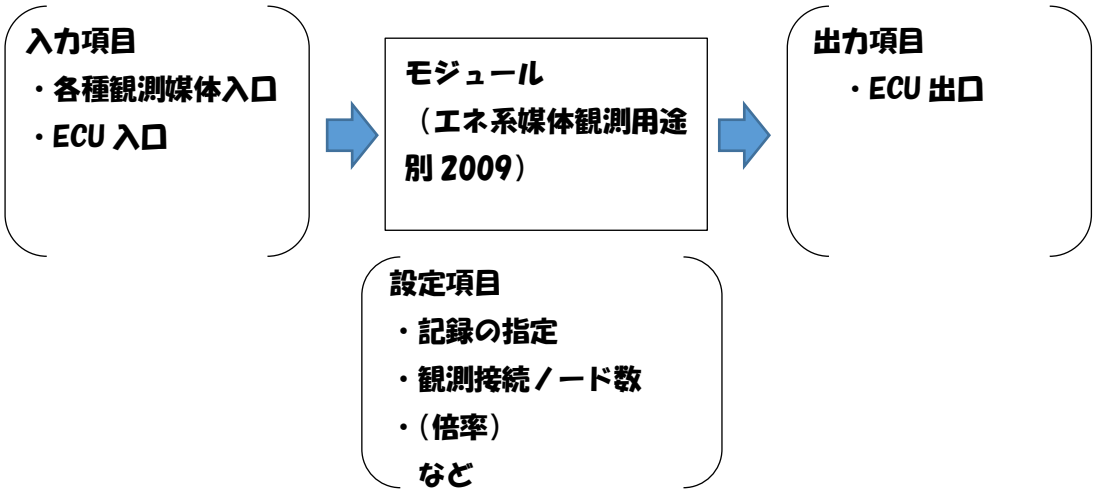


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	―	―		
1	[]空調熱源本体 Ele 観測接続ノード数	Int(int)	numberOfHSmain_EleObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
3	[]空調熱源本体 Gas 観測接続ノード数	Int(int)	numberOfHSmain_GasObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
4	[]空調熱源本体 Oil 観測接続ノード数	Int(int)	numberOfHSmain_OilObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
5	[]空調熱源本体 Dhc 観測接続ノード数	Int(int)	numberOfHSmain_DhcObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
6	[]空調熱源補機 Ele 観測接続ノード数	Int(int)	numberOfHSsub_EleObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
7	[]空調水搬送 Ele 観測接続ノード数	Int(int)	numberOfACpump_EleObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
8	[]空調空気搬送 Ele 観測接続ノード数	Int(int)	numberOfACfan_EleObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
9	[]給湯熱源 Ele 観測接続ノード数	Int(int)	numberOfHWHS_EleObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
10	[]給湯熱源 Gas 観測接続ノード数	Int(int)	numberOfHWHS_GasObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
11	[]給湯熱源 Oil 観測接続ノード数	Int(int)	numberOfHWHS_OilObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
12	[]給湯熱源 Dhc 観測接続ノード数	Int(int)	numberOfHWHS_DhcObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
13	[]照明 Ele 観測接続ノード数	Int(int)	numberOfLighting_EleObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
14	[]コンセント Ele 観測接続ノード数	Int(int)	numberOfConcent_EleObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
15	[]換気 Ele 観測接続ノード数	Int(int)	numberOfVentilation_EleObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
16	[]給排水 Ele 観測接続ノード数	Int(int)	numberOfWaterSupplyDrain_EleObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
17	[]昇降機 Ele 観測接続ノード数	Int(int)	numberOfEV_EleObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
18	[]その他 Ele 観測接続ノード数	Int(int)	numberOfOther_EleObs	1 (1)	[-]	―	―		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください

19	[]その他 Gas 観測接続ノード数	Int(int)	numberOfOther_GasObs	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
20	[]その他 Oil 観測接続ノード数	Int(int)	numberOfOther_OilObs	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
21	[]その他 Dhc 観測接続ノード数	Int(int)	numberOfOther_DhcObs	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
22	[]発電設備用 Ele 観測接続ノード数	Int(int)	numberOfeleObsCGS	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
23	[]発電設備用 Gas 観測接続ノード数	Int(int)	numberOfgasObsCGS	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
24	[]発電設備用 Oil 観測接続ノード数	Int(int)	numberOfoilObsCGS	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
25	[]コージェネ発電 Ele 観測接続ノード数	Int(int)	numberOfeleObsGenCGS	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
26	[]太陽光発電 Ele 観測接続ノード数	Int(int)	numberOfeleObsGenSOL	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
27	[]風力発電 Ele 観測接続ノード数	Int(int)	numberOfeleObsGenWIN	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
28	[]その他発電 Ele 観測接続ノード数	Int(int)	numberOfeleObsGenXXX	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
29	[]蓄電池充電 Ele 観測接続ノード数	Int(int)	numberOfeleObsBAT	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
30	[]蓄電池放電 Ele 観測接続ノード数	Int(int)	numberOfeleObsGenBAT	1 (1)	[-]	—	—		＊観測接続ノード数と（観測したエネルギーへの倍率）を入力してください
31	[]分類エネルギーECU 入口接続ノード数	Int(int)	numberOfecuIn	1 (1)	[-]	—	—		＊ecu 接続ノード数と（エネルギーへの倍率）を入力してください
32	分類エネルギーECU 出口の倍率	Int	x_ecuOut	1	[-]	—	—		＊出口 ecu 分類エネルギーへの倍率を入力してください
33	グラフを表示する	boolean	isGVisible	FALSE	[-]	—	—		←グラフを表示するときはチェックしてください
34	最大同時表示ステップ数	Int	maxItemCount	100	[-]	—	—		←グラフに同時表示する最大ステップ数を入力します
35	記録を有効とする	Boolean	isRecorde	FALSE	[-]	—	—		←このモジュールの記録を有効とするときはチェックしてください

（５）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	発停	発停	入口	上位からの発停信号（未使用）
3	空調熱源本体 Ele 観察	LO_eleObsHSmain[]	eleObsHSmain[]	-	状態	電力	観測	空調熱源本体の電力の観測接続用[0]から[観測接続数-1]
4	空調熱源本体 Gas 観察	LO_gasObsHSmain[]	gasObsHSmain[]	-	状態	ガス	観測	空調熱源本体のガスの観測接続用[0]から[観測接続数-1]
5	空調熱源本体 Oil 観察	LO_oilObsHSmain[]	oilObsHSmain[]	-	状態	油	観測	空調熱源本体の油の観測接続用[0]から[観測接続数-1]
6	空調熱源本体 Dhc 観察	LO_dhcObsHSmain[]	dhcObsHSmain[]	-	状態	熱供給	観測	空調熱源本体の熱供給の観測接続用[0]から[観測接続数-1]
7	空調熱源補機 Ele 観察	LO_eleObsHSsub[]	eleObsHSsub[]	-	状態	電力	観測	空調熱源補機の電力の観測接続用[0]から[観測接続数-1]
8	空調水搬送 Ele 観察	LO_eleObsACpump[]	eleObsACpump[]	-	状態	電力	観測	空調水搬送の電力の観測接続用[0]から[観測接続数-1]
9	空調空気搬送 Ele 観察	LO_eleObsACfan[]	eleObsACfan[]	-	状態	電力	観測	空調空気搬送の電力の観測接続用[0]から[観測接続数-1]
10	給湯熱源 Ele 観察	LO_eleObsHWHS[]	eleObsHWHS[]	-	状態	電力	観測	給湯熱源の電力の観測接続用[0]から[観測接続数-1]
11	給湯熱源 Gas 観察	LO_gasObsHWHS[]	gasObsHWHS[]	-	状態	ガス	観測	給湯熱源のガスの観測接続用[0]から[観測接続数-1]
12	給湯熱源 Oil 観察	LO_oilObsHWHS[]	oilObsHWHS[]	-	状態	油	観測	給湯熱源の油の観測接続用[0]から[観測接続数-1]
13	給湯熱源 Dhc 観察	LO_dhcObsHWHS[]	dhcObsHWHS[]	-	状態	熱供給	観測	給湯熱源の熱供給の観測接続用[0]から[観測接続数-1]
14	照明 Ele 観察	LO_eleObsLighting[]	eleObsLighting[]	-	状態	電力	観測	照明の電力の観測接続用[0]から[観測接続数-1]
15	コンセント Ele 観察	LO_eleObsConcent[]	eleObsConcent[]	-	状態	電力	観測	コンセントの電力の観測接続用[0]から[観測接続数-1]
16	換気 Ele 観察	LO_eleObsVentilation[]	eleObsVentilation[]	-	状態	電力	観測	換気の電力の観測接続用[0]から[観測接続数-1]
17	給排水 Ele 観察	LO_eleObsWaterSupplyDrain[]	eleObsWaterSupplyDrain[]	-	状態	電力	観測	給排水の電力の観測接続用[0]から[観測接続数-1]
18	昇降機 Ele 観察	LO_eleObsEV[]	eleObsEV[]	-	状態	電力	観測	昇降機の電力の観測接続用[0]から[観測接続数-1]
19	その他 Ele 観察	LO_eleObsOther[]	eleObsOther[]	-	状態	電力	観測	その他電力の観測接続用[0]から[観測接続数-1]
20	その他 Gas 観察	LO_gasObsOther[]	gasObsOther[]	-	状態	ガス	観測	その他ガスの観測接続用[0]から[観測接続数-1]
21	その他 Oil 観察	LO_oilObsOther[]	oilObsOther[]	-	状態	油	観測	その他油の観測接続用[0]から[観測接続数-1]
22	その他 Dhc 観察	LO_dhcObsOther[]	dhcObsOther[]	-	状態	熱供給	観測	その他熱供給の観測接続用[0]から[観測接続数-1]
23	発電設備用 ele 観察	LO_eleObsCGS[]	eleObsCGS[]	-	状態	電力	観測	発電設備の電力の観測接続用[0]から[観測接続数-1]
24	発電設備用 Gas 観察	LO_gasObsCGS[]	gasObsCGS[]	-	状態	ガス	観測	発電設備のガスの観測接続用[0]から[観測接続数-1]
25	発電設備用 Oil 観察	LO_oilObsCGS[]	oilObsCGS[]	-	状態	油	観測	発電設備の油の観測接続用[0]から[観測接続数-1]

26	コージェネ発電Ele観察	LO_eleObsGenCGS[]	eleObsGenCGS[]	-	状態	電力	観測	コージェネ発電電力の観測接続用[0]から[観測接続数-1]
27	太陽光発電Ele観察	LO_eleObsGenSOL[]	eleObsGenSOL[]	-	状態	電力	観測	太陽光発電電力の観測接続用[0]から[観測接続数-1]
28	風力発電Ele観察	LO_eleObsGenWIN[]	eleObsGenWIN[]	-	状態	電力	観測	風量発電電力の観測接続用[0]から[観測接続数-1]
29	その他発電Ele観察	LO_eleObsGenXXX[]	eleObsGenXXX[]	-	状態	電力	観測	その他発電電力の観測接続用[0]から[観測接続数-1]
30	蓄電池充電Ele観察	LO_eleObsBAT[]	eleObsBAT[]	-	状態	電力	観測	蓄電池充電電力の観測接続用[0]から[観測接続数-1]
31	蓄電池放電Ele観察	LO_eleObsGenBAT[]	eleObsGenBAT[]	-	状態	電力	観測	蓄電池放電電力の観測接続用[0]から[観測接続数-1]
32	ECU入口	LO_ecuIn[]	ecuIn[]	-	状態	ECU	観測	ECU入口の観測接続用[0]から[観測接続数-1]
33	ECU出口	LO_ecuOut	ecuOut	-	状態	ECU	観測	ECU出口

（6）記録項目

No.	項目	出力内容	単位	分類

（7）計算フロー・計算内容

エネルギー消費を用途先別に分別集計し、BestECU 媒体にてまとめて管理する。

分別集計に際しては、用途先別の倍率を適用して集計する。

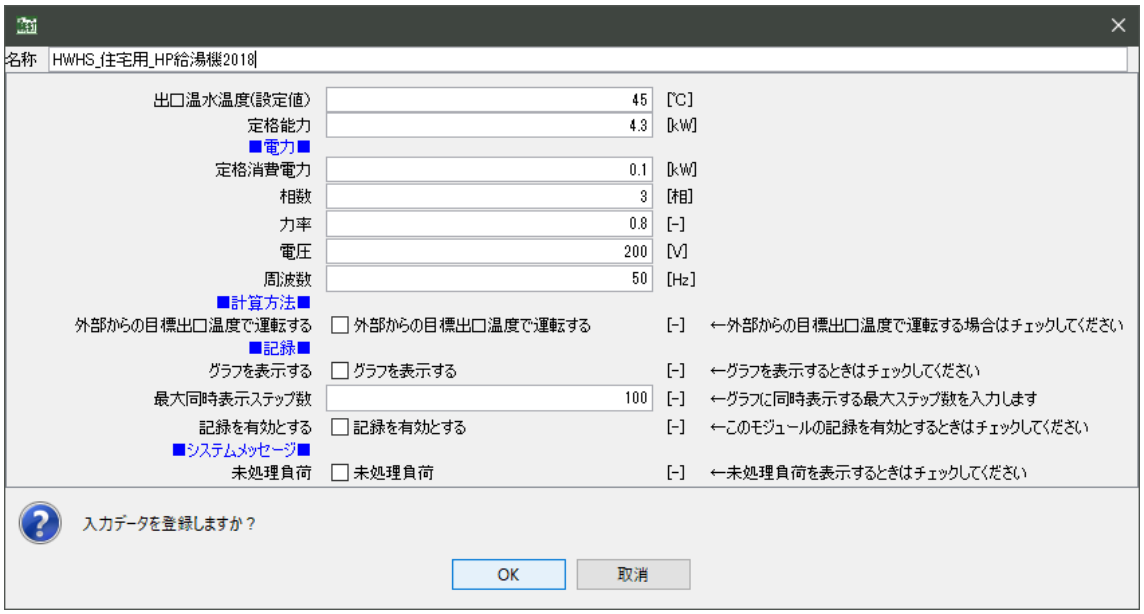
（8）データ範囲と範囲外の取扱い

特になし。

「電気ヒートポンプ温水暖房機（住宅用）」（場所：設備 2015／衛生設備 2015）

モジュール名	HWHS_住宅用_HP 給湯機 2018
クラス	HWEleHeatPumpHeater

（1）入力画面



（２）モジュールの概要

本モジュールは、Ver.08（エネルギー消費性能計算プログラム（住宅版）Ver.02.01～）2017.7の付録Ｄ　電気ヒートポンプ温水暖房機(フロン系)の計算ロジックを参照している。出口温水温度と外気温度により、最大加熱能力や熱源効率が変動する。

入力、出力、スペックのイメージを図１に示す。

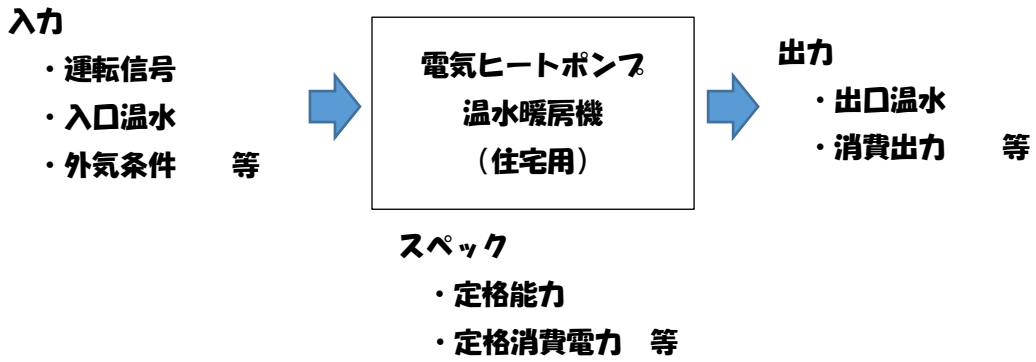


図１　モジュールの設定項目・入力項目・出力項目

（３）モジュールの入出力

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
1	出口温水温度(設定値)	String	sTWout	45	[℃]	ー	0	◎	8.外部からの目標出口温度で運転する場合は無効です。
2	定格能力	String	Qspec	4.3	[kW]	ー	0	◎	
	■電力■								
3	定格消費電力	String	Pspec	0.1	[kW]	ー	0	◎	
4	相数	String		3	[相]	ー	0		
5	力率	String		0.8	[-]	ー	0		
6	電圧	String		200	[V]	ー	0		
7	周波数	Double		50	[Hz]	ー	0		
	■計算方法■								
8	外部からの目標出口温度で運転する	Boolean	isValInSP	FALSE	[-]	ー	ー		外部からの目標出口温度で運転する場合はチェックしてください。
	■記録■								
9	グラフを表示する	Boolean		FALSE	[-]	ー	ー		グラフを表示するときはチェックしてください。
10	最大同時表示ステップ数	Int		100	[-]	ー	0		グラフに同時表示する最大ステップ数を入力します。
11	記録を有効とする	Boolean		FALSE	[-]	ー	ー		このモジュールの記録を有効とするときはチェックしてください。
	■システムメッセージ■								
12	未処理負荷	Boolean	Qerr	FALSE	[-]	ー	ー		未処理負荷を表示するときはチェックしてください。

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	-		記録	メモリ	出口	
2	ON/OFF 信号	L1_swcIn	SwcIn		制御	ON/OFF	入口	
3	入口温水	L0_watIn	Twin GWin	[℃] [g/s]	状態	水	入口	
4	出口温水	L0_watOutHW	TWout		状態	水	出口	
5	外気	L0_airIn	DBO RHO	[℃] [%]	状態	空気	入口	
6	電力	L0_eleIn	P		状態	電気	入口	
7	外部から受け取る出口温度(設定値)	L0_valInSPTHW	ValInSPTHW	[℃]	状態	Double 値	入口	

（6）記録項目

No.	項目	出力内容	単位	分類	記号
1	ON/OFF 状況	制御	–	メッセージ	
2	消費電力	電力	W	エネルギー	P
3	加熱能力	能力	W	負荷	Qout
4	未処理負荷	能力	W	負荷	Qerr
5	出口温水流量	質量流量	g/s	出口	(= GWin)
6	出口温水温度	温度	℃	出口	TWout
7	COP	COP	-	My	Qout /P
8	負荷率	-	-	My	Qout/ Qspec
9	入口水流量	質量流量	g/s	入口	GWin
10	入口水温度	温度	℃	入口	TWin
11	入口空気温度	温度	℃	入口	DBO

（7）計算フロー・計算内容

モジュール内での計算内容を示す。

1）最大消費電力 Pmax[W]の算出

$$P_{max} = \frac{P_{spec}}{0.6} \qquad [W]$$

Pspec：定格消費電力[W]

2）必要処理負荷 Qload[W]（>0）の算出

$$Q_{load} = \left(T_{set}\text{'} - TW_{in}\right) \times GW_{in} \times 4.18605 \qquad [W]$$

TWin：入口水温[℃]

GWin：入口水量[g/s]

sTWout'：出口温水温度(設定値)[℃]

※出口温度(設定値)を外部から受け取る（isValInSP＝true）場合は sTWout'＝ ValInSPTHW、受け取らない場合は sTWout'=sTWout

3）最大加熱能力 Qmax[W]の算出

$$Q_{max} = (11.62 + 0.2781 \times DBO - 0.00113 \times (TW_{out})^2 - 0.1271 \times DBO \times TW_{out}) \times \frac{Q_{de}}{6} \times \frac{C_{def}}{0.85} \qquad [W]$$

TWout：出口温水温度[℃]

※初期値は sTWout'とする。

ValInSPTHW：外部から受け取る出口温度(設定値)

sTWout：出口温度(設定値)

DBO：外気温度[℃]

TWout：出口温水温度[℃]

Qspec：定格能力[W]

Cdef：デフロスト補正係数

※DBO ＜ 5[℃]かつ 80[%] ＜ RHO の場合は Cdef=0.85、それ以外の場合は Cdef=1.0

RHO：外気相対湿度[%]

4）加熱能力（処理負荷）Qout[W]と未処理負荷 Qerr[W]の算出

Qmax ＜ Qload の場合

$$Q_{out} = Q_{max} \qquad [W]、\ Q_{over} = Q_{load} - Q_{max} \qquad [W]$$

Qload≦ Qmax の場合

$$Q_{out} = Q_{load} \quad [W]、\ Q_{err} = 0 \quad [W]$$

5）出口温水温度 TWout[℃]の算出

$$TW_{out} = TW_{in} + \frac{Q_{out}}{GW_{in} \times 4.18605} \quad [℃]$$

※3）の最大加熱能力を再計算し、誤差 5%未満になるまで3)、4)、5）を繰り返す。

TWin：入口水温[℃]

6）熱源機効率比 er[-]の算出

$$er = \left(1.120656 - 0.03703 \times (TW_{out} - DBO)\right) \times (1 - \frac{Q_{out}}{Q_{max}})^2 + \left(-0.36786 + 0.012152 \times (TW_{out} - DBO)\right) \times (1 - \frac{Q_{out}}{Q_{max}}) + 1 \quad [-]$$

7）熱源機効率 e[-]の算出

$$e = \frac{Q_{max}}{P_{max}} \times er \quad [W]$$

Qmax：最大加熱能力[W]

Pmax：最大消費電力[W]

er：熱源機効率比[-]

8）消費電力 P[W]の算出

$$P = \frac{Q_{out}}{e} \quad [W]$$

Qout：加熱能力[W]（処理負荷）

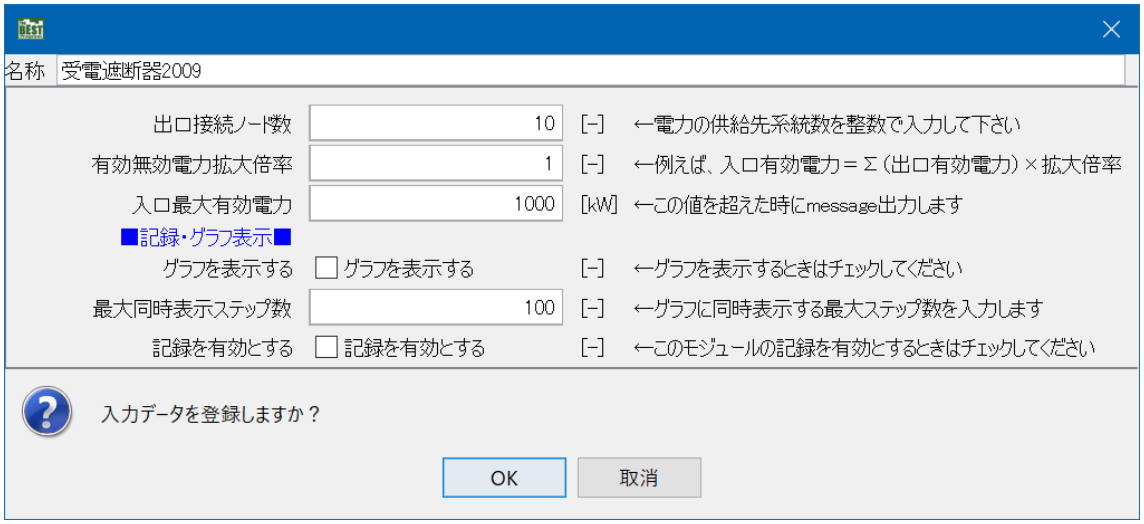
e：熱源機効率[-]

「受電遮断器 2009」（場所：設備 2015／電気設備 2015／）

モジュール名	受電遮断器 2009
クラス	DistributionBoad_nOut1InModule20090101

（1）入力画面

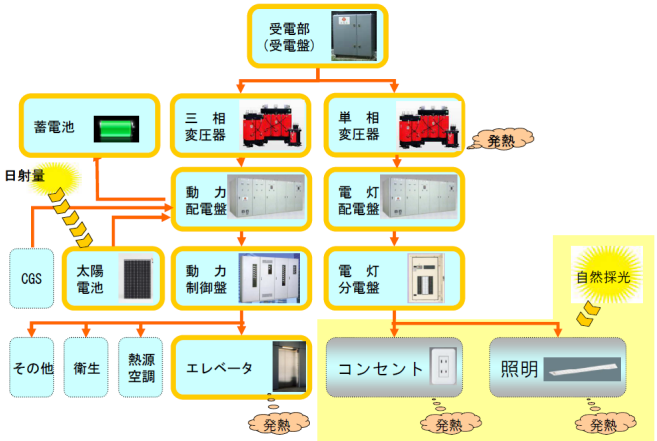
・スペック



（2）モジュールの概要

電力を分配するモジュールで、出口側に接続された電力を集計し、入口電力を設定する。

下図、電気設備の主要モジュール構成を参照。



（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

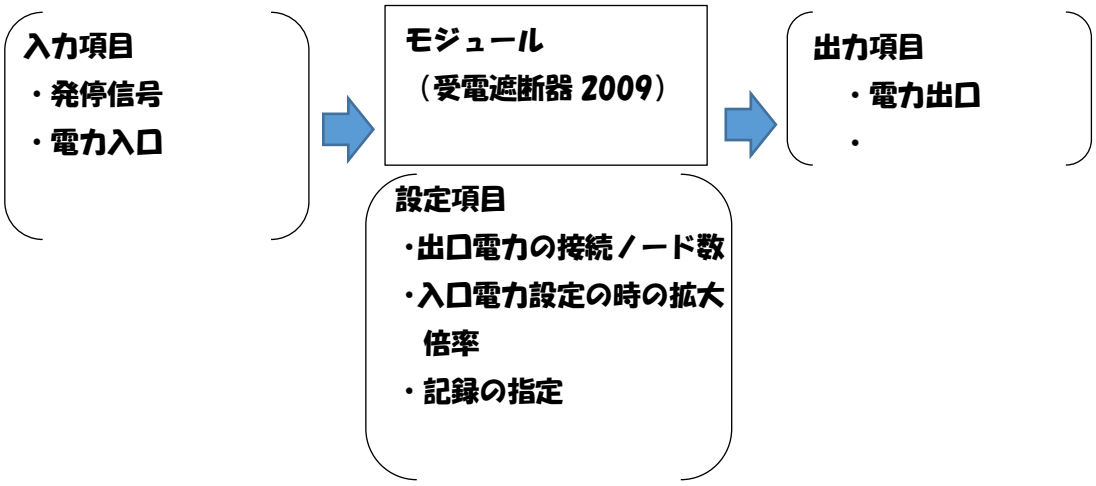


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図1における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String		―	[-]	―	―		
1	出口接続ノード数	int	numberOfOutlet	10	[-]	―	1		←電力の供給先系統数を整数で入力して下さい
2	有効無効電力拡大倍率	double	enlarge	1	[-]	―	0		←例えば、入口有効電力＝Σ（出口有効電力）×拡大倍率
3	入口最大有効電力	double	eleIn_maxActivePower	1000	[kW]	―	0		←この値を超えた時にmessage出力します 計算には影響しない
4	■記録・グラフ表示■					―	―		
5	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
6	最大同時表示ステップ数	int	maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
7	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	空調記録モジュールへ接続する
2	発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	上位からの発停信号
3	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	
4	電力出口[]	L0_eleOut[]	eleOut[]	-	状態	電力	出口	スペックの出口接続ノード数で入力された数のノードが用意される
5	電力デマンド	L0_valOutDemandele	valOutDemandele	-	値	値	出口	電力デマンドを値で出す

（6）記録項目

No.	項目	出力内容	単位	分類
1	DBoad_nOut1InMessage#-#-	メッセージ	—	メッセージ
2	DBoad_nOut1In 入口有効電力#W#-	入口有効電力	W	入口
3	DBoad_nOut1In 入口無効電力#Var#-	入口無効電力	Var	入口
4	DBoad_nOut1InLO_eleOut[i]有効電力#W#-	各出口有効電力	W	出口
5	DBoad_nOut1InLO_eleOut[i]無効電力#Var#-	各出口無効電力	Var	出口

（７）計算フロー・計算内容

・swcln による動作

swcln が停止信号の時

電力は流れないとして次の処理を行う。

eleIn および各 eleOut[]の有効電力、無効電力、電流を＝０とする。

swcln が運転信号の時

・出口電力の有効電力と無効電力を集計する。

出口有効電力合計＝Σ eleOut[i]有効電力

出口無効電力合計＝Σ eleOut[i]無効電力

・拡大倍率の適用

電力デマンド有効電力＝出口有効電力合計×有効無効電力拡大倍率

電力デマンド無効電力＝出口無効電力合計×有効無効電力拡大倍率

・入口電力の算定

入口有効電力＝出口有効電力合計

入口無効電力＝出口無効電力合計

（８）データ範囲と範囲外の取扱い

特になし。

参考　ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.Map;

import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * @author HIROSHI NINOMIYA /20080909
 * 分電盤です
 *   入口１系統
 *   出口ｎ系統
 *   出口側の電力を積算し入口側へ伝えます
 * 記録の有効無効判断を追加
 * 20090101 グラフ表示ステップ数追加　デマンドノード追加
 */
public class DistributionBoad_nOut1InModule20090101 extends AbstractBestModule implements
      IBestMetaModule {

   //
   private final String moduleName = "(DistributionBoad_nOut1InModule20090101) ";

   //接続ノード
   private final String S_NODE_eleIn = "L0_eleIn";
   private final String S_NODE_valOutDemandele = "L0_valOutDemandele";

   //
   //出口　接続名称は　L0_eleOut[0],L0_eleOut[1]・・・・とする
   //
   private String[] S_NODE_eleOut;

   private final String C_NODE_swcIn = "L1_swcIn";
   //private final String C_NODE_modIn = "L1_modIn";

   private final String R_NODE = "L2_recOut";

   //仕様
   private final String SPEC_name = "名称";
   private final String SPEC_NumberOfOutlet = "[]出口接続ノード数";
   private final String SPEC_eleIn_maxActivePower = "入口最大有効電力[W]";

   private final String SPEC_enlarge = "有効無効電力拡大倍率";
   //
   private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
   private final String SPEC_maxItemCount = "最大同時表示ステップ数";//最大同時表示ステップ数
   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする


   //記録
   private final String RECORD_message = "DBoad_nOut1InMessage#-#-";
   private final String RECORD_A_eleIn = "DBoad_nOut1In入口有効電力#W#-";
   private final String RECORD_R_eleIn = "DBoad_nOut1In入口無効電力#Var#-";

   //接続熱媒など
   private BestElectricity[] eleOut = null;
   private BestElectricity   eleIn = null;
   private BestValue valOutDemandele = null;
// private BestElectricity   eleMy = null;
```

```
//制御信号など
private int swcIn;                //
//private int modIn;              //運転モード

//仕様など
private String name;          //名称
private int numberOfOutlet;         //出口　接続数

private double eleIn_maxActivePower;      //入口　最大有効電量[W]
private double enlarge;//= "有効無効電力拡大倍率";
//
private boolean isGVisible = false;//このグラフを表示する=true
private int  maxItemCount = 100;//最大同時表示ステップ数
private boolean isRecord   = false;//記録を有効とする=true

private StringBuffer message = new StringBuffer();


//グラフ表示など
private GraphJFrameD_Boad20080909 gD_Boad = null;

private boolean isCalcEleOut = false;


@Override
public void setProfile(BestSpecs spec) {
   if(spec == null) {
      return;
   }

   Map<String, String> map = spec.getSpec();
   if(map == null) {
      return;
   }

   //名称を取得
   this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

   //numberOfOutletを取得
   if (null != map.get(this.SPEC_NumberOfOutlet)) {
      this.numberOfOutlet
      = Integer.parseInt( (String)map.get( this.SPEC_NumberOfOutlet ));
      //出口ｎ系統のBestWaterの初期化
      if( this.numberOfOutlet > 0 ){
         this.isCalcEleOut = true;
         this.eleOut = new BestElectricity[this.numberOfOutlet];
         this.S_NODE_eleOut = new String[this.numberOfOutlet];
         for( int i=0; i<this.eleOut.length; i++ ){
            //this.eleOut[i] = new BestElectricity();
            //this.eleOut[i] = new BestElectricity( 0, 0, 200, 0, 3, 50);
            this.S_NODE_eleOut[i] = new String( "LO_eleOut[" + i + "]" );
         }
      }

   }

   //SPEC_eleIn_maxActivePowerを取得
   this.eleIn_maxActivePower = spec.getSpecValue( this.SPEC_eleIn_maxActivePower, 0. );

   //SPEC_enlarge = "有効無効電力拡大倍率";
   this.enlarge = spec.getSpecValue( this.SPEC_enlarge, 1. );

   //isGVisibleを取得
   this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

   // 最大同時表示ステップ数
   this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

   //isRecordを取得
   this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

}
```

```
   public void initialize(IBestStateMessage stateNodes,
         IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
         IBestDomainSchedule schedule) {
      //状態ノードを取得
      super.sm = stateNodes;

      //制御ノードを取得
      super.cm = commandNodes;

      //記録ノードを取得
      super.rm = recordNodes;

      //接続ノード　出口
      if( this.isCalcEleOut ){
         for( int i=0; i<this.eleOut.length; i++ ){
            this.eleOut[i] = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleOut[i] );
            //System.out.println( "!  "+1 );
         }
      }

      //接続ノード　入口
      //eleIn = new BestWater();
      this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

      //valOutDemandele;
      this.valOutDemandele = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valOutDemandele );

//    this.eleMy = new BestElectricity();

      //グラフ表示の準備
      if( this.isGVisible ){
         this.gD_Boad = new GraphJFrameD_Boad20080909( this.name, this.maxItemCount, this.eleOut.length,
this.eleIn_maxActivePower * 1.5, this.eleIn_maxActivePower / this.enlarge / this.eleOut.length );
      }
   }

   public void outputs() {
      //if( super.sm == null || super.cm == null ) {
      // return;
      //}
      //接続　入口状態値の取得　（電力は出口）
      if( this.isCalcEleOut ){
         for( int i=0; i<this.eleOut.length; i++ ){
            this.eleOut[i]
            = (BestElectricity) super.sm.getState( super.getConnectionNode( this.S_NODE_eleOut[i] ));
         }
      }

      this.swcIn = super.cm.getCommand(super.getConnectionNode( this.C_NODE_swcIn ));

      double activePower;
      double reactivePower;

      activePower = 0.;
      reactivePower = 0.;

      if( this.isCalcEleOut ){
         this.eleIn.copyAllState( this.eleOut[0] );      //eleOut[]の最初の接続eleを採用
      }

      //if(this.swcIn == 0 ){    //利用不可　遮断時
      if( Airswc.isOFF( this.swcIn ) ){      //利用不可　遮断時
         if( this.isRecord ){
            this.message.append( "(C)遮断中");
         }
         this.eleIn.setActivePower( 0. );
         this.eleIn.setCurrent( 0. );
         this.eleIn.setReactivePower( 0. );

         if( this.isCalcEleOut ){
            for( int i = 0; i < this.eleOut.length; i++ ){
```

```
                this.eleOut[i].setActivePower( 0. );
                this.eleOut[i].setReactivePower( 0. );
                this.eleOut[i].setCurrent( 0. );
            }
        }

    }else{   //利用可能　通電時
        if( this.isRecord ){
            this.message.append( "(C)通電中");
        }
        // 集計
        //入口のactivePower,reactivePower
        //System.out.println( "eleOut length=" + this.eleOut.length );

        if( this.isCalcEleOut ){
            for( int i = 0; i < this.eleOut.length; i++ ){
                //System.out.println( "eleOut i=" + i);
                //if( this.eleOut[i] == null )continue;
                //System.out.println( "eleOut i=" + i +" "+this.name);
                activePower += this.eleOut[i].getActivePower();
                reactivePower += this.eleOut[i].getReactivePower();
            }
        }

        //拡大
        activePower *= this.enlarge;
        reactivePower *= this.enlarge;
        //20090101
        this.valOutDemandele.setValue( activePower );//負荷の合計に倍率を掛けた値をデマンドとする
        if( this.eleIn_maxActivePower < activePower ){
            if( this.isRecord ){
                this.message.append( "(W)最大電力超過");
            }
        }

        this.eleIn.setActivePower( activePower );
        this.eleIn.setReactivePower( reactivePower );
    }


    //watMyの状態値のセット
//    this.eleMy.copyAllState( this.eleIn );

    //ノードに設定
    super.sm.setState(super.getConnectionNode( this.S_NODE_eleIn ),
          this.eleIn );
    super.sm.setState(super.getConnectionNode( this.S_NODE_valOutDemandele ),
          this.valOutDemandele );

    //記録
    //グラフデータ追加
    if( this.isGVisible ){
        gD_Boad.addData( BestTimeManager.getDateWeatherTime(),
              this.swcIn,
              this.eleOut,
              this.eleIn_maxActivePower,
              this.eleIn );
    }

    //記録
    if( this.isRecord  && super.rm != null ){
        this.record();
    }

    message.setLength(0);
}

/**
 * 記録
 */
private void record(){
```

```
        if( CheckPrintModule.isPrintMessage ){
           super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                 this.RECORD_message, this.name, this.message );
        }

        //if( CheckPrintModule.isPrintEnergy ){
        //}

        //if( CheckPrintModule.isPrintLoad ){
        //}

        if( CheckPrintModule.isPrintStateOut ){
           //出口
           if( this.isCalcEleOut ){
              for( int i=0; i<eleOut.length; i++ ){
                 super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                       this.S_NODE_eleOut[i] + "有効電力#W#電力",this.name,
                       AirFormat.df_2( this.eleOut[i].getActivePower() ));
                 super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                       this.S_NODE_eleOut[i] + "無効電力#Var#無効電力",this.name,
                       AirFormat.df_2( this.eleOut[i].getReactivePower() ));
              }
           }
        }

        //if( CheckPrintModule.isPrintStateMy ){
        //}

        if( CheckPrintModule.isPrintStateIn ){
           //入口
           super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                 this.RECORD_A_eleIn, this.name,
                 AirFormat.df_2( this.eleIn.getActivePower() ));
           super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                 this.RECORD_R_eleIn, this.name,
                 AirFormat.df_2( this.eleIn.getReactivePower() ));
        }
     }

     public void update() {

     }

     public Object viewInternal(TestCommand cmd) {
        // とりあえずクラス変数のリストを返します。
        java.util.List<Object> result = new java.util.ArrayList<Object>();

        //接続ノード
        result.add(super.sm);
        result.add(super.cm);

        //外部定義項目
        result.add(eleIn_maxActivePower);

        if( eleOut != null ){
           for( int i = 0; i < eleOut.length; i++ ){
              result.add( eleOut[i] );
           }
        }
//      result.add(this.eleMy);

        return result;
     }
}
```

「変圧器 2009」（場所：設備 2015／電気設備 2015／）

モジュール名	変圧器 2009
クラス	TransformerModule20091111

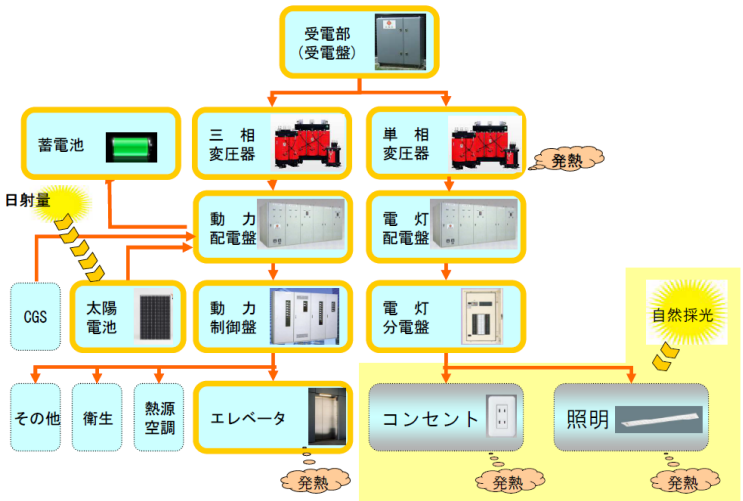
（1）入力画面

・スペック

（2）モジュールの概要

変圧器に関わるエネルギー損失量を求めるモジュールである。

下図、電気設備の主要モジュール構成を参照。



（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String		―	[-]	―	―		
1	相	int	phase	1、3	[φ]	―	―		
2	容量	double	capacity	300	[kVA]	―	―		75、100、150、200、300
3	内部発熱換算係数	double	heatGainFactor	100	[%]	100	0		一般的には100%であるが、換気や空調計算の際に換算係数を考慮できる場合は入力する。
4	無負荷損	double	noLoadLoss	487	[W]	―	0		
5	負荷損	double	loadLoss	2820	[W]	―	0		
6	■並列設定の小計■					―	―		
7	容量小計	double	capacitySubtotal	300	[kVA]	―	―		←変圧器を並列設置している場合の合計容量。これに対して分担率を決める。
8	■設置室■					―	―		
9	室グループ／室／ゾーン	String	grzName	#RoomGroup	[-]	―	―		←室グループ／室／ゾーンを選択してください。変圧器の発熱をこの室へ渡します。
10	■記録・グラフ表示■					―	―		
11	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
12	最大同時表示ステップ数	int	maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
13	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	空調記録モジュールへ接続する
2	発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	上位からの発停信号
3	電力入口	LO_eleIn	eleIn	-	状態	電力	入口	電力入口＝電力出口＋損失電力
4	電力出口	LO_eleOut	eleOut	-	状態	電力	出口	
5	損失電力	LO_eleObsLoss	eleObsLoss	-	状態	電力	観察	変圧器で生ずる損失
6	発熱	LO_heaOut	heaOut	-	状態	熱	出口	（未使用） 変圧器の発熱を変圧器を設置している室へ渡す場合、スペックの室グループ／室／ゾーンで設置室を設定する。

（6）記録項目

No.	項目	出力内容	単位	分類
1	TR_入口有効電力#W#-	入口有効電力	W	入口
2	TR_入口無効電力#Var#-	入口無効電力	Var	入口
3	TR_並列出口有効電力小計#W#-	並列出口有効電力	W	出口
4	TR_並列出口無効電力小計#Var#-	並列出口無効電力	Var	出口
5	TR_出口有効電力#W#-	出口有効電力	W	出口
6	TR_出口無効電力#Var#-	出口無効電力	Var	出口
7	TR_負荷率#-#-	負荷率	-	My
8	TR_変圧器損失#W#電力 その他	変圧器損失電力	W	My
9	TR_変圧器発熱#W#-	変圧器発熱	W	My
10	TR_変圧器ゾーン熱負荷#W#-	ゾーンへの熱負荷	W	負荷
11	TR_変圧器ゾーン熱負荷密度#W/m2#-	ゾーンへの熱負荷密度	W	負荷

（７）計算フロー・計算内容

・容量比率[-]

容量比率とは複数の変圧器を並列設置する場合に、個々の変圧器が容量小計（＝並列設置した変圧器の合計容量）に占める比率として、次の式で求めた値とする。

容量比率＝容量／容量小計

・担当有効電力[W]、担当無効電力[Var]

担当有効電力＝出口有効電力×容量比率

担当無効電力＝出口無効電力×容量比率

・負荷率[-]

負荷率＝（担当有効電力×担当有効電力＋担当無効電力×担当無効電力）$^{0.5}$ ／ 容量

・変圧器損失電力[W]

変圧器損失電力＝無負荷損＋負荷損×負荷率×負荷率

・発熱量[W]

発熱量＝変圧器損失×内部発熱換算係数

・swcln による動作

swcln が停止信号の時

変圧器は動作していないものとし、入口電力は有効・無効電力ともに＝０とする。

出口電力の値は無視する処理を行い上位へ伝達する。

負荷率＝０、変圧器損失電力＝０とする。

swcln が運転信号の時

負荷率、変圧器損失電力、発熱量を計算し、出口有効電力を次式で算定する。

出口有効電力＝担当有効電力＋変圧器損失電力

（８）データ範囲と範囲外の取扱い

特になし。

参考　ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.Map;
import java.util.List;

import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestHeat;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.building.load.SystemHeatGain;
import jp.or.ibec.best.domain.building.spaces.ISpace;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * 変圧器のクラス
 *
 * HIROSNI NINOMIYA /20071121 /20071122 /20080428 /20080909
 *
 * 記録の有効無効判断を追加
 * 20090101 グラフ表示ステップ数追加
 * 20091111 変圧器でのロスをBestElectricityのNodeとして「L0_eleObsLoss」で受け渡し
 * 20091111 発熱Nodeとして「L0_heaOut」で受け渡し
 *
 * //20130801  Heat→BestHeat
 */
public class TransformerModule20091111 extends AbstractBestModule
      implements IBestMetaModule, jp.or.ibec.best.dk.test.IProbeForUT {

   private final String moduleName = "(TransformerModule20091111) ";

   //接続ノード
   //出入り口
   private final String S_NODE_eleIn  = "L0_eleIn";   //状態ノード
   private final String S_NODE_eleOut = "L0_eleOut";  //状態ノード
   private final String S_NODE_eleLoss= "L0_eleObsLoss";//20091111

   private final String S_NODE_heaOut = "L0_heaOut";//20091111機器発熱の出力ノード

   //制御
   private final String C_NODE_swcIn  = "L1_swcIn";

   //記録ノード
   private final String R_NODE = "L2_recOut";   //記録ノード

   private final String RECORD_A_eleIn        = "TR_入口有効電力#W#-";
   private final String RECORD_R_eleIn        = "TR_入口無効電力#Var#-";
   private final String RECORD_A_eleOutALL    = "TR_並列出口有効電力小計#W#-";
   private final String RECORD_R_eleOutALL    = "TR_並列出口無効電力小計#Var#-";
   private final String RECORD_A_eleOutMy     = "TR_出口有効電力#W#-";
   private final String RECORD_R_eleOutMy     = "TR_出口無効電力#Var#-";
   private final String RECORD_loadFactor     = "TR_負荷率#-#-";
   private final String RECORD_transLoss      = "TR_変圧器損失#W#電力 その他";
   private final String RECORD_transHeatGain  = "TR_変圧器発熱#W#-";
   private final String RECORD_transHeatGainZ = "TR_変圧器ゾーン熱負荷#W#-";
   private final String RECORD_transHeatGainZA= "TR_変圧器ゾーン熱負荷密度#W/m2#-";

   //仕様
   private final String SPEC_name             = "名称";
   private final String SPEC_phase            = "相数[-]";
   private final String SPEC_capacity         = "容量[VA]";
   private final String SPEC_heatGainFactor   = "内部発熱換算係数[-]";
```

```
private final String SPEC_noLoadLoss       = "無負荷損[W]";
private final String SPEC_loadLoss         = "負荷損[W]";
//20120723nino
private final String SPEC_capacitySubtotal = "容量小計[VA]";
//
private final String SPEC_grzName = "室グループ／室／ゾーン";
//
private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
private final String SPEC_maxItemCount = "最大同時表示ステップ数";//最大同時表示ステップ数
private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする

//仕様など
private StringBuffer message = new StringBuffer(); //message
private String name;          //名称
private int    phase;            //単相三相の別
private double capacity;         //変圧器容量
private double heatGainFactor;      //発熱係数
private double noLoadLoss;       //無負荷損
private double loadLoss;         //負荷損
private String  nameId;       //識別名
//20120723nino
private double capacitySubtotal; //変圧器容量の小計

//
private boolean isGVisible = false;//このグラフを表示する=true
private int  maxItemCount = 100;//最大同時表示ステップ数
private boolean isRecord   = false;//記録を有効とする=true

private BestElectricity eleIn;
private BestElectricity eleOut;
private BestElectricity eleLoss;//20091111

private BestHeat heaOut;//20091111

private double loadFactor;
private double transformerLoss;
private double heatGain;

private double powerFactor = 1.0;

//20120723nino
private double capacityRate=1;//=定格容量/定格容量の小計


private ISpace coreSpace   ;//建物側インスタンス
private double zoneArea;//ゾーンの床面積
private SystemHeatGain systemHeatGain; //建物側インスタンス
private boolean isSystemHeatGain = false;


//制御など
private int swcIn;         //off/on


//グラフ表示など
private GraphJFrameTrans20080909 gTrans = null;


public void setProfile( BestSpecs spec ){
   if(spec == null) {
      return;
   }

   Map<String, String> map = spec.getSpec();
   if(map == null) {
      return;
   }

   //名称を取得
   this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

   //定格容量を取得
```

```
    this.capacity = spec.getSpecValue( this.SPEC_capacity, 1000. );
    if( this.capacity == 0 ){
       this.capacity = 1000.;
    }

    //定格容量の小計を取得  20120723nino
    this.capacitySubtotal = spec.getSpecValue( this.SPEC_capacitySubtotal, this.capacity );
    if( this.capacitySubtotal == 0 ){
       if( this.isRecord ){
          this.message.append( "(W)定格容量=0のため計算不可能→1000");
       }
       this.capacitySubtotal = this.capacity;
       this.capacityRate = 1;
    }else{
       this.capacityRate = this.capacity / this.capacitySubtotal;
    }

    //相数を取得
    this.phase = spec.getSpecValue( this.SPEC_phase, 3 );

    //内部発熱換算係数を取得
    this.heatGainFactor = spec.getSpecValue( this.SPEC_heatGainFactor, 1000. );
    if( this.heatGainFactor == 0 ){
       if( this.isRecord ){
          this.message.append( "(W)内部発熱換算係数=0のため計算不可能→1000");
       }
       this.heatGainFactor = 1000;
    }

    //無負荷損を取得
    this.noLoadLoss = spec.getSpecValue( this.SPEC_noLoadLoss, 0. );

    //負荷損を取得
    this.loadLoss = spec.getSpecValue( this.SPEC_loadLoss, 0. );

    //室グループ、ゾーン名の取得
    String grzName = null;
    grzName = spec.getSpecValue( this.SPEC_grzName, "" );
    if( grzName.equals( "" ) ){
       this.isSystemHeatGain = false;
    }else{
       this.isSystemHeatGain = true;//systemHeatGain で発熱をゾーンへ渡す
       //SystemHeatGainインスタンスを建物側に登録
       this.coreSpace = ZoneGRZName.getCoreSpace( grzName, this.name );
       this.zoneArea = this.coreSpace.getValue( ISpace.FLOORAREA );
       this.systemHeatGain = new SystemHeatGain();
       this.systemHeatGain.setProfile(
             ZoneGRZName.getMultiSpaceName( grzName, this.name ),
             ZoneGRZName.getZoneName( grzName, this.name ) );
    }

    //isGVisibleを取得
    this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

    // 最大同時表示ステップ数
    this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

    //isRecordを取得
    this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

}


public void initialize(
      IBestStateMessage stateNodes,
      IBestControlMessage commandNodes,
      IBestRecordMessage recordNodes,
      IBestDomainSchedule schedule) {

    //状態ノードを受け取る
    super.sm = stateNodes;
    //制御ノードを受け取る
```

```
    super.cm = commandNodes;
    //記録ノードを受け取る
    super.rm = recordNodes;
    //スケジュールを受け取る
    super.ds = schedule;

    //接続ノード
    //入口
    this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

    //出口
    this.eleOut = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleOut );

    //Loss
    this.eleLoss = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleLoss );

    //HeaOut
    if( super.sm.getState( super.getConnectionNode( this.S_NODE_heaOut )) != null ){
       this.heaOut = (BestHeat) super.sm.getState( super.getConnectionNode( this.S_NODE_heaOut));
    }else{

       this.heaOut = new BestHeat();
       super.sm.setState( super.getConnectionNode( this.S_NODE_heaOut), this.heaOut );
    }

    //グラフ表示の準備
    if( this.isGVisible ){
       this.gTrans = new GraphJFrameTrans20080909( this.name, this.maxItemCount, this.capacity * 1.5, 0 );
    }

}


double eleOut_MyactivePower  ;// = this.eleOut.getActivePower() * this.capacityRate;
double eleOut_MyreactivePower;// = this.eleOut.getReactivePower() * this.capacityRate;


public void outputs(){

    if (null == super.sm && null == super.cm && null == super.ds)
       return;

    this.swcIn = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ) );
    this.swcIn = 1;   //今は常時動作中

    this.eleOut
    = (BestElectricity)super.sm.getState( super.getConnectionNode( this.S_NODE_eleOut ));

    //定格容量の小計　から　担当負荷を算定 20120723nino
    //this.eleOut.setActivePower( this.eleOut.getActivePower() * this.capacityRate );
    //this.eleOut.setReactivePower( this.eleOut.getReactivePower() * this.capacityRate );
    this.eleOut_MyactivePower   = this.eleOut.getActivePower() * this.capacityRate;
    this.eleOut_MyreactivePower = this.eleOut.getReactivePower() * this.capacityRate;

    if( this.swcIn == 0 ){  //動作停止の時　上流側を＝０とする　現在、下流側は無対策
       if( this.isRecord ){
          this.message.append( "(C)動作停止中");
       }
       this.eleIn.setActivePower( 0. );
       this.eleIn.setReactivePower( 0. );

       //負荷率を計算します。
       this.loadFactor = 0.;

       //変圧器損失を計算します。
       this.transformerLoss = 0.;

    }else{
       if( this.isRecord ){
          this.message.append( "(C)動作中");
       }
```

```
        //負荷率を計算します。
        //this.loadFactor =
        //  Math.pow( this.eleOut.getActivePower()*this.eleOut.getActivePower() +
        //  this.eleOut.getReactivePower()* this.eleOut.getReactivePower(), 0.5 )
        //   / this.capacity;
        this.loadFactor =
               Math.pow( this.eleOut_MyactivePower * this.eleOut_MyactivePower +
                     this.eleOut_MyreactivePower * this.eleOut_MyreactivePower, 0.5 )
                / this.capacity;

        //変圧器損失を計算します。
        this.transformerLoss = this.noLoadLoss + this.loadLoss * Math.pow(this.loadFactor,2);

        //
        this.eleIn.copyAllState( this.eleOut );
        //this.eleIn.setActivePower( this.eleOut.getActivePower() + this.transformerLoss );
        //this.eleIn.setReactivePower(
        //    this.eleIn.getActivePower() * Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 ) );
        this.eleIn.setActivePower( this.eleOut_MyactivePower + this.transformerLoss );
        this.eleIn.setReactivePower(
              this.eleOut_MyactivePower * Math.pow( 1/this.powerFactor/ this.powerFactor - 1, 0.5 ) );

    }

    //発熱を計算します。
    this.heatGain = this.transformerLoss * this.heatGainFactor;

    //発熱の受渡し方法
    if( this.isSystemHeatGain ){
        //建物側へ熱量[W]を渡す　すべて対流
        this.systemHeatGain.integrateHeatGain(
              this.heatGain, 0, 0, BestTimeManager.getCurrentInterval() );
        //接続媒体で渡す＝０
        this.heaOut.setConvectiveHeatRate( 0. );
    }else{
        //建物側へ熱量[W]を渡す　すべて対流＝０
        //this.systemHeatGain.integrateHeatGain(
        //    0, 0, 0, BestTimeManager.getCurrentInterval() );
        //接続媒体で渡す
        this.heaOut.setConvectiveHeatRate( this.heatGain );
    }


    //20091111
    this.eleLoss.setActivePower( this.transformerLoss );
    this.eleLoss.setReactivePower( 0. );


    //グラフデータ追加
    if( this.isGVisible ){
        gTrans.addData( BestTimeManager.getDateWeatherTime(),
              this.swcIn,
              this.eleOut,
              this.capacity,
              this.eleIn,
              this.heatGain );
    }

    //記録
    if( this.isRecord  && super.rm != null ){
        this.record();
    }

    message.setLength(0);
}

/**
 * 記録
 */
private void record(){

    //if( CheckPrintModule.isPrintMessage ){
```

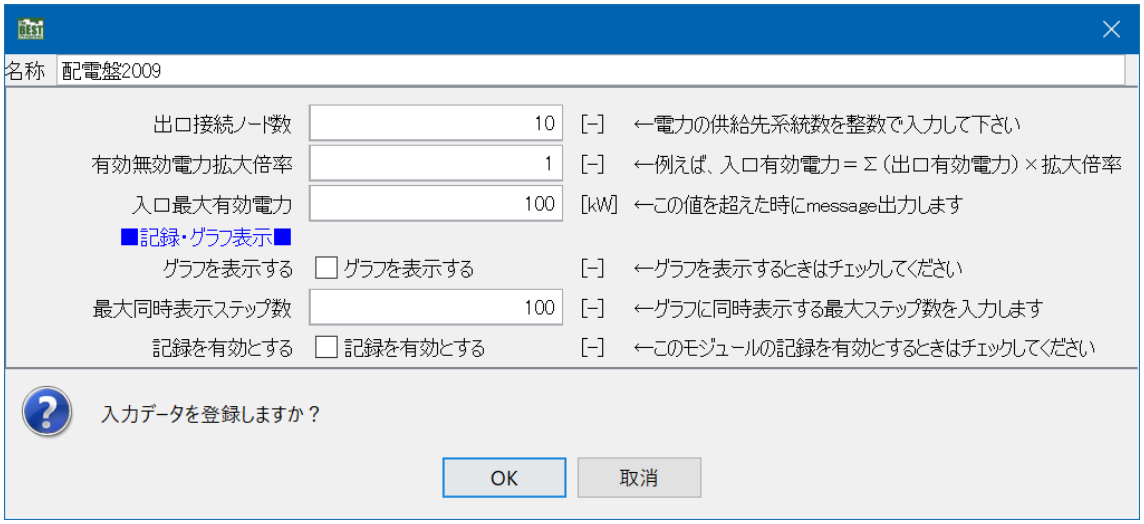
```
    //}

    //if( CheckPrintModule.isPrintEnergy ){
    //}

    if( CheckPrintModule.isPrintLoad ){
       if( this.isSystemHeatGain ){
          //記録ノードにゾーン熱負荷を設定
          super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                this.RECORD_transHeatGainZ, this.name, this.heatGain );
          if( this.zoneArea != 0 ){
             super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                   this.RECORD_transHeatGainZA, this.name, this.heatGain/this.zoneArea );
          }
       }
    }

    if( CheckPrintModule.isPrintStateOut ){
       //eleOut
       super.rm.setRecord(super.getConnectionNode(this.R_NODE),
             this.RECORD_A_eleOutMy, this.name, this.eleOut_MyactivePower );
       super.rm.setRecord(super.getConnectionNode(this.R_NODE),
             this.RECORD_R_eleOutMy, this.name, this.eleOut_MyreactivePower );
       super.rm.setRecord(super.getConnectionNode(this.R_NODE),
             this.RECORD_A_eleOutALL, this.name, this.eleOut.getActivePower() );
       super.rm.setRecord(super.getConnectionNode(this.R_NODE),
             this.RECORD_R_eleOutALL, this.name, this.eleOut.getReactivePower() );
    }

    //if( CheckPrintModule.isPrintStateOut ){
    //}

    if( CheckPrintModule.isPrintStateMy ){
       //記録ノードに負荷率を設定
       super.rm.setRecord(super.getConnectionNode(this.R_NODE),
             this.RECORD_loadFactor, this.name, loadFactor );
       //記録ノードに変圧器損失を設定
       super.rm.setRecord(super.getConnectionNode(this.R_NODE),
             this.RECORD_transLoss, this.name, transformerLoss );
       //記録ノードに発熱を設定
       super.rm.setRecord(super.getConnectionNode(this.R_NODE),
             this.RECORD_transHeatGain, this.name, heatGain );
    }

    if( CheckPrintModule.isPrintStateIn ){
       //eleIn
       super.rm.setRecord(super.getConnectionNode(this.R_NODE),
             this.RECORD_A_eleIn,this.name,  this.eleIn.getActivePower() );
       super.rm.setRecord(super.getConnectionNode(this.R_NODE),
             this.RECORD_R_eleIn, this.name, this.eleIn.getReactivePower() );
    }
 }


 public void update() {

 }
```

「配電盤 2009」（場所：設備 2015／電気設備 2015／）

モジュール名	配電盤 2009
クラス	DistributionBoad_nOut1InModule20090101

（1）入力画面

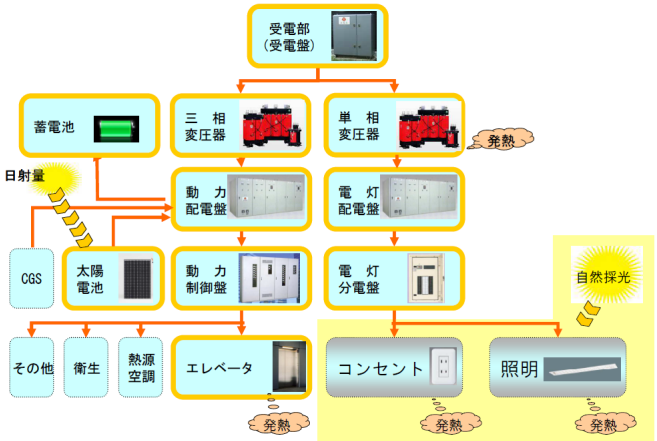
・スペック



（2）モジュールの概要

電力を分配するモジュールで、出口側に接続された電力を集計し、入口電力を設定する。

下図、電気設備の主要モジュール構成を参照。



（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

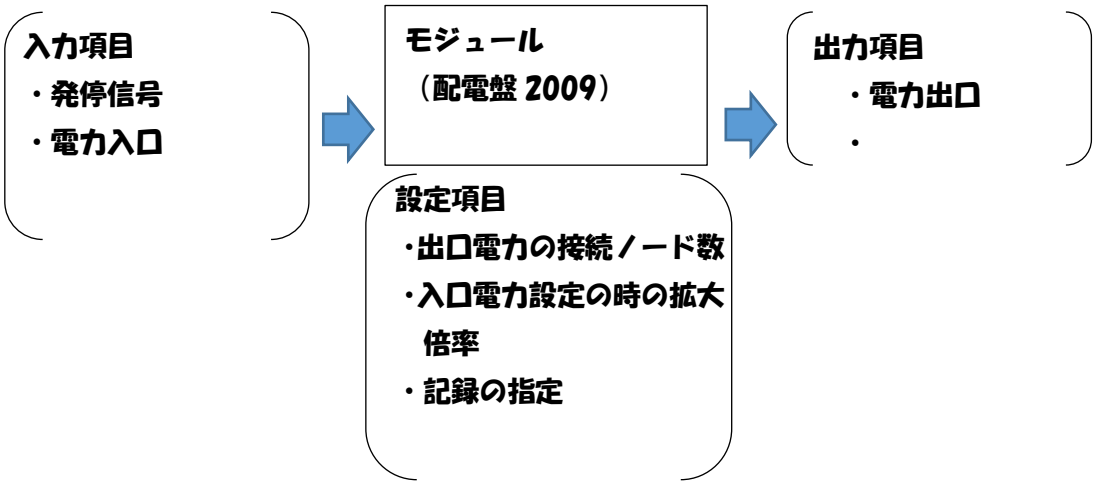


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String		ー	[-]	ー	ー		
1	出口接続ノード数	int	numberOfOutlet	10	[-]	ー	1		←電力の供給先系統数を整数で入力して下さい
2	有効無効電力拡大倍率	double	enlarge	1	[-]	ー	0		←例えば、入口有効電力＝Σ（出口有効電力）×拡大倍率
3	入口最大有効電力	double	eleIn_maxActivePower	100	[kW]	ー	0		←この値を超えた時に message 出力します 計算には影響しない。
4	■記録・グラフ表示■					ー	ー		
5	グラフを表示する	boolean	isGVisible	FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
6	最大同時表示ステップ数	int	maxItemCount	100	[-]	ー	ー		←グラフに同時表示する最大ステップ数を入力します
7	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	空調記録モジュールへ接続する
2	発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	上位からの発停信号
3	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	
4	電力出口[]	L0_eleOut[]	eleOut[]	-	状態	電力	出口	スペックの出口接続ノード数で入力された数のノードが用意される
5	電力デマンド	L0_valOutDemandele	valOutDemandele	-	値	値	出口	電力デマンドを値で出す

（6）記録項目

No.	項目	出力内容	単位	分類
1	DBoad_nOut1InMessage#-#-	メッセージ	—	メッセージ
2	DBoad_nOut1In 入口有効電力#W#-	入口有効電力	W	入口
3	DBoad_nOut1In 入口無効電力#Var#-	入口無効電力	Var	入口
4	DBoad_nOut1InLO_eleOut[i]有効電力#W#-	各出口有効電力	W	出口
5	DBoad_nOut1InLO_eleOut[i]無効電力#Var#-	各出口無効電力	Var	出口

（７）計算フロー・計算内容

・swcln による動作

swcln が停止信号の時

電力は流れないとして次の処理を行う。

eleIn および各 eleOut[]の有効電力、無効電力、電流を＝０とする。

swcln が運転信号の時

・出口電力の有効電力と無効電力を集計する。

出口有効電力合計＝Σ eleOut[i]有効電力

出口無効電力合計＝Σ eleOut[i]無効電力

・拡大倍率の適用

電力デマンド有効電力＝出口有効電力合計×有効無効電力拡大倍率

電力デマンド無効電力＝出口無効電力合計×有効無効電力拡大倍率

・入口電力の算定

入口有効電力＝出口有効電力合計

入口無効電力＝出口無効電力合計

（８）データ範囲と範囲外の取扱い

特になし。

参考　ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.Map;

import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * @author HIROSHI NINOMIYA /20080909
 * 分電盤です
 *   入口１系統
 *   出口ｎ系統
 *   出口側の電力を積算し入口側へ伝えます
 * 記録の有効無効判断を追加
 * 20090101 グラフ表示ステップ数追加　デマンドノード追加
 */
public class DistributionBoad_nOut1InModule20090101 extends AbstractBestModule implements
                        IBestMetaModule {

            //
            private final String moduleName = "(DistributionBoad_nOut1InModule20090101) ";

            //接続ノード
            private final String S_NODE_eleIn = "L0_eleIn";
            private final String S_NODE_valOutDemandele = "L0_valOutDemandele";

            //
            //出口　接続名称は　L0_eleOut[0],L0_eleOut[1]・・・・とする
            //
            private String[] S_NODE_eleOut;

            private final String C_NODE_swcIn = "L1_swcIn";
            //private final String C_NODE_modIn = "L1_modIn";

            private final String R_NODE = "L2_recOut";

            //仕様
            private final String SPEC_name = "名称";
            private final String SPEC_NumberOfOutlet = "[]出口接続ノード数";
            private final String SPEC_eleIn_maxActivePower = "入口最大有効電力[W]";

            private final String SPEC_enlarge = "有効無効電力拡大倍率";
            //
            private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
            private final String SPEC_maxItemCount = "最大同時表示ステップ数";//最大同時表示ステップ数
            private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする


            //記録
            private final String RECORD_message = "DBoad_nOut1InMessage#-#-";
            private final String RECORD_A_eleIn = "DBoad_nOut1In入口有効電力#W#-";
            private final String RECORD_R_eleIn = "DBoad_nOut1In入口無効電力#Var#-";

            //接続熱媒など
            private BestElectricity[] eleOut = null;
            private BestElectricity   eleIn = null;
            private BestValue valOutDemandele = null;
//          private BestElectricity   eleMy = null;

            //制御信号など
            private int swcIn;                                          //
```

```
//private int modIn;                                        //運転モード

//仕様など
private String name;                                        //名称
private int numberOfOutlet;                                 //出口　接続数

private double eleIn_maxActivePower;                        //入口　最大有効電量[W]
private double enlarge;//= "有効無効電力拡大倍率";
//
private boolean isGVisible = false;//このグラフを表示する=true
private int  maxItemCount = 100;//最大同時表示ステップ数
private boolean isRecord   = false;//記録を有効とする=true

private StringBuffer message = new StringBuffer();


//グラフ表示など
private GraphJFrameD_Boad20080909 gD_Boad = null;

private boolean isCalcEleOut = false;


@Override
public void setProfile(BestSpecs spec) {
            if(spec == null) {
                        return;
            }

            Map<String, String> map = spec.getSpec();
            if(map == null) {
                        return;
            }

            //名称を取得
            this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

            //numberOfOutletを取得
            if (null != map.get(this.SPEC_NumberOfOutlet)) {
                        this.numberOfOutlet
                        = Integer.parseInt( (String)map.get( this.SPEC_NumberOfOutlet ));
                        //出口ｎ系統のBestWaterの初期化
                        if( this.numberOfOutlet > 0 ){
                                    this.isCalcEleOut = true;
                                    this.eleOut = new BestElectricity[this.numberOfOutlet];
                                    this.S_NODE_eleOut = new String[this.numberOfOutlet];
                                    for( int i=0; i<this.eleOut.length; i++ ){
                                                //this.eleOut[i] = new BestElectricity();
                                                //this.eleOut[i] = new BestElectricity( 0, 0, 200, 0, 3, 50);
                                                this.S_NODE_eleOut[i] = new String( "LO_eleOut[" + i + "]" );
                                    }
                        }

            }

            //SPEC_eleIn_maxActivePowerを取得
            this.eleIn_maxActivePower = spec.getSpecValue( this.SPEC_eleIn_maxActivePower, 0. );

            //SPEC_enlarge = "有効無効電力拡大倍率";
            this.enlarge = spec.getSpecValue( this.SPEC_enlarge, 1. );

            //isGVisibleを取得
            this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

            // 最大同時表示ステップ数
            this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

            //isRecordを取得
            this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

}

public void initialize(IBestStateMessage stateNodes,
```

```
                                    IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
                                    IBestDomainSchedule schedule) {
                        //状態ノードを取得
                        super.sm = stateNodes;

                        //制御ノードを取得
                        super.cm = commandNodes;

                        //記録ノードを取得
                        super.rm = recordNodes;

                        //接続ノード　出口
                        if( this.isCalcEleOut ){
                                    for( int i=0; i<this.eleOut.length; i++ ){
                                                this.eleOut[i] = BestElectricity.bindnode( super.transferMapState,
super.sm, this.S_NODE_eleOut[i] );
                                                //System.out.println( "!  "+1 );
                                    }
                        }

                        //接続ノード　入口
                        //eleIn = new BestWater();
                        this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

                        //valOutDemandele;
                        this.valOutDemandele = BestValue.bindnode( super.transferMapState, super.sm,
this.S_NODE_valOutDemandele );

//                      this.eleMy = new BestElectricity();

                        //グラフ表示の準備
                        if( this.isGVisible ){
                                    this.gD_Boad = new GraphJFrameD_Boad20080909( this.name, this.maxItemCount,
this.eleOut.length, this.eleIn_maxActivePower * 1.5, this.eleIn_maxActivePower / this.enlarge / this.eleOut.length );
                        }
            }

            public void outputs() {
                        //if( super.sm == null || super.cm == null ) {
                        //          return;
                        //}
                        //接続　入口状態値の取得　（電力は出口）
                        if( this.isCalcEleOut ){
                                    for( int i=0; i<this.eleOut.length; i++ ){
                                                this.eleOut[i]
                                                = (BestElectricity)
super.sm.getState( super.getConnectionNode( this.S_NODE_eleOut[i] ));
                                    }
                        }

                        this.swcIn = super.cm.getCommand(super.getConnectionNode( this.C_NODE_swcIn ));

                        double activePower;
                        double reactivePower;

                        activePower = 0.;
                        reactivePower = 0.;

                        if( this.isCalcEleOut ){
                                    this.eleIn.copyAllState( this.eleOut[0] );                  //eleOut[]の最初の接続ele
を採用
                        }

                        //if(this.swcIn == 0 ){             //利用不可　遮断時
                        if( Airswc.isOFF( this.swcIn ) ){               //利用不可　遮断時
                                    if( this.isRecord ){
                                                this.message.append( "(C)遮断中");
                                    }
                                    this.eleIn.setActivePower( 0. );
                                    this.eleIn.setCurrent( 0. );
                                    this.eleIn.setReactivePower( 0. );
```

```
                        if( this.isCalcEleOut ){
                                for( int i = 0; i < this.eleOut.length; i++ ){
                                        this.eleOut[i].setActivePower( 0. );
                                        this.eleOut[i].setReactivePower( 0. );
                                        this.eleOut[i].setCurrent( 0. );
                                }
                        }

                }else{      //利用可能　通電時
                        if( this.isRecord ){
                                this.message.append( "(C)通電中");
                        }
                        // 集計
                        //入口のactivePower,reactivePower
                        //System.out.println( "eleOut length=" + this.eleOut.length );

                        if( this.isCalcEleOut ){
                                for( int i = 0; i < this.eleOut.length; i++ ){
                                        //System.out.println( "eleOut i=" + i);
                                        //if( this.eleOut[i] == null )continue;
                                        //System.out.println( "eleOut i=" + i +" "+this.name);
                                        activePower += this.eleOut[i].getActivePower();
                                        reactivePower += this.eleOut[i].getReactivePower();
                                }
                        }

                        //拡大
                        activePower *= this.enlarge;
                        reactivePower *= this.enlarge;
                        //20090101
                        this.valOutDemandele.setValue( activePower );//負荷の合計に倍率を掛けた値をデマンドと
する
                        if( this.eleIn_maxActivePower < activePower ){
                                if( this.isRecord ){
                                        this.message.append( "(W)最大電力超過");
                                }
                        }

                        this.eleIn.setActivePower( activePower );
                        this.eleIn.setReactivePower( reactivePower );
                }


                        //watMyの状態値のセット
//                      this.eleMy.copyAllState( this.eleIn );

                        //ノードに設定
                        super.sm.setState(super.getConnectionNode( this.S_NODE_eleIn ),
                                        this.eleIn );
                        super.sm.setState(super.getConnectionNode( this.S_NODE_valOutDemandele ),
                                        this.valOutDemandele );

                        //記録
                        //グラフデータ追加
                        if( this.isGVisible ){
                                gD_Boad.addData( BestTimeManager.getDateWeatherTime(),
                                                this.swcIn,
                                                this.eleOut,
                                                this.eleIn_maxActivePower,
                                                this.eleIn );
                        }

                        //記録
                        if( this.isRecord  && super.rm != null ){
                                this.record();
                        }

                        message.setLength(0);
        }

        /**
         * 記録
```

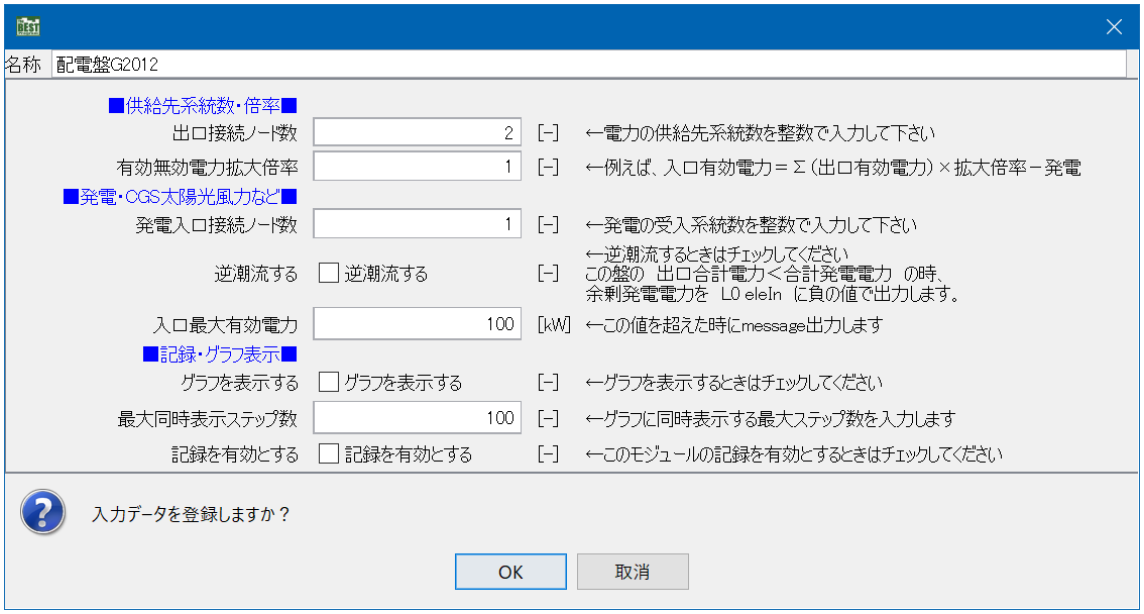
```
	 */
	private void record(){

		if( CheckPrintModule.isPrintMessage ){
			super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
					this.RECORD_message, this.name, this.message );
		}

		//if( CheckPrintModule.isPrintEnergy ){
		//}

		//if( CheckPrintModule.isPrintLoad ){
		//}

		if( CheckPrintModule.isPrintStateOut ){
			//出口
			if( this.isCalcEleOut ){
				for( int i=0; i<eleOut.length; i++ ){
					super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
							this.S_NODE_eleOut[i] + "有効電力#W#
電力",this.name,

	AirFormat.df_2( this.eleOut[i].getActivePower() ));
					super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
							this.S_NODE_eleOut[i] + "無効電力
#Var#無効電力",this.name,

	AirFormat.df_2( this.eleOut[i].getReactivePower() ));
				}
			}
		}

		//if( CheckPrintModule.isPrintStateMy ){
		//}

		if( CheckPrintModule.isPrintStateIn ){
			//入口
			super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
					this.RECORD_A_eleIn, this.name,
					AirFormat.df_2( this.eleIn.getActivePower() ));
			super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
					this.RECORD_R_eleIn, this.name,
					AirFormat.df_2( this.eleIn.getReactivePower() ));
		}
	}

	public void update() {

	}

	public Object viewInternal(TestCommand cmd) {
		// とりあえずクラス変数のリストを返します。
		java.util.List<Object> result = new java.util.ArrayList<Object>();

		//接続ノード
		result.add(super.sm);
		result.add(super.cm);

		//外部定義項目
		result.add(eleIn_maxActivePower);

		if( eleOut != null ){
			for( int i = 0; i < eleOut.length; i++ ){
				result.add( eleOut[i] );
			}
		}
//		result.add(this.eleMy);

		return result;
	}
}
```

「配電盤 G2012」（場所：設備 2015／電気設備 2015／）

モジュール名	配電盤 G2012
クラス	DistributionBoad_nOut1InnInGenModule20120202

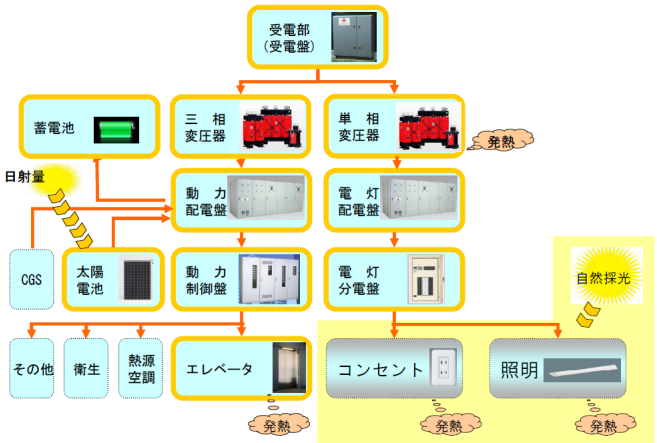
（1）入力画面

・スペック



（2）モジュールの概要

電力を分配するモジュールで、出口側に接続された電力を集計し、発電電力とから入口電力を設定する。下図、電気設備の主要モジュール構成を参照。



（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュ

ールからの出力項目」を下記に示す。

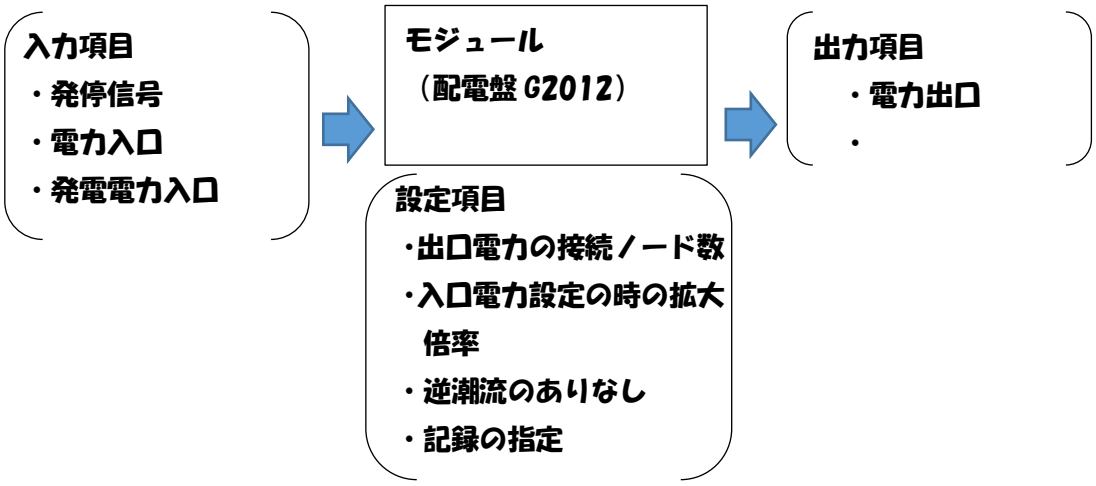


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	Name	ー	[-]	ー	ー		
1	■供給先系統数・倍率■					ー	ー		
2	出口接続ノード数	int	numberOfOutlet	10	[-]	ー	1		←電力の供給先系統数を整数で入力して下さい
3	有効無効電力拡大倍率	double	enlarge	1	[-]	ー	0		←例えば、入口有効電力 ＝Σ（出口有効電力）×拡大倍率－発電
4	■発電・CGS 太陽光風力など■					ー	ー		
5	発電入口接続ノード数	int	numberOfInletGen	1	[-]	ー	0		←発電の受入系統数を整数で入力して下さい
6	逆潮流する	boolean	isBackwardFlow	FALSE	[-]	ー	ー		←逆潮流するときはチェックしてください。 この盤の　出口合計電力＜合計発電電力　の時、余剰発電電力を　LO_eleIn　に負の値で出力します。
7	入口最大有効電力	double	eleIn_maxActivePower	100	[kW]	ー	0		←この値を超えた時に message 出力します。 計算には影響しない。
8	■記録・グラフ表示■					ー	ー		
9	グラフを表示する	boolean	isGVisible	FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
10	最大同時表示ステップ数	int	maxItemCount	100	[-]	ー	ー		←グラフに同時表示する最大ステップ数を入力します
11	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	空調記録モジュールへ接続する
2	発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	上位からの発停信号
3	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	
4	電力出口	L0_eleOut[]	eleOut[]	-	状態	電力	出口	スペックの出口接続ノード数で入力された数のノードが用意される
5	発電電力入口	L0_eleInGen[]	eleInGen[]	-	状態	電力	入口	スペックの発電入口接続ノード数で入力された数のノードが用意される
6	電力デマンド	L0_valOutDemandele	valOutDemandele	-	値	値	出口	電力デマンドを値で出す

（6）記録項目

No.	項目	出力内容	単位	分類
1	DBoad_Message#-#-	メッセージ	—	メッセージ
2	DBoad_入口有効電力#W#-	入口有効電力	W	入口
3	DBoad_入口無効電力#Var#-	入口無効電力	Var	入口
4	DBoad_発電有効電力合計#W#-	発電有効電力の合計値	W	My
5	DBoad_余剰電力#W#-	余剰電力	W	My
6	DBoad_LO_eleOut[*]有効電力#W#-	各出口の有効電力	W	出口
7	DBoad_LO_eleOut[*]無効電力#Var#-	各出口の無効電力	Var	出口
8	DBoad_LO_eleInGen[*]有効電力#W#-	各発電入口の有効電力	W	入口
9	DBoad_LO_eleInGen[*]無効電力#Var#-	各発電入口の無効電力	Var	入口

（７）計算フロー・計算内容

・swcln による動作

swcln が停止信号の時

電力は流れないとして次の処理を行う。

eleIn および各 eleOut[]の有効電力、無効電力、電流を＝０とする。

swcln が運転信号の時

・出口電力の有効電力と無効電力を集計する。

出口有効電力合計＝Σ eleOut[i]有効電力

出口無効電力合計＝Σ eleOut[i]無効電力

・拡大倍率の適用

電力デマンド有効電力＝出口有効電力合計×有効無効電力拡大倍率

電力デマンド無効電力＝出口無効電力合計×有効無効電力拡大倍率

・発電電力の有効電力と無効電力の集計

発電有効電力合計＝Σ eleInGen[i]有効電力

発電無効電力合計＝Σ eleInGen[i]無効電力

・入口電力の算定

入口有効電力＝出口有効電力合計―発電有効電力合計

入口無効電力＝出口無効電力合計―発電無効電力合計

入口有効電力<0 の時、余剰電力＝－入口有効電力が発生している。

入口有効電力<0 の時で逆潮流しない場合は、入口有効電力＝０とする。

入口無効電力<0 の時、入口無効電力＝０とする。

（８）データ範囲と範囲外の取扱い

特になし。

参考　ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.Map;

import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * @author HIROSHI NINOMIYA /20080909
 * 分電盤です
 *   入口商用１系統　発電n系統 /20120202
 *   出口ｎ系統
 *   出口側の電力を積算し入口側へ伝えます
 *   商用系統は発電系統分を差し引いた負荷とします
 * 記録の有効無効判断を追加
 * 20090101 グラフ表示ステップ数追加 デマンドノード追加
 */
public class DistributionBoad_nOut1InnInGenModule20120202 extends AbstractBestModule implements
      IBestMetaModule {

   //
   private final String moduleName = "(DistributionBoad_nOut1InnInGenModule20120202) ";

   //接続ノード
   private final String S_NODE_eleIn   = "L0_eleIn";

   private final String S_NODE_valOutDemandele = "L0_valOutDemandele";
   //
   //出口　接続名称は　L0_eleOut[0],L0_eleOut[1]・・・・とする
   //
   private String[] S_NODE_eleOut;
   private String[] S_NODE_eleInGen;

   private final String C_NODE_swcIn = "L1_swcIn";
   //private final String C_NODE_modIn = "L1_modIn";

   private final String R_NODE = "L2_recOut";

   //仕様
   private final String SPEC_name = "名称";
   private final String SPEC_NumberOfOutlet   = "[]出口接続ノード数";
   private final String SPEC_NumberOfInletGen = "[]発電入口接続ノード数";
   private final String SPEC_eleIn_maxActivePower = "入口最大有効電力[W]";

   private final String SPEC_enlarge = "有効無効電力拡大倍率";

   private final String SPEC_isBackwardFlow = "is逆潮流する";//このグラフを表示する
   //
   private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
   private final String SPEC_maxItemCount = "最大同時表示ステップ数";//最大同時表示ステップ数
   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする


   //記録
   private final String RECORD_message = "DBoad_Message#-#-";
   private final String RECORD_A_eleIn = "DBoad_入口有効電力#W#-";
   private final String RECORD_R_eleIn = "DBoad_入口無効電力#Var#-";
   private final String RECORD_sumGen  = "DBoad_発電有効電力合計#W#-";
   private final String RECORD_surplus = "DBoad_余剰電力#W#-";
```

```
    //接続熱媒など
    private BestElectricity[] eleOut = null;
    private BestElectricity   eleIn = null;
    private BestElectricity[] eleInGen= null;

    private BestValue valOutDemandele = null;
// private BestElectricity   eleMy = null;

    //制御信号など
    private int swcIn;                //
    //private int modIn;              //運転モード

    //仕様など
    private String name;          //名称
    private int numberOfOutlet;         //出口  接続数
    private int numberOfInletGen;       //入口  接続数

    private double eleIn_maxActivePower;      //入口  最大有効電量[W]
    private double enlarge;//= "有効無効電力拡大倍率";

    private boolean isBackwardFlow = false;//逆潮流する=true
    //
    private boolean isGVisible = false;//このグラフを表示する=true
    private int  maxItemCount = 100;//最大同時表示ステップ数
    private boolean isRecord   = false;//記録を有効とする=true

    private StringBuffer message = new StringBuffer();


    //グラフ表示など
    private GraphJFrameD_BoadG320081212 gD_Boad = null;

    //
    private double sumGen;//発電電力合計[W]
    private double surplus;//余剰電力[W]

    private boolean isCalcEleOut = false;
    private boolean isCalEleInGen = false;

    @Override
    public void setProfile(BestSpecs spec) {
       if(spec == null) {
          return;
       }

       Map<String, String> map = spec.getSpec();
       if(map == null) {
          return;
       }

       //名称を取得
       this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

       //numberOfOutletを取得
       if (null != map.get(this.SPEC_NumberOfOutlet)) {
          this.numberOfOutlet
          = Integer.parseInt( (String)map.get( this.SPEC_NumberOfOutlet ));
          //出口ｎ系統のBestWaterの初期化
          if( this.numberOfOutlet > 0 ){
             this.isCalcEleOut = true;
             this.eleOut = new BestElectricity[this.numberOfOutlet];
             this.S_NODE_eleOut = new String[this.numberOfOutlet];
             for( int i=0; i<this.eleOut.length; i++ ){
                //this.eleOut[i] = new BestElectricity();
                //this.eleOut[i] = new BestElectricity( 0, 0, 200, 0, 3, 50);
                this.S_NODE_eleOut[i] = new String( "LO_eleOut[" + i + "]" );
             }
          }
       }

       //numberOfInletGenを取得
```

```
    if (null != map.get(this.SPEC_NumberOfInletGen)) {
       this.numberOfInletGen
       = Integer.parseInt( (String)map.get( this.SPEC_NumberOfInletGen ));
       //出口ｎ系統のBestWaterの初期化
       if( this.numberOfInletGen > 0 ){
          this.isCalEleInGen = true;
          this.eleInGen = new BestElectricity[this.numberOfInletGen];
          this.S_NODE_eleInGen = new String[this.numberOfInletGen];
          for( int i=0; i<this.eleInGen.length; i++ ){
             //this.eleOut[i] = new BestElectricity();
             //this.eleOut[i] = new BestElectricity( 0, 0, 200, 0, 3, 50);
             this.S_NODE_eleInGen[i] = new String( "LO_eleInGen[" + i + "]" );
          }
       }
    }

    //SPEC_eleIn_maxActivePowerを取得
    this.eleIn_maxActivePower = spec.getSpecValue( this.SPEC_eleIn_maxActivePower, 0. );

    //SPEC_enlarge = "有効無効電力拡大倍率";
    this.enlarge = spec.getSpecValue( this.SPEC_enlarge, 1. );

    //isBackwardFlowを取得
    this.isBackwardFlow = spec.getSpecValue( this.SPEC_isBackwardFlow, false );

    //isGVisibleを取得
    this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

    // 最大同時表示ステップ数
    this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

    //isRecordを取得
    this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

 }

 public void initialize(IBestStateMessage stateNodes,
       IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
       IBestDomainSchedule schedule) {
    //状態ノードを取得
    super.sm = stateNodes;

    //制御ノードを取得
    super.cm = commandNodes;

    //記録ノードを取得
    super.rm = recordNodes;

    //接続ノード　出口
    if( this.isCalcEleOut ){
       for( int i=0; i<this.eleOut.length; i++ ){
          this.eleOut[i] = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleOut[i] );
       }
    }

    //接続ノード　入口
    //eleIn = new BestElectricity();
    this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

    //発電入口
    if( this.isCalEleInGen ){
       for( int i=0; i<this.eleInGen.length; i++ ){
          this.eleInGen[i] = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleInGen[i] );
       }
    }

    //valOutDemandele;
    this.valOutDemandele = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valOutDemandele );


    //グラフ表示の準備
    if( this.isGVisible ){
```

```
        this.gD_Boad = new GraphJFrameD_BoadG320081212( this.name, this.maxItemCount, this.eleOut.length,
this.numberOfInletGen, this.eleIn_maxActivePower * 1.5, this.eleIn_maxActivePower / this.enlarge / this.eleOut.length );
      }
   }

   double activePower;
   double reactivePower;

   public void outputs() {
      //if( super.sm == null || super.cm == null ) {
      // return;
      //}
      //接続　入口状態値の取得　（電力は出口）
      if( this.isCalcEleOut ){
         for( int i=0; i<this.eleOut.length; i++ ){
            this.eleOut[i]
            = (BestElectricity) super.sm.getState( super.getConnectionNode( this.S_NODE_eleOut[i] ));
         }
      }

      //発電系の入口は読み込む
      if( this.isCalEleInGen ){
         for( int i=0; i<this.eleInGen.length; i++ ){
            this.eleInGen[i]
            = (BestElectricity) super.sm.getState( super.getConnectionNode( this.S_NODE_eleInGen[i] ));
         }
      }

      this.swcIn = super.cm.getCommand(super.getConnectionNode( this.C_NODE_swcIn ));


      activePower = 0.;
      reactivePower = 0.;

      this.sumGen = 0;
      this.surplus = 0;

      if( this.isCalcEleOut ){
         this.eleIn.copyAllState( this.eleOut[0] );      //eleOut[]の最初の接続eleを採用
      }

      //if(this.swcIn == 0 ){    //利用不可　遮断時
      if( Airswc.isOFF( this.swcIn ) ){      //利用不可　遮断時
         if( this.isRecord ){
            this.message.append( "(C)遮断中");
         }
         this.eleIn.setActivePower( 0. );
         this.eleIn.setReactivePower( 0. );
         this.eleIn.setCurrent( 0. );

         if( this.isCalcEleOut ){
            for( int i = 0; i < this.eleOut.length; i++ ){
               this.eleIn.copyAllState( this.eleOut[0] );      //eleOut[]の最初の接続eleを採用

                this.eleOut[i].setActivePower( 0. );
                this.eleOut[i].setReactivePower( 0. );
                this.eleOut[i].setCurrent( 0. );
            }
         }

      }else{   //利用可能　通電時
         if( this.isRecord ){
            this.message.append( "(C)通電中");
         }
         // 集計
         //入口のactivePower,reactivePower
         //System.out.println( "eleOut length=" + this.eleOut.length );

         if( this.isCalcEleOut ){
            for( int i = 0; i < this.eleOut.length; i++ ){
               //this.eleIn.copyAllState( this.eleOut[0] );    //eleOut[]の最初の接続eleを採用
               //System.out.println( "eleOut i=" + i);
```

```
            //System.out.println( "eleOut i=" + i +" "+this.name);
            activePower += this.eleOut[i].getActivePower();
            reactivePower += this.eleOut[i].getReactivePower();
        }
    }

    //拡大
    activePower *= this.enlarge;
    reactivePower *= this.enlarge;

    this.valOutDemandele.setValue( activePower );//負荷の合計に倍率を掛けた値をデマンドとする

    //System.out.println( "DB valOutDemandele = "+ this.valOutDemandele.getValue() );

    if( this.isCalEleInGen ){
        for( int i=0; i<this.eleInGen.length; i++ ){
            this.sumGen  += this.eleInGen[i].getActivePower();
            reactivePower -= this.eleInGen[i].getReactivePower();
        }
    }

    activePower -= this.sumGen;


    if( activePower < 0 ){
        if( this.isRecord ){
            this.message.append("(C)発電余剰有り" + activePower );
        }
        this.surplus = -activePower;
        if( !this.isBackwardFlow ){
            //逆潮流しない
            activePower = 0;
            reactivePower = 0;
        }
    }
    if( reactivePower < 0 ){
        reactivePower = 0;
    }
    //200901010
    if( this.eleIn_maxActivePower < activePower ){
        if( this.isRecord ){
            this.message.append( "(W)最大電力超過");
        }
    }

    this.eleIn.setActivePower( activePower );
    this.eleIn.setReactivePower( reactivePower );

}


//ノードに設定
super.sm.setState(super.getConnectionNode( this.S_NODE_eleIn ),
        this.eleIn );
super.sm.setState(super.getConnectionNode( this.S_NODE_valOutDemandele ),
        this.valOutDemandele );

//記録
//グラフデータ追加
if( this.isGVisible ){
    gD_Boad.addData( BestTimeManager.getDateWeatherTime(),
            this.swcIn,
            this.eleOut,
            this.eleIn_maxActivePower,
            this.eleIn,
            this.eleInGen );
}

//記録
if( this.isRecord  && super.rm != null ){
    this.record();
}
```

```
    message.setLength(0);
}

/**
 * 記録
 */
private void record(){

    if( CheckPrintModule.isPrintMessage ){
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_message, this.name, this.message );
    }

    //if( CheckPrintModule.isPrintEnergy ){
    //}

    //if( CheckPrintModule.isPrintLoad ){
    //}

    if( CheckPrintModule.isPrintStateOut ){
        //出口
        if( this.isCalcEleOut ){
            for( int i=0; i<eleOut.length; i++ )
            {
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        "DBoad_" + this.S_NODE_eleOut[i] + "有効電力#W#電力",this.name,
                        AirFormat.df_2( this.eleOut[i].getActivePower() ));
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        "DBoad_" + this.S_NODE_eleOut[i] + "無効電力#Var#無効電力",this.name,
                        AirFormat.df_2( this.eleOut[i].getReactivePower() ));
            }
        }
    }

    if( CheckPrintModule.isPrintStateMy ){
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_sumGen, this.name, AirFormat.df_2( this.sumGen ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_surplus, this.name, AirFormat.df_2( this.surplus ));
    }

    if( CheckPrintModule.isPrintStateIn ){
        //入口
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_A_eleIn, this.name,
                AirFormat.df_2( this.eleIn.getActivePower() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_R_eleIn, this.name,
                AirFormat.df_2( this.eleIn.getReactivePower() ));
        //発電入口
        if( this.isCalEleInGen ){
            for( int i=0; i<eleInGen.length; i++ )
            {
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        "DBoad_" + this.S_NODE_eleInGen[i] + "有効電力#W#電力",this.name,
                        AirFormat.df_2( this.eleInGen[i].getActivePower() ));
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        "DBoad_" + this.S_NODE_eleInGen[i] + "無効電力#Var#無効電力",this.name,
                        AirFormat.df_2( this.eleInGen[i].getReactivePower() ));
            }
        }
    }
}


public void update() {

}

public Object viewInternal(TestCommand cmd) {
    // とりあえずクラス変数のリストを返します。
```

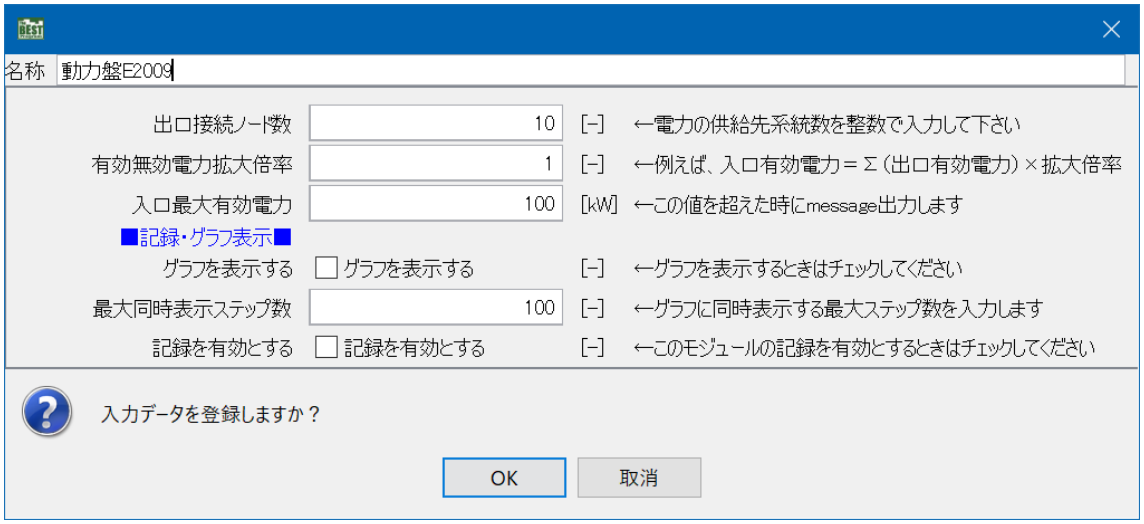
```
        java.util.List<Object> result = new java.util.ArrayList<Object>();

        //接続ノード
        result.add(super.sm);
        result.add(super.cm);

        //外部定義項目
        result.add(eleIn_maxActivePower);

        if( eleOut != null ){
           for( int i = 0; i < eleOut.length; i++ ){
              result.add( eleOut[i] );
           }
        }
//      result.add(this.eleMy);

        return result;
    }
}
```

「動力盤 E2009」（場所：設備 2015／電気設備 2015／）

モジュール名	動力盤 E2009
クラス	DistributionBoad_nOut1InModule20090101

（1）入力画面

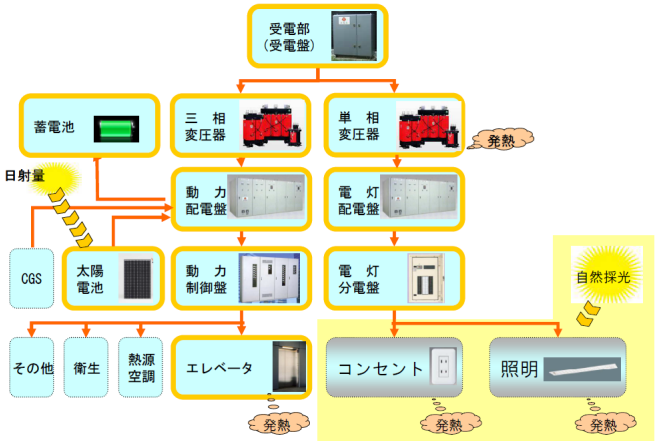
・スペック



（2）モジュールの概要

電力を分配するモジュールで、出口側に接続された電力を集計し、入口電力を設定する。

下図、電気設備の主要モジュール構成を参照。



（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

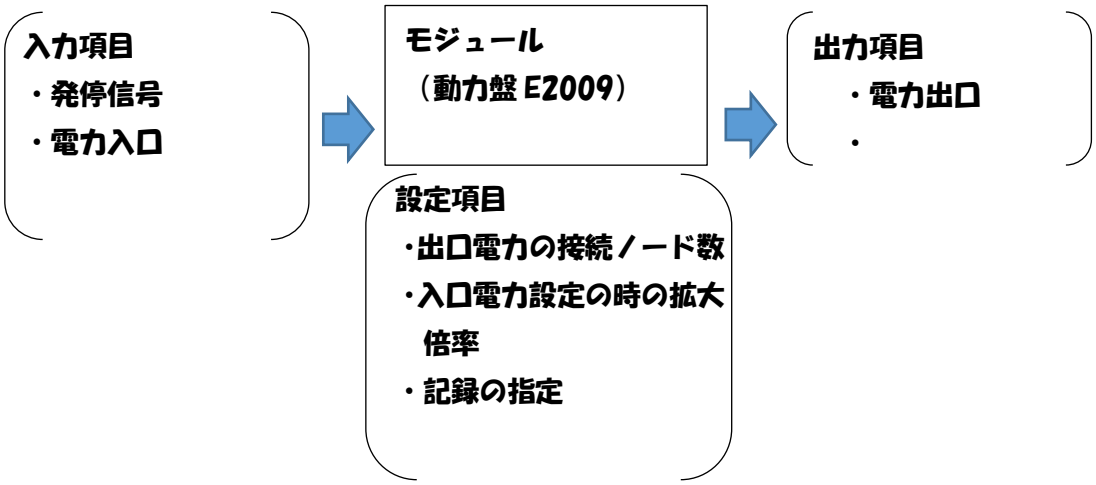


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String		―	[-]	―	―		
1	出口接続ノード数	int	numberOfOutlet	10	[-]	―	1		←電力の供給先系統数を整数で入力して下さい
2	有効無効電力拡大倍率	double	enlarge	1	[-]	―	0		←例えば、入口有効電力＝Σ（出口有効電力）×拡大倍率
3	入口最大有効電力	double	eleIn_maxActivePower	100	[kW]	―	0		←この値を超えた時に message 出力します 計算には影響しない
4	■記録・グラフ表示■					―	―		
5	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
6	最大同時表示ステップ数	int	maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
7	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェック してください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	空調記録モジュールへ接続する
2	発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	上位からの発停信号
3	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	
4	電力出口[]	L0_eleOut[]	eleOut[]	-	状態	電力	出口	スペックの出口接続ノード数で入力された数のノードが用意される
5	電力デマンド	L0_valOutDemandele	valOutDemandele	-	値	値	出口	電力デマンドを値で出す

（6）記録項目

No.	項目	出力内容	単位	分類
1	DBoad_nOut1InMessage#-#-	メッセージ	—	メッセージ
2	DBoad_nOut1In 入口有効電力#W#-	入口有効電力	W	入口
3	DBoad_nOut1In 入口無効電力#Var#-	入口無効電力	Var	入口
4	DBoad_nOut1InLO_eleOut[i]有効電力#W#-	各出口有効電力	W	出口
5	DBoad_nOut1InLO_eleOut[i]無効電力#Var#-	各出口無効電力	Var	出口

（７）計算フロー・計算内容

・swcln による動作

swcln が停止信号の時

電力は流れないとして次の処理を行う。

eleIn および各 eleOut[]の有効電力、無効電力、電流を＝０とする。

swcln が運転信号の時

・出口電力の有効電力と無効電力を集計する。

出口有効電力合計＝Σ eleOut[i]有効電力

出口無効電力合計＝Σ eleOut[i]無効電力

・拡大倍率の適用

電力デマンド有効電力＝出口有効電力合計×有効無効電力拡大倍率

電力デマンド無効電力＝出口無効電力合計×有効無効電力拡大倍率

・入口電力の算定

入口有効電力＝出口有効電力合計

入口無効電力＝出口無効電力合計

（８）データ範囲と範囲外の取扱い

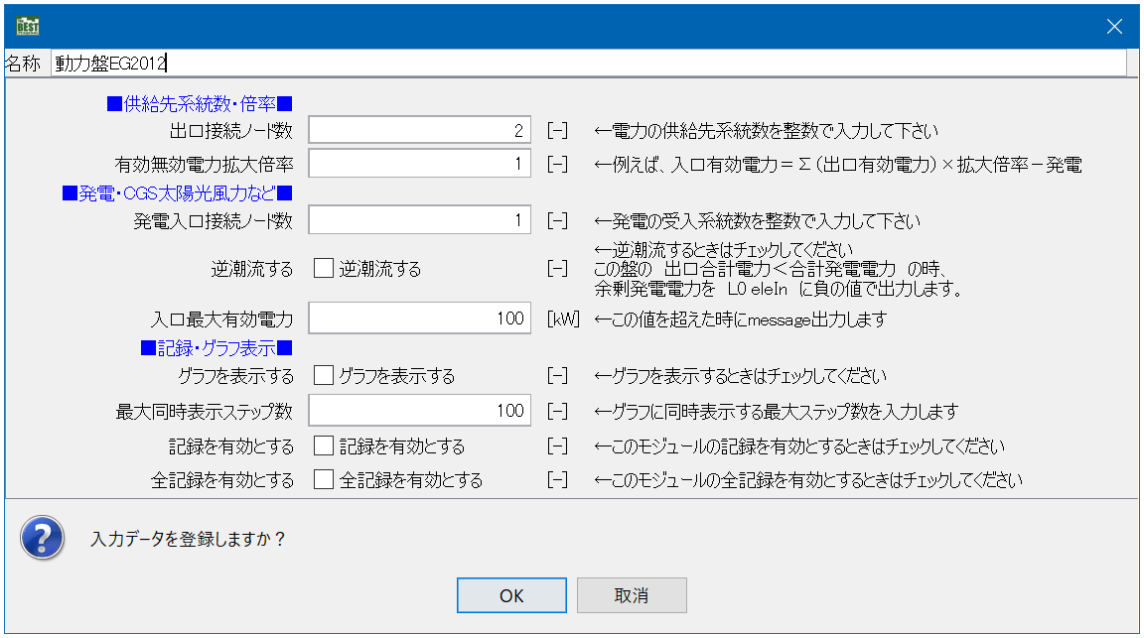
特になし。

「動力盤 EG2012」（場所：設備 2015／電気設備 2015／）

モジュール名	動力盤 EG2012
クラス	DistributionBoad_nOut1InnInGenModule20120202

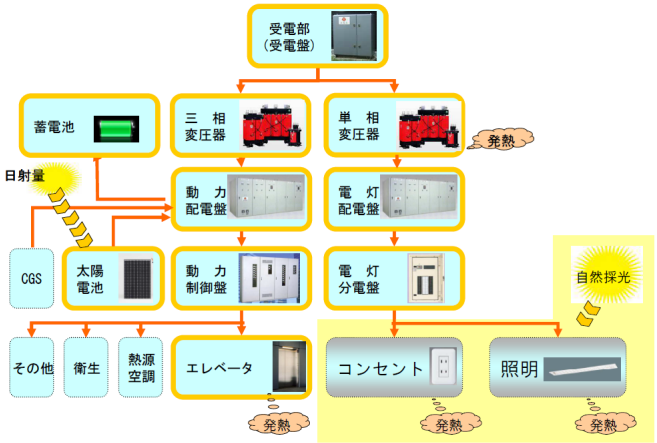
（1）入力画面

・スペック



（2）モジュールの概要

電力を分配するモジュールで、出口側に接続された電力を集計し、発電電力とから入口電力を設定する。下図、電気設備の主要モジュール構成を参照。



（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

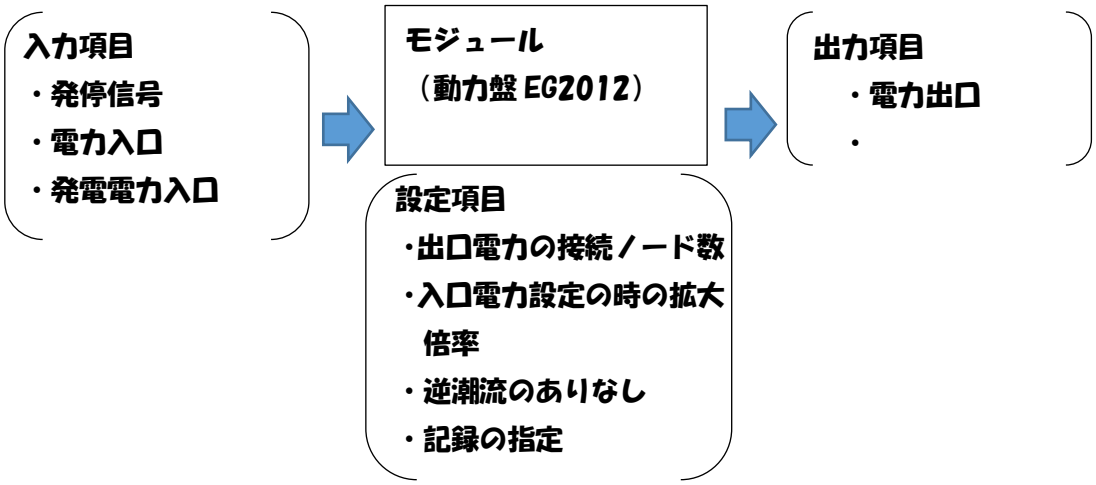


図１　モジュールの設定項目・入力項目・出力項目

（４）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String		―	[-]	―	―		
1	■供給先系統数・倍率■					―	―		
2	出口接続ノード数	int	numberOfOutlet	10	[-]	―	1		←電力の供給先系統数を整数で入力して下さい
3	有効無効電力拡大倍率	double	enlarge	1	[-]	―	0		←例えば、入口有効電力＝Σ（出口有効電力）×拡大倍率－発電
4	■発電・CGS 太陽光風力など■					―	―		
5	発電入口接続ノード数	int	numberOfInletGen	1	[-]	―	0		←発電の受入系統数を整数で入力して下さい
6	逆潮流する	boolean	isBackwardFlow	FALSE	[-]	―	―		←逆潮流するときはチェックしてください この盤の　出口合計電力＜合計発電電力　の時、余剰発電電力を　LO_eleIn　に負の値で出力します。
7	入口最大有効電力	double	eleIn_maxActivePower	100	[kW]	―	0		←この値を超えた時に message 出力します 計算には影響しない
8	■記録・グラフ表示■					―	―		
9	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
10	最大同時表示ステップ数	int	maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
11	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください
12	全記録を有効とする	boolean	isRecordALL	FALSE	[-]	―	―		←このモジュールの全記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	空調記録モジュールへ接続する
2	発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	上位からの発停信号
3	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	
4	電力出口	L0_eleOut[]	eleOut[]	-	状態	電力	出口	スペックの出口接続ノード数で入力された数のノードが用意される
5	発電電力入口	L0_eleInGen[]	eleInGen[]	-	状態	電力	入口	スペックの発電入口接続ノード数で入力された数のノードが用意される
6	電力デマンド	L0_valOutDemandele	valOutDemandele	-	値	値	出口	電力デマンドを値で出す

（6）記録項目

No.	項目	出力内容	単位	分類
1	DBoad_Message#-#-	メッセージ	—	メッセージ
2	DBoad_入口有効電力#W#-	入口有効電力	W	入口
3	DBoad_入口無効電力#Var#-	入口無効電力	Var	入口
4	DBoad_発電有効電力合計#W#-	発電有効電力の合計値	W	My
5	DBoad_余剰電力#W#-	余剰電力	W	My
6	DBoad_LO_eleOut[*]有効電力#W#-	各出口の有効電力	W	出口
7	DBoad_LO_eleOut[*]無効電力#Var#-	各出口の無効電力	Var	出口
8	DBoad_LO_eleInGen[*]有効電力#W#-	各発電入口の有効電力	W	入口
9	DBoad_LO_eleInGen[*]無効電力#Var#-	各発電入口の無効電力	Var	入口

（７）計算フロー・計算内容

・swcln による動作

swcln が停止信号の時

電力は流れないとして次の処理を行う。

eleIn および各 eleOut[]の有効電力、無効電力、電流を＝０とする。

swcln が運転信号の時

・出口電力の有効電力と無効電力を集計する。

出口有効電力合計＝Σ eleOut[i]有効電力

出口無効電力合計＝Σ eleOut[i]無効電力

・拡大倍率の適用

電力デマンド有効電力＝出口有効電力合計×有効無効電力拡大倍率

電力デマンド無効電力＝出口無効電力合計×有効無効電力拡大倍率

・発電電力の有効電力と無効電力の集計

発電有効電力合計＝Σ eleInGen[i]有効電力

発電無効電力合計＝Σ eleInGen[i]無効電力

・入口電力の算定

入口有効電力＝出口有効電力合計―発電有効電力合計

入口無効電力＝出口無効電力合計―発電無効電力合計

入口有効電力<0 の時、余剰電力＝－入口有効電力が発生している。

入口有効電力<0 の時で逆潮流しない場合は、入口有効電力＝０とする。

入口無効電力<0 の時、入口無効電力＝０とする。

（８）データ範囲と範囲外の取扱い

特になし。

参考　ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.Map;

import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * @author HIROSHI NINOMIYA /20080909
 * 分電盤です
 *   入口商用１系統　発電n系統 /20120202
 *   出口ｎ系統
 *   出口側の電力を積算し入口側へ伝えます
 *   商用系統は発電系統分を差し引いた負荷とします
 * 記録の有効無効判断を追加
 * 20090101 グラフ表示ステップ数追加 デマンドノード追加
 */
public class DistributionBoad_nOut1InnInGenModule20120202 extends AbstractBestModule implements
      IBestMetaModule {

   //
   private final String moduleName = "(DistributionBoad_nOut1InnInGenModule20120202) ";

   //接続ノード
   private final String S_NODE_eleIn   = "L0_eleIn";

   private final String S_NODE_valOutDemandele = "L0_valOutDemandele";
   //
   //出口　接続名称は　L0_eleOut[0],L0_eleOut[1]・・・・とする
   //
   private String[] S_NODE_eleOut;
   private String[] S_NODE_eleInGen;

   private final String C_NODE_swcIn = "L1_swcIn";
   //private final String C_NODE_modIn = "L1_modIn";

   private final String R_NODE = "L2_recOut";

   //仕様
   private final String SPEC_name = "名称";
   private final String SPEC_NumberOfOutlet   = "[]出口接続ノード数";
   private final String SPEC_NumberOfInletGen = "[]発電入口接続ノード数";
   private final String SPEC_eleIn_maxActivePower = "入口最大有効電力[W]";

   private final String SPEC_enlarge = "有効無効電力拡大倍率";

   private final String SPEC_isBackwardFlow = "is逆潮流する";//このグラフを表示する
   //
   private final String SPEC_isGVisible = "グラフを表示する";//このグラフを表示する
   private final String SPEC_maxItemCount = "最大同時表示ステップ数";//最大同時表示ステップ数
   private final String SPEC_isRecord = "記録を有効とする";//このモジュールの記録を有効とする


   //記録
   private final String RECORD_message = "DBoad_Message#-#-";
   private final String RECORD_A_eleIn = "DBoad_入口有効電力#W#-";
   private final String RECORD_R_eleIn = "DBoad_入口無効電力#Var#-";
   private final String RECORD_sumGen  = "DBoad_発電有効電力合計#W#-";
   private final String RECORD_surplus = "DBoad_余剰電力#W#-";
```

```
    //接続熱媒など
    private BestElectricity[] eleOut = null;
    private BestElectricity   eleIn = null;
    private BestElectricity[] eleInGen= null;

    private BestValue valOutDemandele = null;
// private BestElectricity   eleMy = null;

    //制御信号など
    private int swcIn;                //
    //private int modIn;              //運転モード

    //仕様など
    private String name;          //名称
    private int numberOfOutlet;         //出口  接続数
    private int numberOfInletGen;       //入口  接続数

    private double eleIn_maxActivePower;      //入口  最大有効電量[W]
    private double enlarge;//= "有効無効電力拡大倍率";

    private boolean isBackwardFlow = false;//逆潮流する=true
    //
    private boolean isGVisible = false;//このグラフを表示する=true
    private int  maxItemCount = 100;//最大同時表示ステップ数
    private boolean isRecord   = false;//記録を有効とする=true

    private StringBuffer message = new StringBuffer();


    //グラフ表示など
    private GraphJFrameD_BoadG320081212 gD_Boad = null;

    //
    private double sumGen;//発電電力合計[W]
    private double surplus;//余剰電力[W]

    private boolean isCalcEleOut = false;
    private boolean isCalEleInGen = false;

    @Override
    public void setProfile(BestSpecs spec) {
       if(spec == null) {
          return;
       }

       Map<String, String> map = spec.getSpec();
       if(map == null) {
          return;
       }

       //名称を取得
       this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

       //numberOfOutletを取得
       if (null != map.get(this.SPEC_NumberOfOutlet)) {
          this.numberOfOutlet
          = Integer.parseInt( (String)map.get( this.SPEC_NumberOfOutlet ));
          //出口ｎ系統のBestWaterの初期化
          if( this.numberOfOutlet > 0 ){
             this.isCalcEleOut = true;
             this.eleOut = new BestElectricity[this.numberOfOutlet];
             this.S_NODE_eleOut = new String[this.numberOfOutlet];
             for( int i=0; i<this.eleOut.length; i++ ){
                //this.eleOut[i] = new BestElectricity();
                //this.eleOut[i] = new BestElectricity( 0, 0, 200, 0, 3, 50);
                this.S_NODE_eleOut[i] = new String( "LO_eleOut[" + i + "]" );
             }
          }
       }

       //numberOfInletGenを取得
```

```
    if (null != map.get(this.SPEC_NumberOfInletGen)) {
       this.numberOfInletGen
       = Integer.parseInt( (String)map.get( this.SPEC_NumberOfInletGen ));
       //出口ｎ系統のBestWaterの初期化
       if( this.numberOfInletGen > 0 ){
          this.isCalEleInGen = true;
          this.eleInGen = new BestElectricity[this.numberOfInletGen];
          this.S_NODE_eleInGen = new String[this.numberOfInletGen];
          for( int i=0; i<this.eleInGen.length; i++ ){
             //this.eleOut[i] = new BestElectricity();
             //this.eleOut[i] = new BestElectricity( 0, 0, 200, 0, 3, 50);
             this.S_NODE_eleInGen[i] = new String( "LO_eleInGen[" + i + "]" );
          }
       }
    }

    //SPEC_eleIn_maxActivePowerを取得
    this.eleIn_maxActivePower = spec.getSpecValue( this.SPEC_eleIn_maxActivePower, 0. );

    //SPEC_enlarge = "有効無効電力拡大倍率";
    this.enlarge = spec.getSpecValue( this.SPEC_enlarge, 1. );

    //isBackwardFlowを取得
    this.isBackwardFlow = spec.getSpecValue( this.SPEC_isBackwardFlow, false );

    //isGVisibleを取得
    this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

    // 最大同時表示ステップ数
    this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

    //isRecordを取得
    this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

 }

 public void initialize(IBestStateMessage stateNodes,
       IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
       IBestDomainSchedule schedule) {
    //状態ノードを取得
    super.sm = stateNodes;

    //制御ノードを取得
    super.cm = commandNodes;

    //記録ノードを取得
    super.rm = recordNodes;

    //接続ノード　出口
    if( this.isCalcEleOut ){
       for( int i=0; i<this.eleOut.length; i++ ){
          this.eleOut[i] = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleOut[i] );
       }
    }

    //接続ノード　入口
    //eleIn = new BestElectricity();
    this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );

    //発電入口
    if( this.isCalEleInGen ){
       for( int i=0; i<this.eleInGen.length; i++ ){
          this.eleInGen[i] = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleInGen[i] );
       }
    }

    //valOutDemandele;
    this.valOutDemandele = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valOutDemandele );


    //グラフ表示の準備
    if( this.isGVisible ){
```

```
        this.gD_Boad = new GraphJFrameD_BoadG320081212( this.name, this.maxItemCount, this.eleOut.length,
this.numberOfInletGen, this.eleIn_maxActivePower * 1.5, this.eleIn_maxActivePower / this.enlarge / this.eleOut.length );
      }
   }

   double activePower;
   double reactivePower;

   public void outputs() {
      //if( super.sm == null || super.cm == null ) {
      // return;
      //}
      //接続　入口状態値の取得　（電力は出口）
      if( this.isCalcEleOut ){
         for( int i=0; i<this.eleOut.length; i++ ){
            this.eleOut[i]
            = (BestElectricity) super.sm.getState( super.getConnectionNode( this.S_NODE_eleOut[i] ));
         }
      }

      //発電系の入口は読み込む
      if( this.isCalEleInGen ){
         for( int i=0; i<this.eleInGen.length; i++ ){
            this.eleInGen[i]
            = (BestElectricity) super.sm.getState( super.getConnectionNode( this.S_NODE_eleInGen[i] ));
         }
      }

      this.swcIn = super.cm.getCommand(super.getConnectionNode( this.C_NODE_swcIn ));


      activePower = 0.;
      reactivePower = 0.;

      this.sumGen = 0;
      this.surplus = 0;

      if( this.isCalcEleOut ){
         this.eleIn.copyAllState( this.eleOut[0] );      //eleOut[]の最初の接続eleを採用
      }

      //if(this.swcIn == 0 ){    //利用不可　遮断時
      if( Airswc.isOFF( this.swcIn ) ){      //利用不可　遮断時
         if( this.isRecord ){
            this.message.append( "(C)遮断中");
         }
         this.eleIn.setActivePower( 0. );
         this.eleIn.setReactivePower( 0. );
         this.eleIn.setCurrent( 0. );

         if( this.isCalcEleOut ){
            for( int i = 0; i < this.eleOut.length; i++ ){
               this.eleIn.copyAllState( this.eleOut[0] );      //eleOut[]の最初の接続eleを採用

                this.eleOut[i].setActivePower( 0. );
                this.eleOut[i].setReactivePower( 0. );
                this.eleOut[i].setCurrent( 0. );
            }
         }

      }else{   //利用可能　通電時
         if( this.isRecord ){
            this.message.append( "(C)通電中");
         }
         // 集計
         //入口のactivePower,reactivePower
         //System.out.println( "eleOut length=" + this.eleOut.length );

         if( this.isCalcEleOut ){
            for( int i = 0; i < this.eleOut.length; i++ ){
               //this.eleIn.copyAllState( this.eleOut[0] );    //eleOut[]の最初の接続eleを採用
               //System.out.println( "eleOut i=" + i);
```

```
            //System.out.println( "eleOut i=" + i +" "+this.name);
            activePower += this.eleOut[i].getActivePower();
            reactivePower += this.eleOut[i].getReactivePower();
        }
    }

    //拡大
    activePower *= this.enlarge;
    reactivePower *= this.enlarge;

    this.valOutDemandele.setValue( activePower );//負荷の合計に倍率を掛けた値をデマンドとする

    //System.out.println( "DB valOutDemandele = "+ this.valOutDemandele.getValue() );

    if( this.isCalEleInGen ){
        for( int i=0; i<this.eleInGen.length; i++ ){
            this.sumGen  += this.eleInGen[i].getActivePower();
            reactivePower -= this.eleInGen[i].getReactivePower();
        }
    }

    activePower -= this.sumGen;


    if( activePower < 0 ){
        if( this.isRecord ){
            this.message.append("(C)発電余剰有り" + activePower );
        }
        this.surplus = -activePower;
        if( !this.isBackwardFlow ){
            //逆潮流しない
            activePower = 0;
            reactivePower = 0;
        }
    }
    if( reactivePower < 0 ){
        reactivePower = 0;
    }
    //200901010
    if( this.eleIn_maxActivePower < activePower ){
        if( this.isRecord ){
            this.message.append( "(W)最大電力超過");
        }
    }

    this.eleIn.setActivePower( activePower );
    this.eleIn.setReactivePower( reactivePower );

}


//ノードに設定
super.sm.setState(super.getConnectionNode( this.S_NODE_eleIn ),
        this.eleIn );
super.sm.setState(super.getConnectionNode( this.S_NODE_valOutDemandele ),
        this.valOutDemandele );

//記録
//グラフデータ追加
if( this.isGVisible ){
    gD_Boad.addData( BestTimeManager.getDateWeatherTime(),
            this.swcIn,
            this.eleOut,
            this.eleIn_maxActivePower,
            this.eleIn,
            this.eleInGen );
}

//記録
if( this.isRecord  && super.rm != null ){
    this.record();
}
```

```
    message.setLength(0);
}

/**
 * 記録
 */
private void record(){

    if( CheckPrintModule.isPrintMessage ){
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_message, this.name, this.message );
    }

    //if( CheckPrintModule.isPrintEnergy ){
    //}

    //if( CheckPrintModule.isPrintLoad ){
    //}

    if( CheckPrintModule.isPrintStateOut ){
        //出口
        if( this.isCalcEleOut ){
            for( int i=0; i<eleOut.length; i++ )
            {
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        "DBoad_" + this.S_NODE_eleOut[i] + "有効電力#W#電力",this.name,
                        AirFormat.df_2( this.eleOut[i].getActivePower() ));
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        "DBoad_" + this.S_NODE_eleOut[i] + "無効電力#Var#無効電力",this.name,
                        AirFormat.df_2( this.eleOut[i].getReactivePower() ));
            }
        }
    }

    if( CheckPrintModule.isPrintStateMy ){
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_sumGen, this.name, AirFormat.df_2( this.sumGen ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_surplus, this.name, AirFormat.df_2( this.surplus ));
    }

    if( CheckPrintModule.isPrintStateIn ){
        //入口
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_A_eleIn, this.name,
                AirFormat.df_2( this.eleIn.getActivePower() ));
        super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                this.RECORD_R_eleIn, this.name,
                AirFormat.df_2( this.eleIn.getReactivePower() ));
        //発電入口
        if( this.isCalEleInGen ){
            for( int i=0; i<eleInGen.length; i++ )
            {
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        "DBoad_" + this.S_NODE_eleInGen[i] + "有効電力#W#電力",this.name,
                        AirFormat.df_2( this.eleInGen[i].getActivePower() ));
                super.rm.setRecord( super.getConnectionNode( this.R_NODE ),
                        "DBoad_" + this.S_NODE_eleInGen[i] + "無効電力#Var#無効電力",this.name,
                        AirFormat.df_2( this.eleInGen[i].getReactivePower() ));
            }
        }
    }
}


public void update() {

}

public Object viewInternal(TestCommand cmd) {
    // とりあえずクラス変数のリストを返します。
```

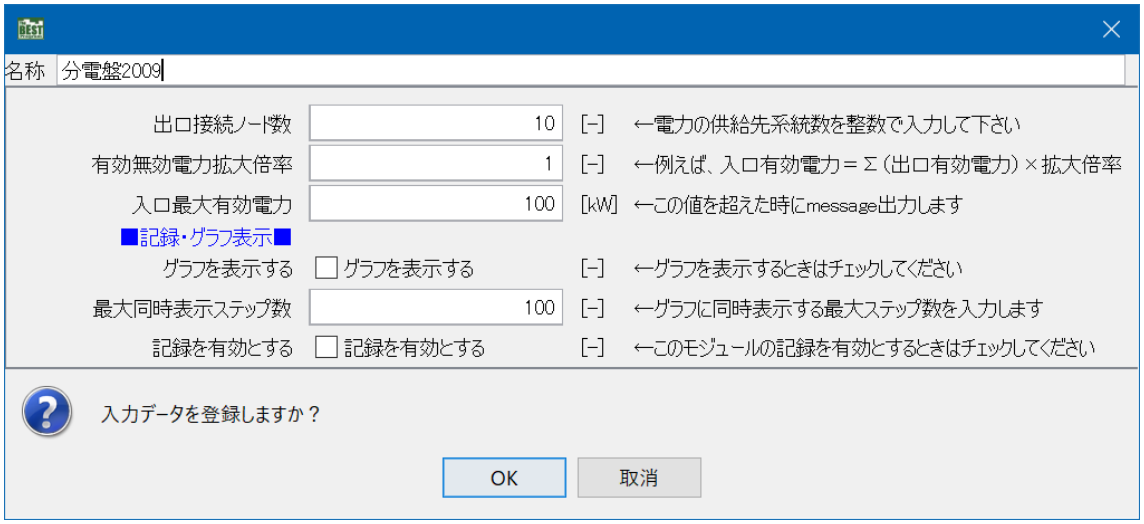
```
        java.util.List<Object> result = new java.util.ArrayList<Object>();

        //接続ノード
        result.add(super.sm);
        result.add(super.cm);

        //外部定義項目
        result.add(eleIn_maxActivePower);

        if( eleOut != null ){
           for( int i = 0; i < eleOut.length; i++ ){
              result.add( eleOut[i] );
           }
        }
//      result.add(this.eleMy);

        return result;
    }
}
```

「分電盤 2009」（場所：設備 2015／電気設備 2015／）

モジュール名	分電盤 2009
クラス	DistributionBoad_nOut1InModule20090101

（1）入力画面

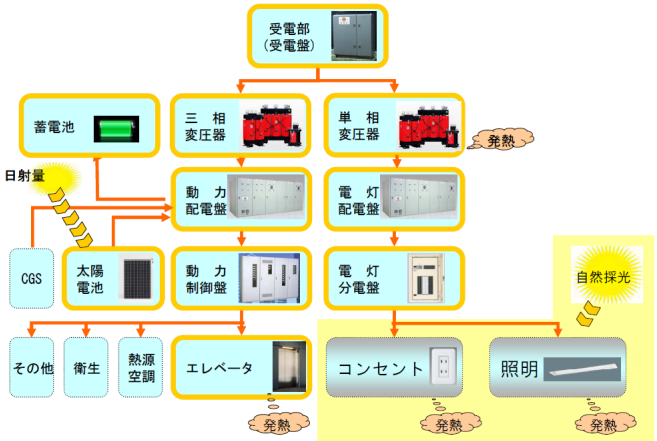
・スペック



（2）モジュールの概要

電力を分配するモジュールで、出口側に接続された電力を集計し、入口電力を設定する。

下図、電気設備の主要モジュール構成を参照。



（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

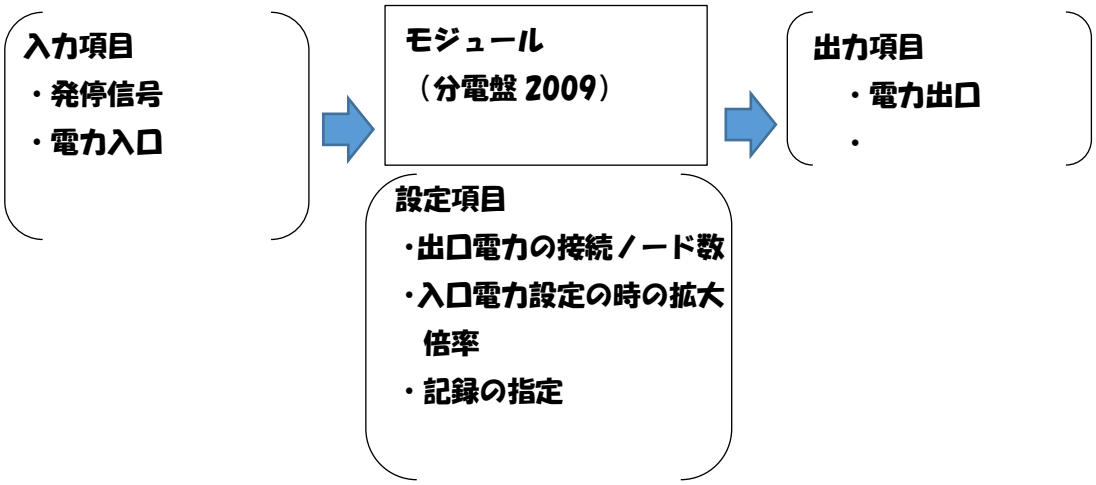


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String		―	[-]	―	―		
1	出口接続ノード数	int	numberOfOutlet	10	[-]	―	1		←電力の供給先系統数を整数で入力して下さい
2	有効無効電力拡大倍率	double	enlarge	1	[-]	―	0		←例えば、入口有効電力＝Σ（出口有効電力）×拡大倍率
3	入口最大有効電力	double	eleIn_maxActivePower	100	[kW]	―	0		←この値を超えた時に message 出力します 計算には影響しない
4	■記録・グラフ表示■					―	―		
5	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
6	最大同時表示ステップ数	int	maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
7	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェック してください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	空調記録モジュールへ接続する
2	発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	上位からの発停信号
3	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	
4	電力出口[]	L0_eleOut[]	eleOut[]	-	状態	電力	出口	スペックの出口接続ノード数で入力された数のノードが用意される
5	電力デマンド	L0_valOutDemandele	valOutDemandele	-	値	値	出口	電力デマンドを値で出す

（6）記録項目

No.	項目	出力内容	単位	分類
1	DBoad_nOut1InMessage#-#-	メッセージ	—	メッセージ
2	DBoad_nOut1In 入口有効電力#W#-	入口有効電力	W	入口
3	DBoad_nOut1In 入口無効電力#Var#-	入口無効電力	Var	入口
4	DBoad_nOut1InLO_eleOut[i]有効電力#W#-	各出口有効電力	W	出口
5	DBoad_nOut1InLO_eleOut[i]無効電力#Var#-	各出口無効電力	Var	出口

（７）計算フロー・計算内容

・swcln による動作

swcln が停止信号の時

電力は流れないとして次の処理を行う。

eleIn および各 eleOut[]の有効電力、無効電力、電流を＝０とする。

swcln が運転信号の時

・出口電力の有効電力と無効電力を集計する。

出口有効電力合計＝Σ eleOut[i]有効電力

出口無効電力合計＝Σ eleOut[i]無効電力

・拡大倍率の適用

電力デマンド有効電力＝出口有効電力合計×有効無効電力拡大倍率

電力デマンド無効電力＝出口無効電力合計×有効無効電力拡大倍率

・入口電力の算定

入口有効電力＝出口有効電力合計

入口無効電力＝出口無効電力合計

（８）データ範囲と範囲外の取扱い

特になし。

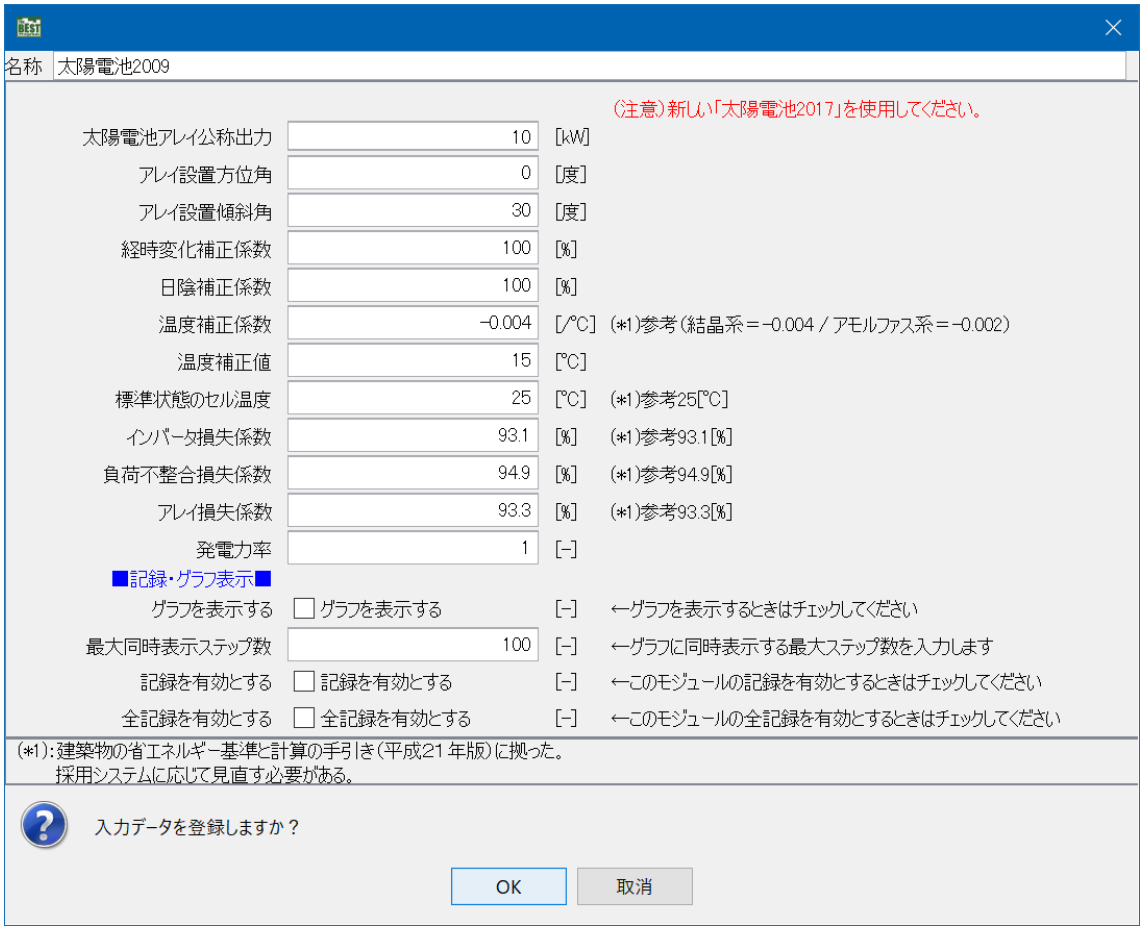
「太陽電池 2009」（場所：設備 2015／電気設備 2015／）

（注意）新しい「太陽電池 2017」を使用すること。

モジュール名	太陽電池 2009
クラス	PhotovoltaicPowerGenerationModule20090808

（1）入力画面

・スペック



（2）モジュールの概要

気象条件より、日射量、太陽位置、気温などを受け、太陽電池による発電電力量を算出するモジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

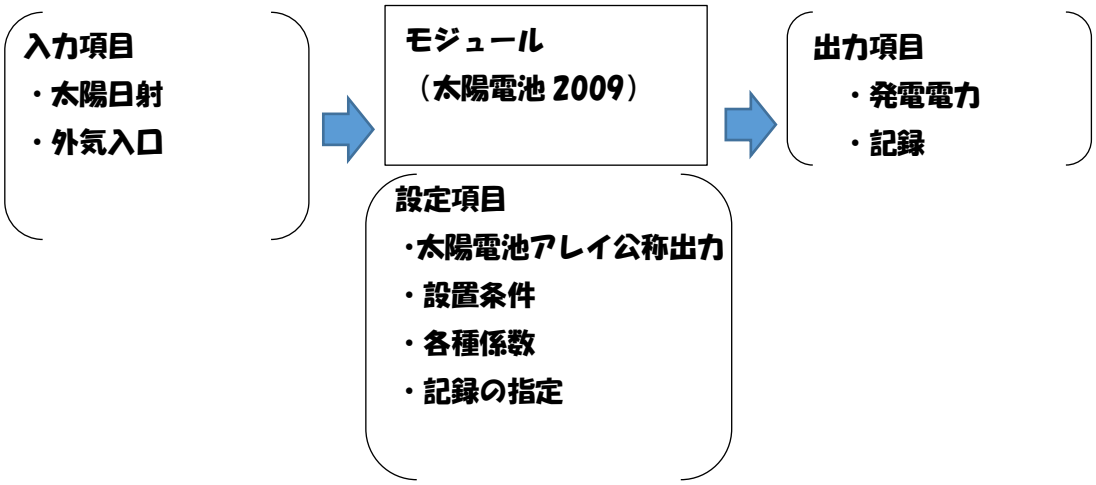


図１　モジュールの設定項目・入力項目・出力項目

（４）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	太陽電池アレイ公称出力	double	PAS	0	[kW]	―	―		
2	アレイ設置方位角	double	r	0	[度]	180	-180		南=0 度
3	アレイ設置傾斜角	double	S	30	[度]	90	0		水平=0 度
4	経時変化補正係数	double	KPD	100	[%]	100	0		
5	日陰補正係数	double	KHS	100	[%]	100	0		
6	温度補正係数	double	pmax	-0.004	[/℃]	―	―		(*1)参考（結晶系＝-0.004 / アモルファス系＝-0.002）
7	温度補正値	double	TCR	15	[℃]	―	―		
8	標準状態のセル温度	double	TS	25	[℃]	―	―		(*1) 参考 25[℃]
9	インバータ損失係数	double	CA1	93.1	[%]	100	0		(*1) 参考 93.1[%]
10	負荷不整合損失係数	double	CA2	94.9	[%]	100	0		(*1) 参考 94.9[%]
11	アレイ損失係数	double	CA3	93.3	[%]	100	0		(*1) 参考 93.3[%]
12	発電力率	double	rate	1	[-]	1	-1		
13	■記録・グラフ表示■					―	―		
14	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
15	最大同時表示ステップ数	int	maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
16	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください
17	全記録を有効とする	boolean	isRecordALL	FALSE	[-]	―	―		←このモジュールの全記録を有効とするときはチェックしてください

(*1)：建築物の省エネルギー基準と計算の手引き（平成２１年版）に拠った。採用システムの応じて見直す必要がある。

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2OrecOut	recOut	-	記録	メモリ	出口	空調記録モジュールへ接続する
2	太陽日射入口	LO_sunIn	sunIn	-	状態	太陽日射	入口	
3	外気	LO_airInOA	airInOA	-	状態	空気	入口	アレイの周囲空気
4	電力出口	LO_eleOut	eleOut	-	状態	電力	出口	発電電力を出力する
5	HA 出口	LO_valOutHA	valOutHA	-	値	値	出口	（未使用）

（6）記録項目

No.	項目	出力内容	単位	分類
1	太陽電池 Message#-#-	メッセージ	—	メッセージ
2	アレイ日射量#W/m2#-	アレイ面の単位日射量	W/m2	My
3	太陽電池発電量#W#-	太陽電池の発電量	W	My
4	法線面直達日射量#W/m2#-	法線面直達日射量	W/m2	My
5	水平面天空日射量#W/m2#-	水平面天空日射量	W/m2	My
6	sunDir#-#-	太陽方位	度	My
7	sunHeight#-#-	太陽高度	度	My
8	S#-#-	アレイ設置傾斜角	度	My
9	eleOut 有効電力#W#-	発電有効電力	W	My
10	eleOut 無効電力#VAR#-	発電無効電力	Var	My

（７）計算フロー・計算内容

・動作について

現在時刻を取得します

times = BestTimeManager.getDateWeatherTime();

現在時刻から時角を算出します

w = 15 * (times[5] - 12);

接続ノードから日射量を取得します

sunIn = (BestSun)super.sm.getState(super.getConnectionNode(this.S_NODE_sunIn));

アレイ日射量を取得します

ha = getHA(w, 0., 0.);

温度を取得します

airInOA = (BestAir)super.sm.getState(super.getConnectionNode(this.S_NODE_airInOA));

発電量を取得します

amount = getEP(airInOA.getTempDB(), ha);

eleOut へ設定する

eleOut.setActivePower(amount);

eleOut.setReactivePower(amount * Math.pow(1/rate/ rate - 1, 0.5));

valOutHA へ設定する

valOutHA.setValue(ha);

・発電量 EP の計算　getEP（）メソッド

EP = HA × KPD × (1. + pmax × (airInOA.getTempDB() + TCR - TS))
× KHS × CA1 × CA2 × CA3 × PAS / 1000.;

・日射量 HA の計算 getHA（）メソッド

HA = hdi + sunIn.getSrs() × (1. + cos(toRadians(S))) / 2.;

hdi = sunIn.getSrd() × (sin(toRadians(sunIn.getSunHeight())) ×
cos(toRadians(S)) + cos(toRadians(sunIn.getSunHeight())) × sin(toRadians(S)) ×
cos(toRadians(sunIn.getSunDir() - r)));

（８）データ範囲と範囲外の取扱い

特になし。

参考　ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestSun;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * PhotovoltaicPowerGenerationModule20090808
 * 太陽光発電Module
 *
 * @author HIROSHI NIOMIYA /20080909
 *
 * SunbeamGenelatorModule20090101を基に作成
 *
 * 20090908アレイ日射量の訂正
 */
public class PhotovoltaicPowerGenerationModule20090808 extends AbstractBestModule
            implements IBestMetaModule, jp.or.ibec.best.dk.test.IProbeForUT{

   private static final String moduleName = "PhotovoltaicPowerGenerationModule20090808";

   //接続ノード
   protected static final String S_NODE_sunIn   = "L0_sunIn";
   protected static final String S_NODE_airInOA = "L0_airInOA";
   protected static final String S_NODE_eleOut  = "L0_eleOut";
   protected static final String S_NODE_valOutHA= "L0_valOutHA";
   //
   private static final String RO_NODE = "L20recOut";
   //private static final String R1_NODE = "L21recOut";

   //外部定義
   private static final String SPEC_name  = "名称";
   private static final String SPEC_ITEM1 = "太陽電池アレイ公称出力[W]";
   //private static final String SPEC_ITEM2 = "アレイ設置場所";
   //private static final String SPEC_ITEM3 = "緯度";
   private static final String SPEC_ITEM4 = "アレイ設置方位角[度]";
   private static final String SPEC_ITEM5 = "アレイ設置傾斜角[度]";
   private static final String SPEC_ITEM6 = "経時変化補正係数[-]";
   private static final String SPEC_ITEM7 = "日陰補正係数[-]";
   private static final String SPEC_ITEM8 = "温度補正係数[/℃]";
   private static final String SPEC_ITEM9 = "温度補正値[℃]";
   private static final String SPEC_ITEMA = "標準状態のセル温度[℃]";
   private static final String SPEC_ITEMB = "インバータ損失係数[-]";
   private static final String SPEC_ITEMC = "負荷不整合損失係数[-]";
   private static final String SPEC_ITEMD = "アレイ損失係数[-]";
   private static final String SPEC_ITEME = "発電力率[-]";
   //private static final String SPEC_ID = "識別名";
   //
   private static final String SPEC_isGVisible = "グラフを表示する";
   private static final String SPEC_maxItemCount = "最大同時表示ステップ数";
   private static final String SPEC_isRecord = "記録を有効とする";
   private static final String SPEC_isRecordALL = "全記録を有効とする";
   //記録項目
   private static final String RECORD_message = "太陽電池Message#-#-";
```

```
    private static final String RECORD_HA = "アレイ日射量#W/m2#-";
    private static final String RECORD_EP = "太陽電池発電量#W#-";
    private static final String RECORD_srd = "法線面直達日射量#W/m2#-";
    private static final String RECORD_srs = "水平面天空日射量#W/m2#-";
    private static final String RECORD_sunDir = "sunDir#-#-";
    private static final String RECORD_sunHeight = "sunHeight#-#-";
    private static final String RECORD_s = "S#-#-";
    private static final String RECORD_a_eleOut = "eleOut有効電力#W#-";
    private static final String RECORD_r_eleOut = "eleOut無効電力#VAR#-";
    //インスタンス変数
    private StringBuffer message = new StringBuffer();    //メッセージ
    private String  name = null;  //機器名称
    //
// protected BestGeneratedPower gp;
    protected BestSun sunIn;
    protected BestAir airInOA;
    protected BestElectricity eleOut;
    protected BestValue valOutHA;

    private double PAS;
    //private String location;
    private double phai;
    private double r;
    private double S;
    private double KPD;
    private double KHS;
    private double pmax;
    private double TCR;
    private double TS;
    private double CA1;
    private double CA2;
    private double CA3;
    private double rate;
    private String ID;
    //
    private boolean isGVisible = false;//このグラフを表示する=true
    private int  maxItemCount = 100;//最大同時表示ステップ数
    private boolean isRecord   = false;//記録を有効とする=true
    private boolean isRecordALL= false;//全記録を有効とする=true

    private double HA;         //アレイ日射量
    private double EP;         //発電量

    //グラフ表示など
    private GraphJFramePhotovoltaicPowerGenerator20090808 gSunGenelator = null;


    @Override
    public void setProfile(BestSpecs spec) {
       if (null == spec)
          return;

       Map<String, String> map =spec.getSpec();
       if (null == map)
          return;

       //名称を取得
       this.name = spec.getSpecValue( SPEC_name, this.moduleName );

       this.PAS = spec.getSpecValue( SPEC_ITEM1, 0. );

       this.r = spec.getSpecValue( SPEC_ITEM4, 0. );

       this.S = spec.getSpecValue( SPEC_ITEM5, 30. );

       this.KPD = spec.getSpecValue( SPEC_ITEM6, 1. );

       this.KHS = spec.getSpecValue( SPEC_ITEM7, 1. );

       this.pmax = spec.getSpecValue( SPEC_ITEM8, -0.0041 );

       this.TCR = spec.getSpecValue( SPEC_ITEM9, 15. );
```

```
        this.TS = spec.getSpecValue( SPEC_ITEMA, 25. );

        this.CA1 = spec.getSpecValue( SPEC_ITEMB, 0.931 );

        this.CA2 = spec.getSpecValue( SPEC_ITEMC, 0.949 );

        this.CA3 = spec.getSpecValue( SPEC_ITEMD, 0.933 );

        this.rate = spec.getSpecValue( SPEC_ITEME, 1. );

        this.message.append( "(C)太陽電池  PAS=" + this.PAS +" S=" + this.S +" KPD=" + this.KPD +" CA1=" + this.CA1 +"
TCR=" + this.TCR );

        //isGVisibleを取得
        this.isGVisible = spec.getSpecValue( SPEC_isGVisible, false );

        // 最大同時表示ステップ数
        this.maxItemCount = spec.getSpecValue( SPEC_maxItemCount, 100 );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( SPEC_isRecord, false );
        this.isRecordALL = spec.getSpecValue( SPEC_isRecordALL, false );

    }

    public void initialize(IBestStateMessage stateNodes,
            IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
            IBestDomainSchedule schedule) {

        //状態ノードを受け取る
        super.sm = stateNodes;
        //記録ノードを受け取る
        super.rm = recordNodes;
        //スケジュールを受け取る
        super.ds = schedule;


        //sunIn
        this.sunIn = BestSun.bindnode( super.transferMapState, super.sm, S_NODE_sunIn );

        //airInOA
        this.airInOA = BestAir.bindnode( super.transferMapState, super.sm, S_NODE_airInOA );

        //eleOut
        this.eleOut = BestElectricity.bindnode( super.transferMapState, super.sm, S_NODE_eleOut );

        //valOutHA
        this.valOutHA = BestValue.bindnode( super.transferMapState, super.sm, S_NODE_valOutHA );

        //グラフ表示の準備
        if( this.isGVisible ){
            this.gSunGenelator = new GraphJFramePhotovoltaicPowerGenerator20090808( this.name, this.maxItemCount, this.PAS *
1.5, this.PAS * 1.5 );
        }

    }


    public void outputs() {
        if (null == super.sm)
            return;

        //現在時刻を取得します
        int[] times = BestTimeManager.getDateWeatherTime();

        //現在時刻から時角を算出します
        double w = 15 * (times[5] - 12);

        //接続ノードから日射量を取得します
        this.sunIn = (BestSun)super.sm.getState(super.getConnectionNode(S_NODE_sunIn));
```

```
        //アレイ日射量を取得します
        //double ha = this.getHA(w, hb, hd);
        double ha = this.getHA(w, 0., 0.);
        //温度を取得します
        this.airInOA = (BestAir)super.sm.getState(super.getConnectionNode(S_NODE_airInOA));
        //double ty = (Double)super.sm.getState(super.getConnectionNode(this.S3_NODE));
        //発電量を取得します
        double amount = this.getEP( this.airInOA.getTempDB(), ha);
        //this.electricity.setActivePower(amount);
        //this.gp.updatePower(this, this.electricity);
        this.eleOut.setActivePower(amount);
        this.eleOut.setReactivePower(
              amount * Math.pow( 1/this.rate/ this.rate - 1, 0.5 ) );
//      this.gp.updatePower(this, this.eleOut);
        //出力します
        //super.valOutHA = Double.valueOf( ha );
        this.valOutHA.setValue( ha );
        super.sm.setState(super.getConnectionNode(S_NODE_eleOut), this.eleOut);
        super.sm.setState(super.getConnectionNode(S_NODE_valOutHA), this.valOutHA);

        //super.sm.setState(super.getConnectionNode(this.SO_NODE), this.gp);
        //グラフデータ追加
        if( this.isGVisible ){
           gSunGenelator.addData( BestTimeManager.getDateWeatherTime(),
                 1,
                 this.eleOut,
                 this.PAS,
                 this.sunIn,
                 this.HA );
        }

        //記録
        if( this.isRecord  && super.rm != null ){
           this.record();
        }

        message.setLength(0);
    }

    /**
     * 記録
     */
    private void record(){
        //super.rm.setRecord(super.getConnectionNode(this.RO_NODE), this.RECORD_ITEM, this.ID, this.HA);

        //super.rm.setRecord(super.getConnectionNode(this.R1_NODE), this.DEBUG_ITEM, this.ID, this.EP);
           //記録
        if( this.isRecordALL || CheckPrintModule.isPrintMessage ){
           //message
           super.rm.setRecord(super.getConnectionNode(RO_NODE),
                 RECORD_message, this.name, message.toString() );
        }

        //if( this.isRecordALL || CheckPrintModule.isPrintEnergy ){
        //}

        //if( this.isRecordALL || CheckPrintModule.isPrintLoad ){
        //}

        //if( this.isRecordALL || CheckPrintModule.isPrintStateOut ){
        //}


        if( this.isRecordALL || CheckPrintModule.isPrintStateMy ){
              super.rm.setRecord( super.getConnectionNode(RO_NODE),
                    RECORD_HA, this.name,   this.HA );
              super.rm.setRecord( super.getConnectionNode(RO_NODE),
                    RECORD_EP, this.name,   this.EP );
              super.rm.setRecord( super.getConnectionNode(RO_NODE),
                    RECORD_srd, this.name,   this.sunIn.getSrd() );
              super.rm.setRecord( super.getConnectionNode(RO_NODE),
                    RECORD_srs, this.name,   this.sunIn.getSrs() );
```

```
            super.rm.setRecord( super.getConnectionNode(RO_NODE),
                  RECORD_sunDir, this.name,  this.sunIn.getSunDir() );
            super.rm.setRecord( super.getConnectionNode(RO_NODE),
                  RECORD_sunHeight, this.name,  this.sunIn.getSunHeight() );
            super.rm.setRecord( super.getConnectionNode(RO_NODE),
                  RECORD_s, this.name,  this.S );
            super.rm.setRecord( super.getConnectionNode(RO_NODE),
                  RECORD_a_eleOut, this.name,  this.eleOut.getActivePower() );
            super.rm.setRecord( super.getConnectionNode(RO_NODE),
                  RECORD_r_eleOut, this.name,  this.eleOut.getReactivePower() );
      }

      //if( this.isRecordALL || CheckPrintModule.isPrintStateIn ){
      //}

      message.setLength( 0 );
   }


   public void update() {

   }

   @Override
   public Object viewInternal(TestCommand cmd) {
      List<Object> result = new ArrayList<Object>();

      result.add(super.sm);
      result.add(super.rm);

      result.add(this.PAS);
      result.add(this.phai);
      result.add(this.r);
      result.add(this.S);
      result.add(this.KPD);
      result.add(this.KHS);
      result.add(this.pmax);
      result.add(this.TCR);
      result.add(this.TS);
      result.add(this.CA1);
      result.add(this.CA2);
      result.add(this.CA3);
      result.add(this.ID);

      result.add(this.HA);
      result.add(this.EP);

      return result;
   }


   protected double getEP(double ty, double ha) {
   /* System.out.println("ty=" + ty);
      double KPT = (1 + this.pmax * (ty + this.TCR - this.TS)) * 100;
      System.out.println("KPT=" + KPT);
      double KO = (this.CA1 / 100) * (this.CA2 / 100) * (this.CA3 / 100) * 100;
      System.out.println("KO=" + KO);
      double K = this.KPD / 100 * KPT / 100 * this.KHS / 100 * KO / 100;
      System.out.println("K=" + K);
      this.EP = ha * K * this.PAS * 1000;
   */
      //200808
      //発電量EP＝HA×K×PAS
      //ただし、補正係数K＝KPD×温度補正係数KPT×KHS×システム損失係数Ko
      //          温度補正係数KPT＝ （1＋αPmax×(Best_sun.To＋TCR-TS) )
      //           システム損失係数Ko＝CA1×CA2×CA3
      this.EP = this.HA
               * this.KPD * ( 1. + this.pmax * ( this.airInOA.getTempDB() + this.TCR - this.TS ))
               * this.KHS * this.CA1 * this.CA2 * this.CA3
               * this.PAS /1000.;

      //System.out.println("EP=" + this.EP);
```

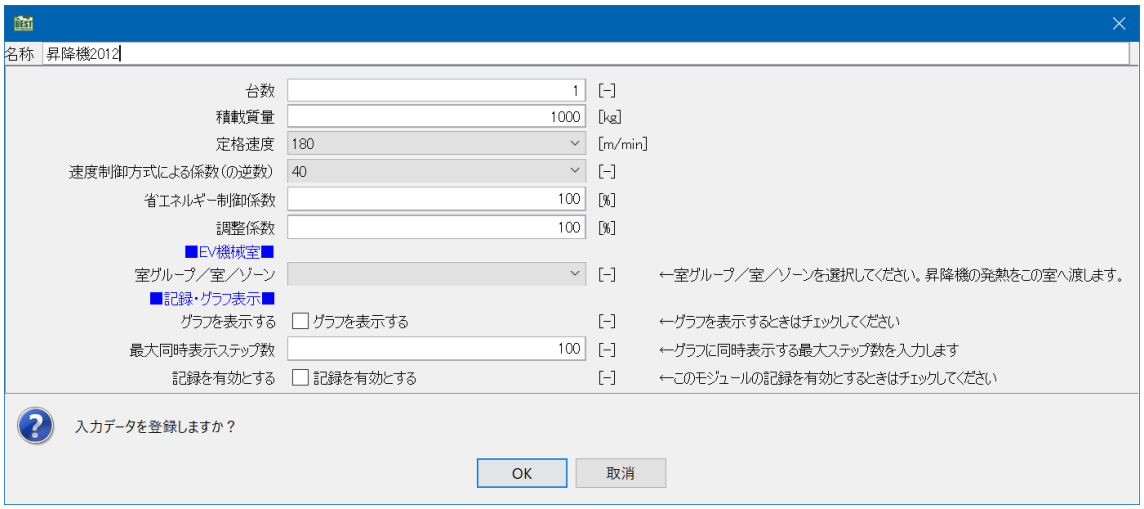
```
        return this.EP;
    }

    protected double getHA(double w, double hb, double hd) {
/*      System.out.println("w=" + w + ";hb=" + hb + ";hd=" + hd);
        double sinD = Math.sin(0.0);
        double sinP = Math.sin(Math.toRadians(this.phai));
        double cosS = Math.cos(Math.toRadians(this.S));
        double cosP = Math.cos(Math.toRadians(this.phai));
        double sinS = Math.sin(Math.toRadians(this.S));
        double cosR = Math.cos(Math.toRadians(this.r));
        double cosD = Math.cos(0.0);
        double cosW = Math.cos(Math.toRadians(w));
        double sinR = Math.sin(Math.toRadians(this.r));
        double sinW = Math.sin(Math.toRadians(w));
        double theta = sinD * sinP * cosS
                    + sinD * cosP * sinS * cosR
                    + cosD * cosP * cosS * cosW
                    - cosD * sinP * sinS * cosR * cosW
                    + cosD * sinS * sinR * sinW;
        double HDI = hb * theta;
        System.out.println("theta=" + theta);
        System.out.println("HDI=" + HDI);
        double HSI = hd * (1 + cosS) / 2;
        System.out.println("HSI=" + HSI);
        this.HA = HDI + HSI;
 */
        //200808
        //アレイ日射量HA＝傾斜直達日射量HDI＋散乱日射量HSI
        //ただし、傾斜直達日射量HDI＝Best_sun.srd×sin(Best_sun.sunHeight+s)×cos(Best_sun.sunDir-γ)
        //              散乱日射量HSI＝Best_sun.srs×（1+cosS）/2

        //20090629
        //this.HA
        //= this.sunIn.getSrd()
        //* Math.sin( Math.toRadians( this.sunIn.getSunHeight() + this.S ))
        //* Math.cos( Math.toRadians(this.sunIn.getSunDir() - this.r ))
        //+ this.sunIn.getSrs() * ( 1. + Math.cos( Math.toRadians(this.S) )) / 2.;
        double hdi;

        //20090908
        //if( Math.cos( Math.toRadians( this.sunIn.getSunDir() - this.r )) <= 0 ){
        // hdi = 0;
        //}else{
           //hdi = this.sunIn.getSrd()
           //* Math.sin( Math.toRadians( this.sunIn.getSunHeight() + this.S ))
           //* Math.cos( Math.toRadians(this.sunIn.getSunDir() - this.r ));
           hdi = this.sunIn.getSrd()
           * ( Math.sin( Math.toRadians( this.sunIn.getSunHeight() ) ) * Math.cos( Math.toRadians( this.S ))
                 +  Math.cos( Math.toRadians( this.sunIn.getSunHeight() ))
                 * Math.sin( Math.toRadians( this.S ))
                 * Math.cos( Math.toRadians( this.sunIn.getSunDir() - this.r )) );
        //}

        if( hdi < 0 ){
           hdi = 0;
        }

        this.HA = hdi
              + this.sunIn.getSrs() * ( 1. + Math.cos( Math.toRadians(this.S) )) / 2.;



        this.message.append( " HA=" + this.HA + " sin="
              + Math.sin( Math.toRadians( this.sunIn.getSunHeight() + this.S ) )
              +" cos="+Math.cos( Math.toRadians(this.sunIn.getSunDir() - this.r ))
              +" Srs="+this.sunIn.getSrs() * ( 1. + Math.cos( Math.toRadians(this.S ))) / 2.);
        //
        return this.HA;
    }

}
```

「昇降機 2012」（場所：設備 2015／電気設備 2015／）

モジュール名	昇降機 2012
クラス	EVModule20120303

（1）入力画面

・スペック



（2）モジュールの概要

積載質量、定格速度などにより、エレベータの消費電力量を算出するモジュールである。

CEC/EV の計算法をベースに、負荷パターンの入力と、各種省エネルギー手法の採用を考慮した補正を可能としている。

昇降機の計算は、昇降機 1 台ごとの入力部分（昇降機モジュール：本モジュール）と運転パターン（昇降機境界条件指定モジュール）にて行う。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String		―	[-]	―	―		
1	台数	double	N	1	[-]	―	0		
2	積載質量	double	L	1000	[kg]	―	―		
3	定格速度	double	V	60、90、105、120、150、180、210、240、300、360、420、480、540、600	[m/min]	―	―		
4	速度制御方式による係数（の逆数）	double	FT	45、40	[-]	100	0		
5	省エネルギー制御係数	double	FE	100	[%]	100	0		
6	調整係数	double	C	100	[%]	100	0		
7	■EV 機械室■					―	―		
8	室グループ／室／ゾーン	String		#RoomGroup	[-]	―	―		←室グループ／室／ゾーンを選択してください。昇降機の発熱をこの室へ渡します。
9	■記録・グラフ表示■					―	―		
10	グラフを表示する	boolean	isGVisible	FALSE	[-]	―	―		←グラフを表示するときはチェックしてください
11	最大同時表示ステップ数	int	maxItemCount	100	[-]	―	―		←グラフに同時表示する最大ステップ数を入力します
12	記録を有効とする	boolean	isRecord	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	空調記録モジュールへ接続する
2	発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	上位からの発停信号
3	モード信号	L1_modIn	modIn	-	制御	制御モード	入口	上位からのモード信号
4	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	（消費電力）
5	制御操作入口	L0_valInOperate	valInOperate	-	値	値	入口	EV 運転条件モジュールの L0_valOutSch と接続する

（6）記録項目

No.	項目	出力内容	単位	分類
1	EV_Message	メッセージ	—	メッセージ
2	EV_消費電力#W#その他昇降機電力	昇降機の消費電力	W	エネルギー
3	EV_発熱#W#-	昇降機設備の発熱	W	My
4	EV_ゾーン熱負荷#W#-	設置室への発熱	W	負荷
5	EV_ゾーン熱負荷密度#W/m2#-	設置室への発熱密度	W/m2	負荷

（７）計算フロー・計算内容

・標準 WV 消費電力　EEVn

EEVn =（L / 1000. × V × 60. / FT / 860）×（FE × C）× 1000.;

N;//台数[-]

L;//積載質量[g]

V;//定格速度[m/s]

FT;//制御方式による係数の逆数[-]

FE;//省エネルギー制御係数[-]

C;//調整係数[0-1]

・swcln による動作

swcln が停止の時

昇降機は停止とし、消費電力＝０とする。

swcln が運転の時

消費電力＝EEV n ×制御操作量（運転パターンから）×台数

発熱　＝　消費電力（100%）

（８）データ範囲と範囲外の取扱い

特になし。

参考　ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.building.load.SystemHeatGain;
import jp.or.ibec.best.domain.building.spaces.ISpace;
import jp.or.ibec.best.domain.electric.IBestElectricDomainService;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * @author
 * 20081212/HIROSHI NIOMIYA
 *
 * 20090101 グラフ表示ステップ数追加
 *
 * swc  mod 追加
 */
public class EVModule20120303 extends AbstractBestModule
                                          implements IBestMetaModule, jp.or.ibec.best.dk.test.IProbeForUT {

            private final String moduleName = "(EVModule20120303) ";

            protected IBestElectricDomainService service;
            protected String data;

            private final String C_NODE_swcIn = "L1_swcIn" ;            //運転状態：off/on
            private final String C_NODE_modIn = "L1_modIn"; //空調モード：停止/冷却/加熱

            //接続ノード
            protected final String S_NODE_eleIn        = "L0_eleIn";
            protected final String S_NODE_valInOperate = "L0_valInOperate";//運転率

            private final String R1_NODE = "L2_recOut";
            //外部定義
            private final String SPEC_name  = "名称";
            //
            private final String SPEC_NUM = "台数[-]";
            private final String SPEC_ITEM1 = "積載質量[g]";//[g]
            private final String SPEC_ITEM2 = "定格速度[m/s]";//[m/s]
            private final String SPEC_ITEM3 = "速度制御方式による係数（の逆数）[-]";//[-]
            private final String SPEC_ITEM4 = "省エネルギー制御係数[-]";//[0-1]
            private final String SPEC_ITEM6 = "調整係数[-]";//[0-1]
            //
            private final String SPEC_grzName = "室グループ／室／ゾーン";
            //
            private final String SPEC_isGVisible   = "グラフを表示する";
            private final String SPEC_maxItemCount = "最大同時表示ステップ数";
            private final String SPEC_isRecord     = "記録を有効とする";
            //記録項目
            private final String RECORD_PPE        = "EV_消費電力#W#その他昇降機電力";
            private final String RECORD_HeatGain   = "EV_発熱#W#-";
            private final String RECORD_HeatGainZ  = "EV_ゾーン熱負荷#W#-";
            private final String RECORD_HeatGainZA = "EV_ゾーン熱負荷密度#W/m2#-";
            private final String RECORD_message    = "EV_Message" ;

            //インスタンス変数
```

```
private StringBuffer message = new StringBuffer();          //message
private String name;                                        //名称
private double N;//台数[-]
private double L;//積載質量[g]
private double V;//定格速度[m/s]
private double FT;//制御方式による係数の逆数[-]
private double FE;//省エネルギー制御係数[-]
private double C;//調整係数[0-1]

private int phase = 3;                          //相数[-]
private double voltage = 200;                   //電圧[V]
private double frequency = 50;                  //周波数[Hz]
private double powerFactor = 0.8;   //力率[-]
//
private boolean isGVisible = false;//このグラフを表示する=true
private int  maxItemCount = 100;//最大同時表示ステップ数
private boolean isRecord   = false;//記録を有効とする=true

private double EEVn;                            //標準エレベータ消費電力量[W]
private BestElectricity eleIn;      //消費電力
private BestValue valInOperate = null;  //

private double heatGain = 0;

private ISpace coreSpace   ;//建物側インスタンス
private double zoneArea;//室の面積
private SystemHeatGain systemHeatGain;          //建物側インスタンス
private boolean isSystemHeatGain = false;

private int modIn  ;
private int swcIn ;

//グラフ表示など
private GraphJFrameEV20080909 gEV = null;



@Override
public void setProfile(BestSpecs spec) {
            if (null == spec)
                        return;

            Map<String, String> map =spec.getSpec();
            if (null == map)
                        return;

            //名称を取得
            this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

            this.N = spec.getSpecValue( this.SPEC_NUM, 0. );

            this.L = spec.getSpecValue( this.SPEC_ITEM1, 0. );
            //System.out.println( "EV L="+ this.L );

            this.V = spec.getSpecValue( this.SPEC_ITEM2, 0. );
            //System.out.println( "EV V="+ this.V );

            this.FT = spec.getSpecValue( this.SPEC_ITEM3, 40. );
            //System.out.println( "EV FT="+ this.FT );

            this.FE = spec.getSpecValue( this.SPEC_ITEM4, 1. );
            //System.out.println( "EV FE="+ this.FE );

            this.C = spec.getSpecValue( this.SPEC_ITEM6, 1. );

            //室グループ、ゾーン名の取得
            String grzName = null;
            grzName = spec.getSpecValue( this.SPEC_grzName, "" );
            if( grzName.equals( "" ) ){
                        this.isSystemHeatGain = false;
            }else{
                        this.isSystemHeatGain = true;//systemHeatGain で発熱をゾーンへ渡す
```

```
                        //SystemHeatGainインスタンスを建物側に登録
                        this.coreSpace = ZoneGRZName.getCoreSpace( grzName, this.name );
                        this.zoneArea = this.coreSpace.getValue( ISpace.FLOORAREA );
                        this.systemHeatGain = new SystemHeatGain();
                        this.systemHeatGain.setProfile(
                                        ZoneGRZName.getMultiSpaceName( grzName, this.name ),
                                        ZoneGRZName.getZoneName( grzName, this.name ) );
                }

                //isGVisibleを取得
                this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

                // 最大同時表示ステップ数
                this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

                //isRecordを取得
                this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

        }

        @Override
        public void initialize(IBestStateMessage stateNodes,
                        IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
                        IBestDomainSchedule schedule) {

                //状態ノードを受け取る
                super.sm = stateNodes;
                //制御ノードを取得
                super.cm = commandNodes;
                //記録ノードを受け取る
                super.rm = recordNodes;
                //スケジュールを受け取る
                super.ds = schedule;


                //接続ノード
                //入口
                this.eleIn = BestElectricity.bindnode( super.transferMapState, super.sm, this.S_NODE_eleIn );
                this.eleIn.setPhase( this.phase );
                this.eleIn.setFrequency( this.frequency );
                this.eleIn.setVoltage( this.voltage );


                //出口
                this.valInOperate = BestValue.bindnode( super.transferMapState, super.sm,
this.S_NODE_valInOperate );


                this.EEVn = ( L/1000. * V*60. / FT / 860) * (FE * C) * 1000.;


                //グラフ表示の準備
                if( this.isGVisible ){
                        this.gEV = new GraphJFrameEV20080909( this.name, this.maxItemCount, this.EEVn * 1.5,
0 );
                }
        }

        public void outputs() {

                if (null == super.sm || super.cm == null )
                        return;

                this.swcIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE_swcIn) );
                this.modIn = super.cm.getCommand(super.getConnectionNode(this.C_NODE_modIn));

                double activePower;
                double reactivePower;

                activePower = 0.;
                reactivePower = 0.;
```

```
                        if( Airswc.isOFF( this.swcIn )){
                                    if( this.isRecord ){
                                                message.append("(C)停止");
                                    }
                                    cal_Stop();
                        }else{
                                    if( this.isRecord ){
                                                message.append("(C)運転");
                                    }

                                    //運転パターン係数を取得
                                    this.valInOperate =
(BestValue)super.sm.getState(super.getConnectionNode(this.S_NODE_valInOperate));
                                    //System.out.println("運転率：" + this.valInOperate );

                                    activePower = this.EEVn * this.valInOperate.getValue() * N;//台数N倍
                                    reactivePower = activePower * Math.pow( 1/this.powerFactor/ this.powerFactor - 1,
0.5 );
                        }

                        this.eleIn.setActivePower( activePower );
                        this.eleIn.setReactivePower( reactivePower );

                        //
                        this.heatGain = activePower;//今は、消費電力100%が発熱とする

                        //発熱の受渡し方法
                        if( this.isSystemHeatGain ){
                                    //建物側へ熱量[W]を渡す　すべて対流
                                    this.systemHeatGain.integrateHeatGain(
                                                            this.heatGain, 0, 0, BestTimeManager.getCurrentInterval() );
                                    //接続媒体で渡す＝０
                                    //this.heaOut.setConvectiveHeatRate( 0. );
                        }else{
                                    //建物側へ熱量[W]を渡す　すべて対流＝０
                                    //this.systemHeatGain.integrateHeatGain(
                                    //                        0, 0, 0, BestTimeManager.getCurrentInterval() );
                                    //接続媒体で渡す
                                    //this.heaOut.setConvectiveHeatRate( this.heatGain );
                        }

                        //ノードに書き出します
                        super.sm.setState(super.getConnectionNode(this.S_NODE_eleIn), this.eleIn);

                        //グラフデータ追加
                        if( this.isGVisible ){
                                    gEV.addData( BestTimeManager.getDateWeatherTime(),
                                                            1,
                                                            this.EEVn,
                                                            this.eleIn,
                                                            this.heatGain );
                        }

                        //記録
                        if( this.isRecord  && super.rm != null ){
                                    this.record();
                        }

                        message.setLength(0);
            }

            private void cal_Stop(){

            }


            /**
             * 記録
             */
            private void record(){

                        if( CheckPrintModule.isPrintMessage ){
```

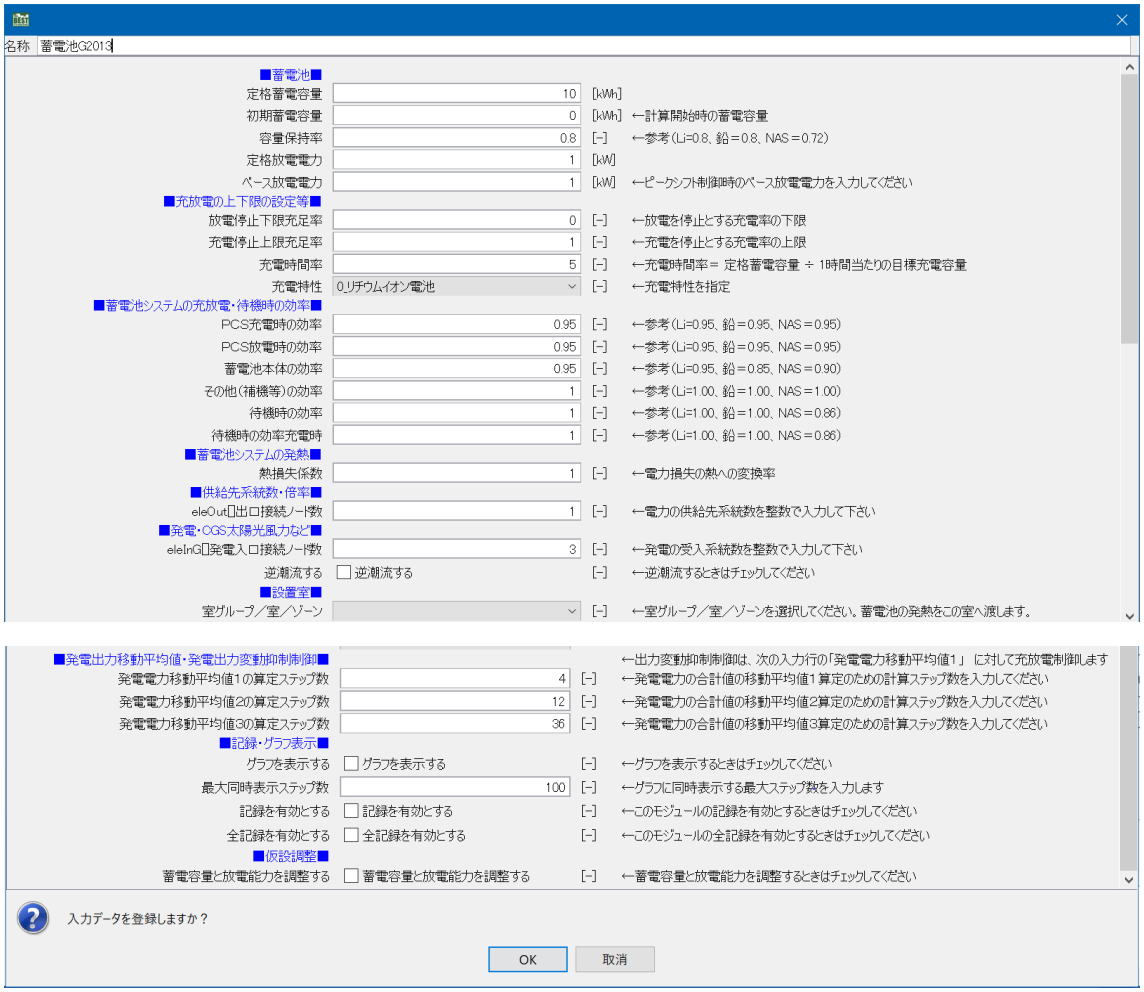
```
                                        super.rm.setRecord(super.getConnectionNode(this.R1_NODE),
                                                        this.RECORD_message, this.name, message.toString() );
                                }

                                //if( CheckPrintModule.isPrintEnergy ){
                                //}

                                if( CheckPrintModule.isPrintLoad ){
                                        if( this.isSystemHeatGain ){
                                                super.rm.setRecord(super.getConnectionNode(this.R1_NODE),
                                                                this.RECORD_HeatGainZ, this.name,
AirFormat.df_2( this.heatGain ));
                                                if( this.zoneArea != 0 ){
                                                        super.rm.setRecord(super.getConnectionNode(this.R1_NODE),
                                                                        this.RECORD_HeatGainZA, this.name,
AirFormat.df_2( this.heatGain/this.zoneArea ));
                                                }
                                        }
                                }

                                if( CheckPrintModule.isPrintEnergy ){
                                        //eleOut
                                        super.rm.setRecord(super.getConnectionNode(this.R1_NODE),
                                                        this.RECORD_PPE, this.name,
AirFormat.df_2( this.eleIn.getActivePower()));
                                }

                                //if( CheckPrintModule.isPrintStateOut ){
                                //}

                                if( CheckPrintModule.isPrintStateMy ){
                                        super.rm.setRecord(super.getConnectionNode(this.R1_NODE),
                                                        this.RECORD_HeatGain, this.name,
AirFormat.df_2( this.heatGain ));
                                }

                                //if( CheckPrintModule.isPrintStateIn ){
                                //}
                        }

                        @Override
                        public void update() {

                        }


                        @Override
                        public Object viewInternal(TestCommand cmd) {
                                List<Object> result = new ArrayList<Object>();

                                result.add(super.sm);
                                result.add(super.rm);

                                result.add(this.modIn);
                                result.add(this.L);
                                result.add(this.V);
                                result.add(this.FT);
                                result.add(this.FE);
                                result.add(this.C);
                                result.add(this.EEVn);

                                return result;
                        }


}
```

「蓄電池 G2013」（場所：設備 2015／電気設備 2015／）

モジュール名	蓄電池 G2013
クラス	RechargeableBatteryModule201312

（1）入力画面

・スペック



（2）モジュールの概要

蓄電池の充電時・放電時の挙動・エネルギー損失量に関わるモジュールである。

本モジュールは、蓄電池本体部分をモデル化したモジュールで、蓄電池充放電制御モジュールと組み合わせて計算を行う。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

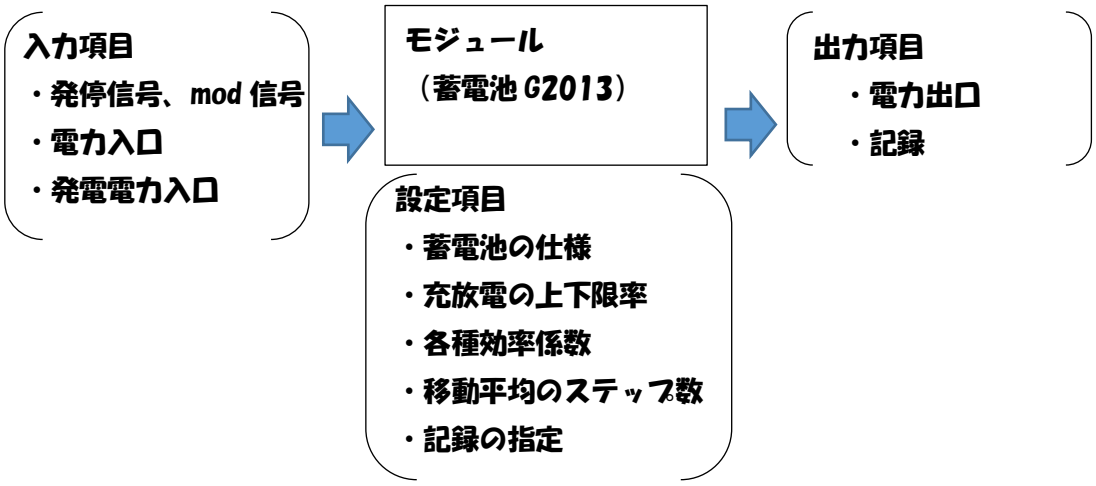


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	－	[-]	－	－		
1	■蓄電池■					－	－		
2	定格蓄電容量	double	capacity_des_Ws	10	[kWh]	－	0		
3	初期蓄電容量	double	capacity_start_Ws	0	[kWh]	－	0		←計算開始時の蓄電容量
4	容量保持率	double	maintenance_factor	0.8	[-]	1	0		←参考（Li=0.8、鉛＝0.8、NAS＝0.72）
5	定格放電電力	double	discharge_power_des_W	1	[kW]	－	0		
6	ベース放電電力	double	discharge_power_base_W	1	[kW]	－	0		←ピークシフト制御時のベース放電電力を入力してください
7	■充放電の上下限の設定等■					－	－		
8	放電停止下限充足率	double	min_operating_ratio	0	[-]	1	0		←放電を停止とする充電率の下限
9	充電停止上限充足率	double	max_operation_ratio	1	[-]	1	0		←充電を停止とする充電率の上限
10	充電時間率	double	charge_hour_factor	5	[-]	24	0		←充電時間率＝ 定格蓄電容量 ÷ 1時間当たりの目標充電容量
11	充電特性	String	chargingModel	0_リチウムイオン電池、1_鉛電池、2_NAS電池、3_充電時間比例、4_レドックスフロー電池	[-]	－	0		←充電特性を指定
12	■蓄電池システムの充放電・待機時の効率■					－	－		
13	ＰＣＳ充電時の効率	String	chargingPCS_factor	0.95	[-]	－	－		←参考（Li=0.95、鉛＝0.95、NAS＝0.95）

14	ＰＣＳ放電時の効率	String	dischargingPCS_factor	0.95	[-]	―	―		←参考（Li=0.95、鉛＝0.95、NAS＝0.95）
15	蓄電池本体の効率	double	battery_factor	0.95	[-]	1	0		←参考（Li=0.95、鉛＝0.85、NAS＝0.90）
16	その他（補機等）の効率	double	auxiliary_factor	1	[-]	1	0		←参考（Li=1.00、鉛＝1.00、NAS＝1.00）
17	待機時の効率	double	standby_factor	1	[-]	1	0		←参考（Li=1.00、鉛＝1.00、NAS＝0.86）
18	待機時の効率充電時	double	standby_factor_charge	1	[-]	1	0		←参考（Li=1.00、鉛＝1.00、NAS＝0.86）
19	■蓄電池システムの発熱■					―	―		
20	熱損失係数	double	eleLossheat_factor	1	[-]	1	0		←電力損失の熱への変換率
21	■供給先系統数・倍率■					―	―		
22	eleOut[]出口接続ノード数	int	numberOfOutlet	1	[-]	―	1		←電力の供給先系統数を整数で入力して下さい
23	■発電･CGS太陽光風力など■					―	―		
24	eleInG[]発電入口接続ノード数	int	numberOfInletGen	1	[-]	―	1		←発電の受入系統数を整数で入力して下さい
25	逆潮流する	boolean	isBackwardFlow	FALSE	[-]	―	―		←逆潮流するときはチェックしてください
26	■設置室■					―	―		
27	室グループ／室／ゾーン	String	grzName	#RoomGroup	[-]	―	―		←室グループ／室／ゾーンを選択してください。蓄電池の発熱をこの室へ渡します。
28	■発電出力移動平均値・発電出力変動抑制制御■					―	―		←出力変動抑制制御は、次の入力行の「発電電力移動平均値１」 に対して充放電制御します
29	発電電力移動平均値１の算定ステップ数	int	numAdjustSteps1	4	[-]	―	0		←発電電力の合計値の移動平均値１算定のための計算ステップ数を入力してください
30	発電電力移動平均値２の算定ステップ数	int	numAdjustSteps2	12	[-]	―	0		←発電電力の合計値の移動平均値２算定のための計算ステップ数を入力してください
31	発電電力移動平均値３の算定ステップ数	int	numAdjustSteps3	36	[-]		0		←発電電力の合計値の移動平均値３算定のための計算ステップ数を入力してください
32	■記録・グラフ表示■					―	―		

33	グラフを表示する	boolean	isGVisible	FALSE	[-]	ー	ー		←グラフを表示するときはチェックしてください
34	最大同時表示ステップ数	int	maxItemCount	100	[-]	ー	1		←グラフに同時表示する最大ステップ数を入力します
35	記録を有効とする	boolean	isRecord	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください
36	全記録を有効とする	boolean	isRecordALL	FALSE	[-]	ー	ー		←このモジュールの全記録を有効とするときはチェックしてください
37	■仮設調整■					ー	ー		
38	蓄電容量と放電能力を調整する	boolean	isAdjust2012	FALSE	[-]	ー	ー		←蓄電容量と ｊ 放電能力を調整するときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	空調記録モジュールへ接続する
2	発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	上位からの発停信号
3	充電信号	L1_swcInECharge	swcInECharge	-	制御	OnOff 信号	入口	上位からのモード信号
4	放電信号	L1_swcInEDischarge	swcInEDischarge	-	制御	OnOff 信号	入口	
5	モード信号	L1_modIn	modIn	-	制御	制御モード	入口	
6	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	
7	電力出口[]	L0_eleOut[]	eleOut[]	-	状態	電力	出口	
8	発電電力入口[]	L0_eleInGen[]	eleInGen[]	-	状態	電力	入口	
9	空気入口	L0_airIn	airIn	-	状態	空気	入口	
10	熱出口	L0_heaOut	heaOut	-	状態	熱	出口	
11	電力損失観察	L0_eleObsLoss	eleObsLoss	-	状態	電力	観察	
12	充電電力観察	L0_eleObsCharge	eleObsCharge	-	状態	電力	観察	
13	放電電力観察	L0_eleObsDischarge	eleObsDischarge	-	状態	電力	観察	
14	移動平均電力出口	L0_eleOutSmooth	eleOutSmooth	-	状態	電力	出口	
15	充電率の値の入口	L0_valInEChargeRate	valInEChargeRate	-	値	値	入口	
16	放電率の値の入口	L0_valInEDischargeRate	valInEDischargeRate	-	値	値	入口	
17	目標電力の値の入口	L0_valInTargetW	_valInTargetW	-	値	値	入口	ピークカット目標値など
18	使い切りのための放電率の入口	L0_valInZeroRate	valInZeroRate	-	値	値	入口	

19	平準化商用充電の上限充足率の入口	LO_valInOpeSetRate	valInOpeSetRate	-	値	値	入口	

（6）記録項目

No.	項目	出力内容	単位	分類
1	RB_Message#-#-	メッセージ	—	メッセージ
2	RB_入口有効電力#W#-	入口有効電力	W	入口
3	RB_入口無効電力#Var#-	入口無効電力	Var	入口
4	RB_蓄放電電力#W#-	蓄放電電力	W	My
5	RB_電池部分蓄放電電力#W#-	電池部分蓄放電電力	W	My
6	RB_待機時電力#W#-	待機時電力	W	My
7	RB_蓄電容量変化量#Ws/Step#-	蓄電容量変化量	Ws/Step	My
8	RB_蓄電容量#Ws#-	蓄電容量	Ws	My
9	RB_損失合計#Ws/Step#-	損失合計（Ws/Step)）	Ws/Step	My
10	RB_損失合計#W#-	損失合計（W）	W	My
11	RB_発熱量#W#-	発熱量	W	My
12	RB_充電率#-#-	充電率	—	My
13	RB_放電停止下限充足率#-#-	放電停止加減充足率	—	My
14	RB_充電停止上限充足率#-#-	充電停止上限充足率	—	My
15	RB_充電特性充電可能電力#W#-	充電特性充電可能電力	W	My
16	RB_定格放電電力#W#-	定格放電電力	W	My
17	RB_ベース放電電力#W#-	ベース放電電力	W	My
18	RB_発電電力合計有効電力#W#-	発電電力合計有効電力	W	My
19	RB_発電電力移動平均値 1#W#-	発電電力移動平均値１	W	My
20	RB_発電電力移動平均値 2#W#-	発電電力移動平均値２	W	My
21	RB_発電電力移動平均値 3#W#-	発電電力移動平均値３	W	My
22	RB_発電電力と移動平均値１の差#W#-	発電電力と移動平均値１の差	W	My
23	RB_発電電力と移動平均値２の差#W#-	発電電力と移動平均値２の差	W	My
24	RB_発電電力と移動平均値３の差#W#-	発電電力と移動平均値３の差	W	My
25	RB_発電電力出力変動抑制後#W#-	発電電力出力変動抑制後	W	出口
26	RB_出側合計有効電力＃W#-	出側合計有効電力	W	出口
27	RB_未消費発電電力＃W#-	未消費発電電力	W	My
28	RB_未消費放電電力＃W#-	未消費放電電力	W	My
29	A_eleOut	eleOut[i]有効電力	W	出口
30	R_eleOut	eleOut[i]無効電力	Var	出口
31	A_eleInG	eleInG[i]有効電力	W	入口

32	R_eleInG	eleIn[i]無効電力	Var	入口
33	RB_調整蓄電容量#Ws#-	調整蓄電容量	Ws	調整
34	RB_調整放電能力#W#-	調整放電能力	W	調整

（7）計算フロー・計算内容

省略。

（8）データ範囲と範囲外の取扱い

特になし。

参考　ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.util.LinkedList;
import java.util.Map;

import jp.or.ibec.best.DO.BestAir;
import jp.or.ibec.best.DO.BestElectricity;
import jp.or.ibec.best.DO.BestHeat;
import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.dk.test.TestCommand;
import jp.or.ibec.best.domain.building.load.SystemHeatGain;
import jp.or.ibec.best.domain.building.spaces.ISpace;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;

/**
 * @author HIROSHI NINOMIYA /20120909
 * 蓄電池　リチウムイオン　鉛　NAS　です
 * 20121022　改訂
 * //20130801　Heat→BestHeat
 */
public class RechargeableBatteryModule201312 extends AbstractBestModule implements
      IBestMetaModule {

   //
   private static final String moduleName = "(RechargeableBatteryModule201312) ";

   //接続ノード
   private static final String S_NODE_eleIn          = "L0_eleIn";
   private static final String S_NODE_eleLoss        = "L0_eleObsLoss";
   private static final String S_NODE_eleObsCharge   = "L0_eleObsCharge";
   private static final String S_NODE_eleObsDischarge= "L0_eleObsDischarge";
   private static final String S_NODE_eleOutSmooth   = "L0_eleOutSmooth";
   //
   private static String[] S_NODE_eleOut;
   private static String[] S_NODE_eleInGen;

   private static final String S_NODE_airIn  = "L0_airIn";

   private static final String S_NODE_heaOut = "L0_heaOut";//発熱
   //
   private static final String S_NODE_valInEChargeRate    = "L0_valInEChargeRate";//制御モジュールからの充電率[/h]
   private static final String S_NODE_valInEDischargeRate = "L0_valInEDischargeRate";//制御モジュールからの放電率[/h]
// private static final String S_NODE_valOutDemandele = "L0_valOutDemandele";

   private static final String S_NODE_valInTargetW    = "L0_valInTargetW";//ピークカット目標値など
   private static final String S_NODE_valInZeroRate   = "L0_valInZeroRate";//使い切りのための放電率
   private static final String S_NODE_valInOpeSetRate = "L0_valInOpeSetRate";//平準化商用充電の上限充足率

   private static final String C_NODE_swcIn = "L1_swcIn";
   private static final String C_NODE_modIn = "L1_modIn";

   private static final String C_NODE_swcInECharge    = "L1_swcInECharge";
   private static final String C_NODE_swcInEDischarge = "L1_swcInEDischarge";

   private static final String R_NODE = "L2_recOut";

   //仕様
   private static final String SPEC_name = "名称";

   private static final String SPEC_capacity_des_Ws   = "定格蓄電容量[Ws]";
   private static final String SPEC_capacity_start_Ws = "初期蓄電容量[Ws]";
```

```
    private static final String SPEC_discharge_power_des_W = "定格放電電力[W]";
//  private static final String SPEC_discharge_power_factor = "放電電力係数[-]";
    private static final String SPEC_discharge_power_base_W = "ベース放電電力[W]";

    private static final String SPEC_charge_hour_factor  = "充電時間率[-]";
    private static final String SPEC_chargingModel       = "充電特性[-]";

//  private static final String SPEC_target_power        = "ピークカット目標値[W]";

    private static final String SPEC_min_operating_ratio = "放電停止下限充足率[-]";
    private static final String SPEC_max_operation_ratio = "充電停止上限充足率[-]";

    private static final String SPEC_maintenance_factor  = "容量保持率[-]";

//  private static final String SPEC_dischargeContorolType  = "放電制御方式[-]";

    private static final String SPEC_chargingPCS_factor    = "ＰＣＳ充電時の効率[-]";
    private static final String SPEC_dischargingPCS_factor = "ＰＣＳ放電時の効率[-]";
    private static final String SPEC_battery_factor        = "蓄電池本体の効率[-]";
    private static final String SPEC_auxiliary_factor      = "その他補機等の効率[-]";
    private static final String SPEC_standby_factor        = "待機時の効率[-]";
    private static final String SPEC_standby_factor_charge = "待機時の効率充電時[-]";

    private static final String SPEC_eleLossheat_factor  = "熱損失係数[-]";

    private static final String SPEC_PowerFactor = "放電力率[-]";
    private static final String SPEC_PowerFactorStandby = "待機時力率[-]";

    private static final String SPEC_NumberOfOutlet   = "[]出口接続ノード数";
    private static final String SPEC_NumberOfInletGen = "[]発電入口接続ノード数";

    private static final String SPEC_isBackwardFlow = "is逆潮流する";
    //
    private static final String SPEC_grzName = "室グループ／室／ゾーン";
    //
    private static final String SPEC_isAdjust2012   = "蓄電容量と放電能力を調整する";

    private static final String SPEC_isGVisible   = "グラフを表示する";
    private static final String SPEC_maxItemCount = "最大同時表示ステップ数";
    private static final String SPEC_isRecord     = "記録を有効とする";
    private static final String SPEC_isRecordALL  = "全記録を有効とする";

    private static final String SPEC_rf_a1 ="負荷係数a1[-]";//負荷係数a1[-]
    private static final String SPEC_rf_a2 ="負荷係数a2[-]";//負荷係数a2[-]
    private static final String SPEC_rf_b3 ="負荷係数b3[-]";//負荷係数b3[-]
    private static final String SPEC_rf_b4 ="負荷係数b4[-]";//負荷係数b4[-]
    private static final String SPEC_rf_c1 ="負荷係数c1[-]";//負荷係数c1[-]
    private static final String SPEC_rf_d1 ="負荷係数d1[-]";//負荷係数d1[-]
    private static final String SPEC_rf_d2 ="負荷係数d2[-]";//負荷係数d2[-]
    private static final String SPEC_rf_d3 ="負荷係数d3[-]";//負荷係数d3[-]
    private static final String SPEC_rf_f1 ="負荷係数f1[-]";//負荷係数f1[-]
    private static final String SPEC_rf_f2 ="負荷係数f2[-]";//負荷係数f2[-]
    private static final String SPEC_rf_f3 ="負荷係数f3[-]";//負荷係数f3[-]
    private static final String SPEC_rf_g1 ="負荷係数g1[-]";//負荷係数g1[-]

    //記録
    private static final String RECORD_message      = "RB_Message#-#-";
    private static final String RECORD_A_eleIn      = "RB_入口有効電力#W#-";
    private static final String RECORD_R_eleIn      = "RB_入口無効電力#Var#-";
    private static final String RECORD_chargedWatt  = "RB_蓄放電電力#W#-";
    private static final String RECORD_b_chargedWatt= "RB_電池部分蓄放電電力#W#-";
    private static final String RECORD_powerStandby = "RB_待機時電力#W#-";

    private static final String RECORD_chargedWS    = "RB_蓄電容量変化量#Ws/Step#-";
    private static final String RECORD_chargedWSSum = "RB_蓄電容量#Ws#-";
    private static final String RECORD_lossWSs      = "RB_損失合計#Ws/Step#-";
    private static final String RECORD_lossW        = "RB_損失合計#W#-";

    private static final String RECORD_heatLossW    = "RB_発熱量#W#-";

    private static final String RECORD_chargedRate  = "RB_充電率#-#-";
```

```
private static final String RECORD_min_operating_ratio = "RB_放電停止下限充足率#-#-";
private static final String RECORD_max_operating_ratio = "RB_充電停止上限充足率#-#-";

private static final String RECORD_maxChargeableW2CA = "RB_充電特性充電可能電力#W#-";
private static final String RECORD_discharge_power_des_W  = "RB_定格放電電力#W#-";
private static final String RECORD_discharge_power_base_W = "RB_ベース放電電力#W#-";

private static final String RECORD_A_eleInGSum    = "RB_発電電力合計有効電力#W#-";
private static final String RECORD_avePowerOutG1  = "RB_発電電力移動平均値1#W#-";
private static final String RECORD_avePowerOutG2  = "RB_発電電力移動平均値2#W#-";
private static final String RECORD_avePowerOutG3  = "RB_発電電力移動平均値3#W#-";
private static final String RECORD_davePowerOutG1 = "RB_発電電力と移動平均値1の差#W#-";
private static final String RECORD_davePowerOutG2 = "RB_発電電力と移動平均値2の差#W#-";
private static final String RECORD_davePowerOutG3 = "RB_発電電力と移動平均値3の差#W#-";
private static final String RECORD_smoothPowerOutG= "RB_発電電力出力変動抑制後#W#-";

private static final String RECORD_A_eleOutSum = "RB_出側合計有効電力＃W#-";

private static final String RECORD_powerGenUnconsumed = "RB_未消費発電電力＃W#-";
private static final String RECORD_powerDischargeUnconsumed = "RB_未消費放電電力＃W#-";

private static String[] RECORD_A_eleOut = null;
private static String[] RECORD_R_eleOut = null;
private static String[] RECORD_A_eleInG = null;
private static String[] RECORD_R_eleInG = null;

private static final String RECORD_capacity_des_Ws_Adjust       = "RB_調整蓄電容量#Ws#-";//定格蓄電容量[Ws]
private static final String RECORD_discharge_power_des_W_Adjust = "RB_調整放電能力#W#-";//定格放電能力[W]";

//接続熱媒など
private BestElectricity[] eleOut  = null;
private BestElectricity[] eleInGen = null;
private BestElectricity eleIn   = null;
private BestElectricity eleLoss = null;
private BestElectricity eleObsCharge = null;
private BestElectricity eleObsDischarge = null;
private BestElectricity eleOutSmooth = null;

private BestAir airIn = null;

private BestHeat heaOut;//20130801  Heat→BestHeat

private BestValue valInEChargeRate = null;//"LO_valInEChargeRate";//制御モジュールからの充電率[/h]

private BestValue valInEDischargeRate = null;//"LO_valInEDischargeRate";//制御モジュールからの放電率[/h]

private BestValue valInTargetW = null;//ピークカット、発電電力平準化など目標値

private BestValue valInZeroRate = null;//使い切りのための放電率

private BestValue valInOpeSetRate = null;//平準化商用充電の上限充足率

//制御信号など
private int swcIn;
private int modIn;

private int swcInECharge;
private int swcInEDischarge;
//仕様など
private String name;
private int numberOfOutlet;         //出口　接続数
private int numberOfInletGen;       //入口　接続数

private boolean isBackwardFlow = false;//逆潮流する=true

private double capacity_max_Ws;//最大蓄電容量[Ws]
private double capacity_des_Ws;//定格蓄電容量[Ws]
private double capacity_start_Ws;//初期蓄電容量[Ws]
private double capacity_opeRange_Ws;//運転時の上下限充電率域の蓄電容量[Ws]

private double discharge_power_des_W;//定格放電能力[W]";
private double discharge_power_max_W;//最大放電能力[W]";
```

```
    private double discharge_power_base_W;//ベース放電能力[W]";
    private double discharge_hour_factor;// = "放電時間率[-]";

    private double charge_hour_factor;// = "充電時間率[-]";
    private double charge_power_max_W;//最大充電電力[W]";

    private double charge_power_valInECR;//制御モジュールからの充電率から求めた充電電力[W]
    private double discharge_power_valInEDCR;//制御モジュールからの充電率から求めた放電電力[W]

    private int chargingModel;//充電特性のタイプ番号

    private double target_power;//   = "ピークカット目標値[W]","発電電力平準化目標値[W]";

    private double min_operating_ratio;// = "放電停止下限充電率[-]";
    private double max_operating_ratio;// = "充電停止上限充電率[-]";
    private double operating_ratio;// = "充電率[-]"

    private double maintenance_factor;//  = "容量保持率[-]";

    private double des_chargingPCS_factor;    //= "ＰＣＳ充電時の効率[-]";
    private double[] des_chargingPCS_factors;

    private double des_dischargingPCS_factor; //= "ＰＣＳ放電時の効率[-]";
    private double[] des_dischargingPCS_factors;

    private double battery_factor;        //= "蓄電池の効率[-]";
    private double auxiliary_factor;      //= "その他補機等の効率[-]";

    private double charging_factor   ;//  = "充電時の効率[-]";
    private double discharging_factor;//  = "放電時の効率[-]";
    private double standby_factor    ;//  = "待機時の効率[-]";
    private double standby_factor_charge;//  = "待機時の効率充電時[-]";

    private double eleLossheat_factor;//  = "熱損失係数[-]";

    private double powerFactor;//放電力率[-]
    private double powerFactorStandby;//待機時力率[-]

    private double heatLossW;//  = "RB_発熱量#W#-";

    double powerStandby; //待機時の電力
    double des_powerStandby; //待機時の電力
    double des_powerStandby_charge; //待機時の電力充電時

// private String dischargeContorolType = null;//  = "放電制御方式[-]";

    private double chargedWattSecondsSum;//[Ws]
    private double chargedWattSecondsSum_min_Ws;//[Ws]最小充電量
    private double lossWattSeconds;//[Ws]

    private double chargedWatt;//[W]蓄電池システムとして
    private double b_chargedWatt;//[W]蓄電池部分
    private double chargedWattSeconds;//[Ws]

// private double lossRate = 0.0015;//[-]

// private boolean isBackwardFlow = false;//逆潮流する=true
    //
    private boolean isGVisible = false;//このグラフを表示する=true
    private int  maxItemCount = 100;//最大同時表示ステップ数
    private boolean isRecord   = false;//記録を有効とする=true
    private boolean isRecordALL= false;//記録を有効とする=true

// private double rf_a1 ;//負荷係数a1[-]
// private double rf_a2 ;//負荷係数a2[-]
// private double rf_b3 ;//負荷係数b3[-]
// private double rf_b4 ;//負荷係数b4[-]
// private double rf_c1 ;//負荷係数c1[-]
// private double rf_d1 ;//負荷係数d1[-]
// private double rf_d2 ;//負荷係数d2[-]
// private double rf_d3 ;//負荷係数d3[-]
// private double rf_f1 ;//負荷係数f1[-]
```

```
// private double rf_f2 ;//負荷係数f2[-]
// private double rf_f3 ;//負荷係数f3[-]
// private double rf_g1 ;//負荷係数g1[-]

    private StringBuffer message = new StringBuffer();

    private boolean isEleCharge = false;
    private boolean isEleDischarge = false;
    private boolean isEleSurplusCharge = false;//出力補正制御時の余剰分の充電

    private boolean isPeakCut  = false;
    private boolean isPeakCutALL  = false;
    private boolean isPeakShift = false;
    private boolean isConstPower = false;//発電電力平準化（一定値）制御
    private boolean isCtrlPowerChangeRate = false;//出力変動抑制制御
    private boolean isLoadFollowingControl = false;//負荷追従制御

    /**
     * 充電可能量[W]  2CA特性より
     */
    private double maxChargeableW2CA;

    private double powerGenUnconsumed;//未消費発電電力[W]
    private double powerDischargeUnconsumed;//未消費放電電力[W]

    //グラフ表示など
    private GraphCommonJFrame2015 gGCJF = null;
    private GraphCommonData2015[] gData = null;


    private boolean isCalcEleOut = false;
    private boolean isCalEleInGen = false;

    /**
     * 蓄電池タイプ番号  0_リチウムイオン電池、1_鉛電池、2_NAS電池、3_充電時間比例
     */
    protected static final int RBTYPE_LIB = 0;//0_リチウムイオン電池
    protected static final int RBTYPE_PBB = 1;//1_鉛電池
    protected static final int RBTYPE_NAS = 2;//2_NAS電池
    protected static final int RBTYPE_FLI = 3;//3_充電時間比例
    protected static final int RBTYPE_RFB = 4;//_レドックスフロー電池
    //
// private static final String SPEC_isAdjust       = "発電出力変動を抑制する";

    private static final String SPEC_NumAdjustSteps1 = "発電電力移動平均値１の算定ステップ数[-]";
    private static final String SPEC_NumAdjustSteps2 = "発電電力移動平均値２の算定ステップ数[-]";
    private static final String SPEC_NumAdjustSteps3 = "発電電力移動平均値３の算定ステップ数[-]";

    private double changeFlowRate;

    //private boolean isAdjust  = false;//true="最大風量を調整する";//
    //private boolean isAdjustPowerOutG = false;//発電出力変動を抑制する
    private int numAdjustSteps1;//"調整の計算ステップ数";
    private int numAdjustSteps2;//"調整の計算ステップ数";
    private int numAdjustSteps3;//"調整の計算ステップ数";
    private double maxChangeFlowRate;

    private LinkedList<Double> powerOutList1 = null;
    private LinkedList<Double> powerOutList2 = null;
    private LinkedList<Double> powerOutList3 = null;
    private double maxAdjustFlowRate = 0;
    private double aveAdjustPowerInG1 = 0;
    private double aveAdjustPowerInG2 = 0;
    private double aveAdjustPowerInG3 = 0;
    //private double maxFlowRate;
    private double minFlowRate;

    private ISpace coreSpace   ;//建物側インスタンス
    private double zoneArea;//ゾーンの床面積
    private SystemHeatGain systemHeatGain; //建物側インスタンス
    private boolean isSystemHeatGain = false;
    //
```

```
//仮設調整モード2012
private boolean isAdjust2012  = false;//true="蓄電容量と放電能力を調整する";//
private double capacity_des_Ws_1;//定格蓄電容量[Ws]
private double discharge_power_des_W_1;//定格放電能力[W]";


@Override
public void setProfile(BestSpecs spec) {
   if(spec == null) {
      return;
   }

   Map<String, String> map = spec.getSpec();
   if(map == null) {
      return;
   }

   //名称を取得
   this.name = spec.getSpecValue( SPEC_name, moduleName );

   //numberOfOutletを取得
   this.numberOfOutlet = spec.getSpecValue( SPEC_NumberOfOutlet, 0 );
   //出口ｎ系統のBestWaterの初期化
   if( this.numberOfOutlet > 0 ){
      this.isCalcEleOut = true;
      this.eleOut = new BestElectricity[this.numberOfOutlet];
      S_NODE_eleOut = new String[this.numberOfOutlet];
      RECORD_A_eleOut = new String[this.numberOfOutlet];
      RECORD_R_eleOut = new String[this.numberOfOutlet];
      for( int i=0; i<this.eleOut.length; i++ ){
         //this.eleOut[i] = new BestElectricity();
         //this.eleOut[i] = new BestElectricity( 0, 0, 200, 0, 3, 50);
         S_NODE_eleOut[i]   = new String( "LO_eleOut[" + i + "]" );
         RECORD_A_eleOut[i] = new String( "RB_eleOut[" + i + "]有効電力#W#有効電力" );
         RECORD_R_eleOut[i] = new String( "RB_eleOut[" + i + "]無効電力#Var#無効電力" );
      }
   }

   //numberOfInletGenを取得
   this.numberOfInletGen = spec.getSpecValue( SPEC_NumberOfInletGen, 0 );
   //出口ｎ系統のBestWaterの初期化
   if( this.numberOfInletGen > 0 ){
      this.isCalEleInGen = true;
      this.eleInGen = new BestElectricity[this.numberOfInletGen];
      S_NODE_eleInGen = new String[this.numberOfInletGen];
      RECORD_A_eleInG = new String[this.numberOfInletGen];
      RECORD_R_eleInG = new String[this.numberOfInletGen];
      for( int i=0; i<this.eleInGen.length; i++ ){
         //this.eleOut[i] = new BestElectricity();
         //this.eleOut[i] = new BestElectricity( 0, 0, 200, 0, 3, 50);
         S_NODE_eleInGen[i] = new String( "LO_eleInGen[" + i + "]" );
         RECORD_A_eleInG[i] = new String( "RB_eleInGen[" + i + "]有効電力#W#有効電力" );
         RECORD_R_eleInG[i] = new String( "RB_eleInGen[" + i + "]無効電力#Var#無効電力" );
      }
   }

   //SPEC_capacity_desを取得
   this.capacity_des_Ws = spec.getSpecValue( SPEC_capacity_des_Ws, 3600000. );

   //SPEC_maintenance_factor  = "容量保持率[-]";
   this.maintenance_factor = spec.getSpecValue( SPEC_maintenance_factor, 0.8 );

   //最大蓄電容量[Ws]
   this.capacity_max_Ws = this.capacity_des_Ws * this.maintenance_factor;
   //if( this.capacity_max_Ws == 0 ){
   // this.capacity_max_Ws = 1.;
   //}

   //SPEC_capacity_start = "初期蓄電容量[Ws]";
   this.capacity_start_Ws = spec.getSpecValue( SPEC_capacity_start_Ws, 0. );
```

```
        if( this.capacity_max_Ws == 0 ){
           this.operating_ratio = 0;
        }else{
           this.operating_ratio = this.capacity_start_Ws / this.capacity_max_Ws;
           this.chargedWattSecondsSum = - this.capacity_max_Ws * this.operating_ratio;
        }

        //SPEC_discharge_power_des_W = "定格放電能力[W]";
        this.discharge_power_des_W = spec.getSpecValue( SPEC_discharge_power_des_W, 1000. );

        this.discharge_power_max_W = this.discharge_power_des_W;

//      //SPEC_discharge_power_factor = "放電電力係数[-]";
//      this.discharge_power_factor = spec.getSpecValue( this.SPEC_discharge_power_factor, 0.2 );
        if( this.discharge_power_max_W == 0 ){
           this.discharge_hour_factor = 0.1;
        }else{
           this.discharge_hour_factor = this.capacity_max_Ws / this.discharge_power_max_W / 3600.;
        }

        //SPEC_discharge_power_base_W = "ベース放電能力[W]";
        this.discharge_power_base_W = spec.getSpecValue( SPEC_discharge_power_base_W, 1000. );

        //SPEC_charge_power_factor = "充電電力係数[-]";
        this.charge_hour_factor = spec.getSpecValue( SPEC_charge_hour_factor, 5. );
        if( this.charge_hour_factor == 0 ){
           this.charge_hour_factor = 0.1;
        }else{
           this.charge_hour_factor = 1./ this.charge_hour_factor;
        }

        //SPEC_chargingModel       = "充電特性[-]";
        String typeName = spec.getSpecValue( SPEC_chargingModel, "0_リチウムイオン電池" );

        if( typeName.equals( "0_リチウムイオン電池" ) ){
           this.chargingModel = RBTYPE_LIB;
        }else if( typeName.equals( "1_鉛電池" )){
           this.chargingModel = RBTYPE_PBB;
        }else if( typeName.equals( "2_NAS電池" )){
           this.chargingModel = RBTYPE_NAS;
        }else if( typeName.equals( "3_充電時間比例" )){
           this.chargingModel = RBTYPE_FLI;
        }else if( typeName.equals( "4__レドックスフロー電池" )){
           this.chargingModel = RBTYPE_RFB;
        }else{
           this.chargingModel = RBTYPE_FLI;
        }

        //SPEC_target_power    = "ピークカット目標値[W]";
        //valInTargetから受け取るように変更
//      this.target_power = spec.getSpecValue( this.SPEC_target_power, 0. );

        //SPEC_min_operating_ratio = "放電停止下限充電率[-]";
        this.min_operating_ratio = spec.getSpecValue( SPEC_min_operating_ratio, 0. );

        this.chargedWattSecondsSum_min_Ws = this.capacity_max_Ws * this.min_operating_ratio;//[Ws]

        //SPEC_max_operation_ratio = "充電停止上限充電率[-]";
        this.max_operating_ratio = spec.getSpecValue( SPEC_max_operation_ratio, 1. );

        this.capacity_opeRange_Ws
        = ( this.max_operating_ratio - this.min_operating_ratio ) * this.capacity_max_Ws;//[Ws]

        //    //SPEC_dischargeContorolType  = "放電制御方式[-]";
//      this.dischargeContorolType = spec.getSpecValue( this.SPEC_dischargeContorolType, "0_ピークシフト制御" );

//      if( this.dischargeContorolType.equals( "0_ピークシフト制御" ) ){
//         this.isPeakShift = true;
//      }else if( this.dischargeContorolType.equals( "1_ピークカット制御" ) ){
//         this.isPeakCut = true;
//      }else if( this.dischargeContorolType.equals( "2_出力補正制御" ) ){
//         this.isConstPower = true;
```

```
//    }

      //SPEC_chargingPCS_factor    = "ＰＣＳ充電時の効率[-]";
      //this.chargingPCS_factor = spec.getSpecValue( this.SPEC_chargingPCS_factor, 0.95 );
      //if( this.chargingPCS_factor == 0 ){
      // this.chargingPCS_factor = 0.01;
      //}
      this.des_chargingPCS_factors = new double[3];
      String str1 = spec.getSpecValue( SPEC_chargingPCS_factor, "0.95" );
      String[] strs1 = str1.split( " " );
      if( strs1.length==3 ) {
         this.des_chargingPCS_factors[0] = Double.parseDouble( strs1[0] );
         this.des_chargingPCS_factors[1] = Double.parseDouble( strs1[1] );
         this.des_chargingPCS_factors[2] = Double.parseDouble( strs1[2] );
      }else {
         this.des_chargingPCS_factors[0] = 0;
         this.des_chargingPCS_factors[1] = 0;
         this.des_chargingPCS_factors[2] = 1. / Double.parseDouble( str1 );
      }
      this.des_chargingPCS_factor = this.calc_PCSefficiency( this.des_chargingPCS_factors, 1. );
      if( this.des_chargingPCS_factor == 0 ){
         this.des_chargingPCS_factor = 0.01;
      }
      System.out.println("new des_chargingPCS_facto="+this.des_chargingPCS_factor );

      //SPEC_dischargingPCS_factor = "ＰＣＳ放電時の効率[-]";
      //this.dischargingPCS_factor = spec.getSpecValue( this.SPEC_dischargingPCS_factor, 0.95 );
      //if( this.dischargingPCS_factor == 0 ){
      // this.dischargingPCS_factor = 0.01;
      //}
      this.des_dischargingPCS_factors = new double[3];
      String str2 = spec.getSpecValue( SPEC_dischargingPCS_factor, "0.95" );
      String[] strs2 = str2.split( " " );
      if( strs2.length==3 ) {
         this.des_dischargingPCS_factors[0] = Double.parseDouble( strs2[0] );
         this.des_dischargingPCS_factors[1] = Double.parseDouble( strs2[1] );
         this.des_dischargingPCS_factors[2] = Double.parseDouble( strs2[2] );
      }else {
         this.des_dischargingPCS_factors[0] = 0;
         this.des_dischargingPCS_factors[1] = 0;
         this.des_dischargingPCS_factors[2] = 1. / Double.parseDouble( str2 );
      }
      this.des_dischargingPCS_factor = this.calc_PCSefficiency( this.des_dischargingPCS_factors, 1. );
      if( this.des_dischargingPCS_factor == 0 ){
         this.des_dischargingPCS_factor = 0.01;
      }
      System.out.println("new des_dischargingPCS_facto="+this.des_dischargingPCS_factor );

      //SPEC_battery_factor        = "蓄電池の効率[-]";
      this.battery_factor = spec.getSpecValue( SPEC_battery_factor, 0.95 );
      if( this.battery_factor == 0 ){
         this.battery_factor = 0.01;
      }

      //SPEC_auxiliary_factor      = "その他補機等の効率[-]";
      this.auxiliary_factor = spec.getSpecValue( SPEC_auxiliary_factor, 1.0 );
      if( this.auxiliary_factor == 0 ){
         this.auxiliary_factor = 0.01;
      }

      //SPEC_standby_factor      = "待機時の効率[-]";
      this.standby_factor = spec.getSpecValue( SPEC_standby_factor, 1.0 );
      if( this.standby_factor == 0 ){
         this.standby_factor = 0.01;
      }

      //SPEC_standby_factor_charge      = "待機時の効率充電時[-]";
      this.standby_factor_charge = spec.getSpecValue( SPEC_standby_factor_charge, 1.0 );
      if( this.standby_factor_charge == 0 ){
         this.standby_factor_charge = 0.01;
      }
```

```
        this.charging_factor = this.des_chargingPCS_factor * this.auxiliary_factor * this.battery_factor;
        this.discharging_factor = this.des_dischargingPCS_factor;

        //SPEC_eleLossheat_factor  = "熱損失係数[-]";
        this.eleLossheat_factor = spec.getSpecValue( SPEC_eleLossheat_factor, 1.0 );

        //SPEC_PowerFactor = "放電力率[-]";
        this.powerFactor = spec.getSpecValue( SPEC_PowerFactor, 1.0 );

        //SPEC_PowerFactorStandby = "待機時力率[-]";
        this.powerFactorStandby = spec.getSpecValue( SPEC_PowerFactorStandby, 1.0 );

        //isBackwardFlowを取得
        this.isBackwardFlow = spec.getSpecValue( SPEC_isBackwardFlow, false );


        //this.charge_power_max_W = this.capacity_max_Ws * this.charge_hour_factor / 3600.;
        this.charge_power_max_W
        = this.capacity_des_Ws * this.charge_hour_factor / this.charging_factor / 3600.;//20150323定格蓄電容量に充電時間率
を適用する

        if( this.charge_power_max_W > this.discharge_power_des_W ){//定格放電能力[W]による上限チェック
           this.charge_power_max_W = this.discharge_power_des_W;
        }

        //室グループ、ゾーン名の取得
        String grzName = null;
        grzName = spec.getSpecValue( SPEC_grzName, "" );
        if( grzName.equals( "" ) ){
           this.isSystemHeatGain = false;
        }else{
           this.isSystemHeatGain = true;//systemHeatGain で発熱をゾーンへ渡す
           //SystemHeatGainインスタンスを建物側に登録
           this.coreSpace = ZoneGRZName.getCoreSpace( grzName, this.name );
           this.zoneArea = this.coreSpace.getValue( ISpace.FLOORAREA );
           this.systemHeatGain = new SystemHeatGain();
           this.systemHeatGain.setProfile(
                 ZoneGRZName.getMultiSpaceName( grzName, this.name ),
                 ZoneGRZName.getZoneName( grzName, this.name ) );
        }

        //isGVisibleを取得
        this.isGVisible = spec.getSpecValue( SPEC_isGVisible, false );

        // 最大同時表示ステップ数
        this.maxItemCount = spec.getSpecValue( SPEC_maxItemCount, 100 );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( SPEC_isRecord, false );
        this.isRecordALL = spec.getSpecValue( SPEC_isRecordALL, false );

        //SPEC_isAdjust = "最大風量を調整する";//
        //this.SPEC_isAdjust = spec.getSpecValue( this.SPEC_isAdjust, false );

        // 調整の計算ステップ数
        this.numAdjustSteps1 = spec.getSpecValue( SPEC_NumAdjustSteps1, 4 );
        this.numAdjustSteps2 = spec.getSpecValue( SPEC_NumAdjustSteps2, 12 );
        this.numAdjustSteps3 = spec.getSpecValue( SPEC_NumAdjustSteps3, 36 );

        this.powerOutList1 = new LinkedList<Double>();
        for( int i=0; i<this.numAdjustSteps1; i++ ){
           this.powerOutList1.add( 0. );
        }
        this.powerOutList2 = new LinkedList<Double>();
        for( int i=0; i<this.numAdjustSteps2; i++ ){
           this.powerOutList2.add( 0. );
        }

        this.powerOutList3 = new LinkedList<Double>();
        for( int i=0; i<this.numAdjustSteps3; i++ ){
           this.powerOutList3.add( 0. );
        }
```

```
        //蓄電池の待機時の電力
        this.des_powerStandby = this.discharge_power_des_W * ( 1. - this.standby_factor );
        this.des_powerStandby_charge = this.discharge_power_des_W * ( 1. - this.standby_factor_charge );

        this.isAdjust2012 = spec.getSpecValue( SPEC_isAdjust2012, false );

        if( this.isAdjust2012 ){
           this.capacity_des_Ws_1 = this.capacity_des_Ws;
           this.discharge_power_des_W_1 = this.discharge_power_des_W;//定格放電能力[W]";
        }

        //private double rf_a1 ;//負荷係数a1[-]
//      this.rf_a1 = spec.getSpecValue( this.SPEC_rf_a1, 0.032 );
        //private double rf_a2 ;//負荷係数a2[-]
//      this.rf_a2 = spec.getSpecValue( this.SPEC_rf_a2, 0.048 );
        //private double rf_b3 ;//負荷係数b3[-]
//      this.rf_b3 = spec.getSpecValue( this.SPEC_rf_b3, 0.016 );
        //private double rf_b4 ;//負荷係数b4[-]
//      this.rf_b4 = spec.getSpecValue( this.SPEC_rf_b4, 0.005 );
        //private double rf_c1 ;//負荷係数c1[-]
//      this.rf_c1 = spec.getSpecValue( this.SPEC_rf_c1, 0.012 );
        //private double rf_d1 ;//負荷係数d1[-]
//      this.rf_d1 = spec.getSpecValue( this.SPEC_rf_d1, 0.9 );
        //private double rf_d2 ;//負荷係数d2[-]
//      this.rf_d2 = spec.getSpecValue( this.SPEC_rf_d2, 0.75 );
        //private double rf_d3 ;//負荷係数d3[-]
//      this.rf_d3 = spec.getSpecValue( this.SPEC_rf_d3, 0.5 );
        //private double rf_f1 ;//負荷係数f1[-]
//      this.rf_f1 = spec.getSpecValue( this.SPEC_rf_f1, 0. );
        //private double rf_f2 ;//負荷係数f2[-]
//      this.rf_f2 = spec.getSpecValue( this.SPEC_rf_f2, 0.5 );
        //private double rf_f3 ;//負荷係数f3[-]
//      this.rf_f3 = spec.getSpecValue( this.SPEC_rf_f3, 1. );
        //private double rf_g1 ;//負荷係数g1[-]
//      this.rf_g1 = spec.getSpecValue( this.SPEC_rf_g1, 0.008 );

    }

    /**
     * @param _efficiency  効率式の係数配列  参考{0.02,1.05,0.018}
     * @param _ratio 負荷率[0-1]
     * @return
     */
    private double calc_PCSefficiency( double[] _efficiency, double _ratio ) {
        double eff =  _efficiency[0] * _ratio * _ratio + _efficiency[1] * _ratio + _efficiency[2];
        if( eff <= 0 ) {
            eff = 1;
        }else {
            eff = _ratio / eff;
        }
        return eff;
    }

    private void calcAdjustMaxDischargePower( double changeDischarge_power_W ){
        if( !this.isAdjust2012 ){
            return;
        }

        this.discharge_power_des_W = changeDischarge_power_W;
        this.discharge_power_max_W = this.discharge_power_des_W;

        System.out.println( "calcAdjustMaxDischargePower -->"+ this.discharge_power_max_W);
    }

    private void calcAdjust( double demand_dischargeW,  double maxdischargeW ){
        if( !this.isAdjust2012 ){
            return;
        }
        if( demand_dischargeW < maxdischargeW ){
            return;
        }
```

```
      this.capacity_des_Ws += (demand_dischargeW - maxdischargeW) * this.timeInterval;

      if( demand_dischargeW >  this.discharge_power_des_W ){
         this.discharge_power_des_W = demand_dischargeW;
         this.discharge_power_max_W = this.discharge_power_des_W;
      }


      //最大蓄電容量[Ws]
      this.capacity_max_Ws = this.capacity_des_Ws * this.maintenance_factor;
      //
      this.charge_power_max_W
      = this.capacity_des_Ws * this.charge_hour_factor / this.charging_factor / 3600.;//20150323定格蓄電容量に充電時間率を適用する
   // this.charge_power_max_W *= this.valInOpeSetRate.getValue();

      if( this.charge_power_max_W > this.discharge_power_des_W ){//定格放電能力[W]による上限チェック
         this.charge_power_max_W = this.discharge_power_des_W; //2019.1.15
   //    this.discharge_power_des_W = this.charge_power_max_W; //2019.1.15
      }


      //SPEC_min_operating_ratio = "放電停止下限充電率[-]";
      this.chargedWattSecondsSum_min_Ws = this.capacity_max_Ws * this.min_operating_ratio;//[Ws]
      //SPEC_max_operation_ratio = "充電停止上限充電率[-]";
      this.capacity_opeRange_Ws
      = ( this.max_operating_ratio - this.min_operating_ratio ) * this.capacity_max_Ws;//[Ws]

      //System.out.println( "calcAdjust addCapacity=" + ((demand_dischargeW - maxdischargeW) * this.timeInterval)
      //    + "-->capacity_des_Ws="+capacity_des_Ws
      //    + " CapamaxWs="+this.capacity_max_Ws
      //    +" charge_power_max_W="+charge_power_max_W);

   }

   public void initialize(IBestStateMessage stateNodes,
         IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
         IBestDomainSchedule schedule) {
      //状態ノードを取得
      super.sm = stateNodes;

      //制御ノードを取得
      super.cm = commandNodes;

      //記録ノードを取得
      super.rm = recordNodes;

      //接続ノード　出口
      if( this.isCalcEleOut ){
         for( int i=0; i<this.eleOut.length; i++ ){
            this.eleOut[i]
            = BestElectricity.bindnode( super.transferMapState, super.sm, S_NODE_eleOut[i] );
         }
      }

      //接続ノード　入口
      this.eleIn
      = BestElectricity.bindnode( super.transferMapState, super.sm, S_NODE_eleIn );


      //発電入口
      if( this.isCalEleInGen ){
         for( int i=0; i<this.eleInGen.length; i++ ){
            this.eleInGen[i]
            = BestElectricity.bindnode( super.transferMapState, super.sm, S_NODE_eleInGen[i] );
         }
      }

      //eleLoss
      this.eleLoss
      = BestElectricity.bindnode( super.transferMapState, super.sm, S_NODE_eleLoss );
```

```
    //eleCharge
    this.eleObsCharge
    = BestElectricity.bindnode( super.transferMapState, super.sm, S_NODE_eleObsCharge );

    //eleObsDischarge
    this.eleObsDischarge
    = BestElectricity.bindnode( super.transferMapState, super.sm, S_NODE_eleObsDischarge );

    //eleOutSmooth
    this.eleOutSmooth
    = BestElectricity.bindnode( super.transferMapState, super.sm, S_NODE_eleOutSmooth );

    //airIn
    this.airIn
    = BestAir.bindnode( super.transferMapState, super.sm, S_NODE_airIn );

    //HeaOut
    if( super.sm.getState( super.getConnectionNode( S_NODE_heaOut )) != null ){
       this.heaOut
       = (BestHeat) super.sm.getState( super.getConnectionNode( S_NODE_heaOut));
    }else{

       this.heaOut = new BestHeat();
       super.sm.setState( super.getConnectionNode( S_NODE_heaOut), this.heaOut );
    }

    // private final String S_NODE_valInEChargeRate = "LO_valInEChargeRate";//制御モジュールからの充電率[/h]
    this.valInEChargeRate
    = BestValue.bindnode( super.transferMapState,  super.sm, S_NODE_valInEChargeRate );

    this.valInEDischargeRate
    = BestValue.bindnode( super.transferMapState,  super.sm, S_NODE_valInEDischargeRate );

    this.valInTargetW
    = BestValue.bindnode( super.transferMapState,  super.sm, S_NODE_valInTargetW );

    this.valInZeroRate
    = BestValue.bindnode( super.transferMapState,  super.sm, S_NODE_valInZeroRate );

    this.valInOpeSetRate
    = BestValue.bindnode( super.transferMapState,  super.sm, S_NODE_valInOpeSetRate );

    this.target_power = this.valInTargetW.getValue();

    //20121108
//  this.lossRate /= ( 24. * 3600. );

    //グラフ表示の準備
    if( this.isGVisible ){

       int count = 0;
       if( this.isAdjust2012 ){
          count = 1;
       }
       this.gData = new GraphCommonData2015[20+count];

       int i = 0;

       this.gData[i++] = new GraphCommonData2015( "蓄放電電力[W]", this.maxItemCount,
             -this.charge_power_max_W * 1.1, this.discharge_power_des_W * 1.1, 0 );
       this.gData[i++] = new GraphCommonData2015( "電池部分蓄放電電力[W]", this.maxItemCount,
             -this.charge_power_max_W * 1.1, this.discharge_power_des_W * 1.1, 0 );
       this.gData[i++] = new GraphCommonData2015( "待機時電力[W]", this.maxItemCount,
             0, this.discharge_power_des_W * 1.1, 0 );
       this.gData[i++] = new GraphCommonData2015( "損失合計[W]", this.maxItemCount,
             0, this.discharge_power_des_W * 1.1, 0 );
       this.gData[i++] = new GraphCommonData2015( "出側合計電力[W]", this.maxItemCount,
             0, this.discharge_power_des_W * 1.1, 0 );
       this.gData[i++] = new GraphCommonData2015( "発電合計電力[W]", this.maxItemCount,
             0, this.discharge_power_des_W * 1.1, 0 );
       this.gData[i++] = new GraphCommonData2015( "入口有効電力[W]", this.maxItemCount,
             0, this.discharge_power_des_W * 1.1, 0 );
```

```
        this.gData[i++] = new GraphCommonData2015( "充電特性充電可能電力[W]", this.maxItemCount,
              0, this.discharge_power_des_W * 1.1, 0 );
        if( this.isAdjust2012 ){
           this.gData[i++] = new GraphCommonData2015( "調整定格放電電力[W]", this.maxItemCount,
                 0, this.discharge_power_des_W * 1.1, 0 );
        }else{
           this.gData[i++] = new GraphCommonData2015( "定格放電電力[W]", this.maxItemCount,
                 0, this.discharge_power_des_W * 1.1, 0 );
        }
        this.gData[i++] = new GraphCommonData2015( "ベース放電電力[W]", this.maxItemCount,
              0, this.discharge_power_des_W * 1.1, 0 );

        this.gData[i++] = new GraphCommonData2015( "入口有効電力[W]", this.maxItemCount,
              0, this.discharge_power_des_W * 1.1, 1 );
        this.gData[i++] = new GraphCommonData2015( "発電合計電力[W]", this.maxItemCount,
              0, this.discharge_power_des_W * 1.1, 1 );
        this.gData[i++] = new GraphCommonData2015( "出側合計電力[W]", this.maxItemCount,
              0, this.discharge_power_des_W * 1.1, 1 );
        this.gData[i++] = new GraphCommonData2015( "蓄放電電力[W]", this.maxItemCount,
              -this.charge_power_max_W * 1.1, this.discharge_power_des_W * 1.1, 1 );

        this.gData[i++] = new GraphCommonData2015( "蓄電容量[Ws]", this.maxItemCount,
              -this.capacity_max_Ws * 1.1, this.capacity_max_Ws, 2 );
        this.gData[i++] = new GraphCommonData2015( "損失合計[Ws/step]", this.maxItemCount,
              0, 100, 2 );
        this.gData[i++] = new GraphCommonData2015( "蓄電容量変化量[Ws/step]", this.maxItemCount,
              0, 100, 2 );

        this.gData[i++] = new GraphCommonData2015( "充電率[-]", this.maxItemCount, -0.1, 1.1, 3 );
        this.gData[i++] = new GraphCommonData2015( "放電停止下限充足率[-]", this.maxItemCount, -0.1, 1.1, 3 );
        this.gData[i++] = new GraphCommonData2015( "充電停止上限充足率[-]", this.maxItemCount, -0.1, 1.1, 3 );

        if( this.isAdjust2012 ){
           this.gData[i++] = new GraphCommonData2015( "調整蓄電容量[Ws]", this.maxItemCount, -0.1, 1.1, 2 );
        }

        String[] yName = { "電力[W]","電力[W]","蓄電容量等[Ws]","充電率[-]" };

        gGCJF = new GraphCommonJFrame2015( this.name+"_RB", this.gData, yName );

     }
  }

  /**
   * 下流側の有効電力[W]（負荷側）
   */
  double activePowerOut;
  /**
   * 下流側の無効電力[var]（負荷側）
   */
  double reactivePowerOut;
  /**
   * 発電側の有効電力[W]
   */
  double activePowerInG;
  /**
   * 発電側の無効電力[var]
   */
  double reactivePowerInG;
  /**
   * 上流側への有効電力[W]
   */
  double activePowerIn;
  /**
   * 上流側への無効電力[var]
   */
  double reactivePowerIn;

  /**
   *    計算時間間隔[s]
   */
  double timeInterval;// = BestTimeManager.getCurrentInterval();
```

```
public void outputs() {
   //if( super.sm == null || super.cm == null ) {
   // return;
   //}

   this.activePowerOut = 0.;
   this.reactivePowerOut = 0.;

   this.activePowerInG = 0.;
   this.reactivePowerInG = 0.;

   this.activePowerIn = 0.;
   this.reactivePowerIn = 0.;

   this.powerStandby = 0;

   this.powerGenUnconsumed = 0;
   this.powerDischargeUnconsumed = 0;

   //計算時間間隔[s]
   this.timeInterval = BestTimeManager.getCurrentInterval();

   //接続　入口状態値の取得　（電力は出口）
   this.swcIn = super.cm.getCommand(super.getConnectionNode( C_NODE_swcIn ));
   this.modIn = super.cm.getCommand(super.getConnectionNode( C_NODE_modIn ));

   if( Airmod.isPEAKCUT( this.modIn )){
      this.isPeakCut = true;
   }else{
      this.isPeakCut = false;
   }

   if( Airmod.isPEAKCUTALL( this.modIn )){
      this.isPeakCutALL = true;
   }else{
      this.isPeakCutALL = false;
   }

   if( Airmod.isPEAKSHIFT( this.modIn )){
      this.isPeakShift = true;
   }else{
      this.isPeakShift = false;
   }

   if( Airmod.isCONSTPOWER( this.modIn )){
      this.isConstPower = true;
   }else{
      this.isConstPower = false;
   }

   if( Airmod.isSMOOTH( this.modIn )){
      //出力変動抑制制御　する
      this.isCtrlPowerChangeRate = true;
   }else{
      //出力変動抑制制御　しない
      this.isCtrlPowerChangeRate = false;
   }

   if( Airmod.isLOAD_FOLLOWING_CONTROL( this.modIn )){
      //負荷追従制御　する
      this.isLoadFollowingControl = true;
   }else{
      //負荷追従制御　しない
      this.isLoadFollowingControl = false;
   }

   this.swcInECharge = super.cm.getCommand(super.getConnectionNode( C_NODE_swcInECharge ));
   this.swcInEDischarge = super.cm.getCommand(super.getConnectionNode( C_NODE_swcInEDischarge ));

   this.valInEChargeRate
```

```
= (BestValue) super.sm.getState( super.getConnectionNode( S_NODE_valInEChargeRate ));
this.valInEDischargeRate
= (BestValue) super.sm.getState( super.getConnectionNode( S_NODE_valInEDischargeRate ));

this.charge_power_valInECR
= this.discharge_power_des_W * this.valInEChargeRate.getValue();//[W]
this.discharge_power_valInEDCR
= this.discharge_power_des_W * this.valInEDischargeRate.getValue();//[W]

this.valInTargetW
= (BestValue) super.sm.getState( super.getConnectionNode( S_NODE_valInTargetW ));
this.valInOpeSetRate
= (BestValue) super.sm.getState( super.getConnectionNode( S_NODE_valInOpeSetRate ));

//出口（負荷）側　の合計
if( this.isCalcEleOut ){
   for( int i=0; i<this.eleOut.length; i++ ){
      this.eleOut[i]
      = (BestElectricity) super.sm.getState( super.getConnectionNode( S_NODE_eleOut[i] ));
      this.activePowerOut += this.eleOut[i].getActivePower();
      this.reactivePowerOut += this.eleOut[i].getReactivePower();
   }
   this.eleIn.copyAllState( this.eleOut[0] );
}

//発電側からの供給量の合計
if( this.isCalEleInGen ){
   for( int i=0; i<this.eleInGen.length; i++ ){
      this.eleInGen[i]
      = (BestElectricity) super.sm.getState( super.getConnectionNode( S_NODE_eleInGen[i] ));
      this.activePowerInG  += this.eleInGen[i].getActivePower();
      //System.out.println(" eleInGen["+i+"]=" +this.eleInGen[i].getActivePower() );
      this.reactivePowerInG += this.eleInGen[i].getReactivePower();
   }
   //発電量の移動平均値aveAdjustPowerInGを求める
   if( this.isCtrlPowerChangeRate ){
      //発電出力を調整するとき
      //this.message.append( "(C)調整最大出力[W]="+AirFormat.df_0( this.maxChangeFlowRate ) );
      this.powerOutList1.removeLast();
      this.powerOutList2.removeLast();
      this.powerOutList3.removeLast();
      this.powerOutList1.addFirst( this.activePowerInG );
      this.powerOutList2.addFirst( this.activePowerInG );
      this.powerOutList3.addFirst( this.activePowerInG );
      //
      double sumPowerInG1 = 0;
      double sumPowerInG2 = 0;
      double sumPowerInG3 = 0;
      for( int i=0; i<this.numAdjustSteps1; i++ ){
         sumPowerInG1 += this.powerOutList1.get( i );
      }
      for( int i=0; i<this.numAdjustSteps2; i++ ){
         sumPowerInG2 += this.powerOutList2.get( i );
      }
      for( int i=0; i<this.numAdjustSteps3; i++ ){
         sumPowerInG3 += this.powerOutList3.get( i );
      }
      if( this.isRecord ){
         this.message.append( "(C)sumPowerInG1=" + AirFormat.df_0( sumPowerInG1 ));
      }

      this.aveAdjustPowerInG1 = sumPowerInG1 / this.numAdjustSteps1;
      this.aveAdjustPowerInG2 = sumPowerInG2 / this.numAdjustSteps2;
      this.aveAdjustPowerInG3 = sumPowerInG3 / this.numAdjustSteps3;
      if( this.isRecord ){
         this.message.append( "(C)aveAdjustFlowRate1=" + AirFormat.df_0( this.aveAdjustPowerInG1 ));
      }
   }
}

if( this.isRecord ){
   this.message.append( "(C)sumGen="+AirFormat.df_2( activePowerInG ));
```

```
      this.message.append( "(C)sumeleOut="+AirFormat.df_2( activePowerOut ));
    }


    if( this.isCtrlPowerChangeRate ){
      //出力変動抑制制御　する
      this.target_power = this.aveAdjustPowerInG1;
      //出力変動抑制制御の場合　出口はtarget_power
      this.activePowerOut = this.target_power;
      this.reactivePowerOut = this.target_power;
    }else{
      this.target_power = this.valInTargetW.getValue();
    }

    // this.lossWattSeconds = this.chargedWattSecondsSum * this.lossRate * timeInterval;
// this.chargedWattSecondsSum -= this.lossWattSeconds;
    if( this.chargedWattSecondsSum > 0 ){
      //蓄電容量は　負の値
      this.chargedWattSecondsSum = 0;
    }
    //最大蓄電容量のチェック
    if( this.chargedWattSecondsSum < -this.capacity_max_Ws ){
      this.chargedWattSecondsSum = -this.capacity_max_Ws;
    }

    //充電率の算定①
    if( this.capacity_max_Ws == 0 ){
      this.operating_ratio = 0;
    }else{
      this.operating_ratio = this.chargedWattSecondsSum / this.capacity_max_Ws;
    }

    //■充電、放電運転時の判断
    this.isEleCharge = false;
    this.isEleDischarge = false;
    this.isEleSurplusCharge = false;

    if( this.capacity_max_Ws <=0 ){
      //20130118

    }else if( this.isConstPower ){
      //"2_発電電力平準化（一定値）制御"
      if( Airswc.isON( this.swcInECharge ) && Airswc.isON( this.swcInEDischarge ) ){
        //充放電時間帯
        if( this.activePowerInG > 0 ){
          //発電電力があるとき
          if( this.activePowerInG > this.target_power ){
            //余剰発電電力あり　充電に使う
            if( this.isRecord ){
              this.message.append( "(C)発電平準化->余剰充電");
            }
            this.isEleSurplusCharge = true;
            //System.out.println("■②平準化＝＞余剰充電");
          }
          if( this.activePowerInG < this.target_power ){
            //余剰発電電力なし　不足分を放電する
            if( this.isRecord ){
              this.message.append( "(C)発電平準化->不足放電");
            }
            this.isEleDischarge = true;
            //System.out.println("△発電平準化＞不足放電");
          }

        }else{
          //発電電力がないとき
          if( this.activePowerOut > 0 ){
            //放電
            if( this.isRecord ){
              this.message.append( "(C)発電平準化->放電");
            }
            this.isEleDischarge = true;
          }else{
```

```
            //待機
          }
        }

      }else
      if(  Airswc.isON( this.swcInECharge ) && Airmod.isE_CHARGE( this.modIn )){//充電時
        //System.out.println("■①平準化＝充電時間");
        this.isEleCharge = true;

      }else
      if(  Airswc.isON( this.swcInEDischarge ) ){//放電時
        //System.out.println("■②平準化＝放電時間");
        if( this.activePowerInG > this.target_power ){
          //余剰発電電力あり　充電に使う
          if( this.isRecord ){
            this.message.append( "(C)発電平準化->余剰充電");
          }
          this.isEleSurplusCharge = true;
          //System.out.println("■②平準化＝＞余剰充電");
        }
        if( this.activePowerInG < this.target_power ){
          //余剰発電電力なし　不足分を放電する
          if( this.isRecord ){
            this.message.append( "(C)発電平準化->不足放電");
          }
          this.isEleDischarge = true;
          //System.out.println("△発電平準化＞不足放電");
        }
      }

    }else if( this.isCtrlPowerChangeRate ){
      //"3_出力変動抑制制御"
      if( this.activePowerInG > this.target_power ){
        if( this.isRecord ){
          this.message.append( "(C)出力抑制＞余剰充電");
        }
        //this.isEleSurplusCharge = true;
        this.isEleCharge = true;
        //System.out.println("■②出力抑制＝＞余剰充電");
      }
      if( this.activePowerInG < this.target_power ){
        if( this.isRecord ){
          this.message.append( "(C)出力抑制＞不足放電");
        }
        this.isEleDischarge = true;
        //System.out.println("△出力抑制＞不足放電");
      }
    }else if( this.isLoadFollowingControl ){
      //"8_負荷追従制御
      //if( this.activePowerInG > this.target_power ){
      if( this.activePowerInG > this.activePowerOut ){
        if( this.isRecord ){
          this.message.append( "(C)負荷追従制御 発電＞負荷");
        }
        //this.isEleSurplusCharge = true;
        this.isEleCharge = true;
        //System.out.println("■②出力抑制＝＞余剰充電");
      }
      if( this.activePowerInG < this.activePowerOut ){
        if( this.isRecord ){
          this.message.append( "(C)負荷追従制御 負荷＞発電");
        }
        this.isEleDischarge = true;
        //System.out.println("△出力抑制＞不足放電");
      }
    }else if( Airswc.isON( this.swcInECharge) && Airmod.isE_CHARGE( this.modIn )){
      //充電時
      this.isEleCharge = true;
      this.isEleDischarge = false;
    }else if( Airswc.isON( this.swcInEDischarge) && !Airmod.isE_CHARGE( this.modIn )){
      //放電時
      this.isEleCharge = false;
```

```
    this.isEleDischarge = true;
}else{
    this.isEleCharge = false;
    this.isEleDischarge = false;
}


//■停止、充電、放電運転時の処理
if( Airswc.isOFF( this.swcIn ) ){
    if( this.isRecord ){
        this.message.append( "(C)停止中→待機" );
    }
    //蓄電池ではswcInがOFFは待機として損失を計算する
    this.calcStandby();

}else if( this.isEleSurplusCharge ){//出力補正時の余剰分の充電
    if( this.isRecord ){
        this.message.append( "(C)余剰分充電");
    }
    this.calcSurplusCharge();


}else if( this.isEleCharge ){//充電時
    if( this.isRecord ){
        this.message.append( "(C)充電");
    }
    this.calcCharge();

}else if( this.isEleDischarge ){//放電時
    if( this.isRecord ){
        this.message.append("(C)放電");
    }
    this.maxChargeableW2CA = 0;
    this.calcDischarge();


}else{
    if( this.isRecord ){
        this.message.append("(C)待機");
    }
    this.calcStandby();

}

if( this.chargedWatt > 0 ){
    this.eleObsCharge.setActivePower( this.chargedWatt );
    this.eleObsDischarge.setActivePower( 0 );
}else{
    this.eleObsCharge.setActivePower( 0 );
    this.eleObsDischarge.setActivePower( -this.chargedWatt );
}

//充電率の算定②
if( this.capacity_max_Ws == 0 ){
    this.operating_ratio = 0;
}else{
    this.operating_ratio = this.chargedWattSecondsSum / this.capacity_max_Ws;
}

//発熱の受渡し方法
this.heatLossW = this.lossWattSeconds / this.timeInterval * this.eleLossheat_factor;
if( this.isSystemHeatGain ){
    //建物側へ熱量[W]を渡す　すべて対流
    this.systemHeatGain.integrateHeatGain(
          this.heatLossW, 0, 0, BestTimeManager.getCurrentInterval() );
    //接続媒体で渡す＝０
    this.heaOut.setConvectiveHeatRate( 0. );
}else{
    //建物側へ熱量[W]を渡す　すべて対流＝０
    //this.systemHeatGain.integrateHeatGain(
    //    0, 0, 0, BestTimeManager.getCurrentInterval() );
    //接続媒体で渡す
```

```
    this.heaOut.setConvectiveHeatRate( this.heatLossW );
}

//ノードに設定
super.sm.setState(super.getConnectionNode( S_NODE_eleIn ), this.eleIn );
super.sm.setState(super.getConnectionNode( S_NODE_eleObsCharge ), this.eleObsCharge );
super.sm.setState(super.getConnectionNode( S_NODE_eleObsDischarge ), this.eleObsDischarge );
super.sm.setState(super.getConnectionNode( S_NODE_eleOutSmooth ), this.eleOutSmooth );
//super.sm.setState(super.getConnectionNode( S_NODE_valOutDemandele ),
//    this.valOutDemandele );

//記録
//グラフデータ追加
if( this.isGVisible ){

    int count = 0;
    if( this.isAdjust2012 ){
        count = 1;
    }
    double[] gData = new double[20+count];

/*
    this.gData[i++] = new GraphCommonData2015( "蓄放電電力[W]", this.maxItemCount,
          -this.charge_power_max_W * 1.1, this.discharge_power_des_W * 1.1, 0 );
    this.gData[i++] = new GraphCommonData2015( "電池部分蓄放電電力[W]", this.maxItemCount,
          -this.charge_power_max_W * 1.1, this.discharge_power_des_W * 1.1, 0 );
    this.gData[i++] = new GraphCommonData2015( "待機時電力[W]", this.maxItemCount,
          0, this.discharge_power_des_W * 1.1, 0 );
    this.gData[i++] = new GraphCommonData2015( "損失合計[W]", this.maxItemCount,
          0, this.discharge_power_des_W * 1.1, 0 );
    this.gData[i++] = new GraphCommonData2015( "出側合計電力[W]", this.maxItemCount,
          0, this.discharge_power_des_W * 1.1, 0 );
    this.gData[i++] = new GraphCommonData2015( "発電合計電力[W]", this.maxItemCount,
          0, this.discharge_power_des_W * 1.1, 0 );
    this.gData[i++] = new GraphCommonData2015( "入口有効電力[W]", this.maxItemCount,
          0, this.discharge_power_des_W * 1.1, 0 );
    this.gData[i++] = new GraphCommonData2015( "充電特性充電可能電力[W]", this.maxItemCount,
          0, this.discharge_power_des_W * 1.1, 0 );
    this.gData[i++] = new GraphCommonData2015( "定格放電電力[W]", this.maxItemCount,
          0, this.discharge_power_des_W * 1.1, 0 );
    this.gData[i++] = new GraphCommonData2015( "ベース放電電力[W]", this.maxItemCount,
          0, this.discharge_power_des_W * 1.1, 0 );

    this.gData[i++] = new GraphCommonData2015( "蓄電容量[Ws]", this.maxItemCount,
          -this.capacity_max_Ws * 1.1, this.capacity_max_Ws, 1 );
    this.gData[i++] = new GraphCommonData2015( "損失合計[Ws/step]", this.maxItemCount,
          0, 100, 1 );
    this.gData[i++] = new GraphCommonData2015( "蓄電容量変化量[Ws/step]", this.maxItemCount,
          0, 100, 1 );

    this.gData[i++] = new GraphCommonData2015( "充電率[-]", this.maxItemCount, -0.1, 1.1, 2 );
    this.gData[i++] = new GraphCommonData2015( "放電停止下限充足率[-]", this.maxItemCount, -0.1, 1.1, 2 );
    this.gData[i++] = new GraphCommonData2015( "充電停止上限充足率[-]", this.maxItemCount, -0.1, 1.1, 2 );
*/

    int i=0;
    gData[i++] = this.chargedWatt;
    gData[i++] = this.b_chargedWatt;
    gData[i++] = this.powerStandby;
    gData[i++] = this.lossWattSeconds / this.timeInterval;
    gData[i++] = this.activePowerOut;
    gData[i++] = this.activePowerInG;
    gData[i++] = this.eleIn.getActivePower();
    gData[i++] = this.maxChargeableW2CA;
    gData[i++] = this.discharge_power_des_W;
    gData[i++] = -this.discharge_power_base_W;

    gData[i++] = this.eleIn.getActivePower();
    gData[i++] = this.activePowerInG;
    gData[i++] = this.activePowerOut;
    gData[i++] = this.chargedWatt;
```

```
        gData[i++] = this.chargedWattSecondsSum;
        gData[i++] = this.lossWattSeconds;
        gData[i++] = this.chargedWattSeconds;

        gData[i++] = -this.operating_ratio;
        gData[i++] = this.min_operating_ratio;
        gData[i++] = this.max_operating_ratio;

        if( this.isAdjust2012 ){
           gData[i++] = -this.capacity_des_Ws;
        }

        this.gGCJF.addData( BestTimeManager.getDateWeatherTime(), gData );

     }

     //記録
     if( this.isRecord  && super.rm != null ){
        this.record();
     }

     message.setLength(0);
  }


  /**
   * 記録
   */
  private void record(){

     if( this.isRecordALL || CheckPrintModule.isPrintMessage ){
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
              RECORD_message, this.name, this.message.toString() );
     }

     //if( this.isRecordALL || CheckPrintModule.isPrintEnergy ){
     //}

     //if( this.isRecordALL || CheckPrintModule.isPrintLoad ){
     //}

     if( this.isRecordALL || CheckPrintModule.isPrintStateOut ){
        //出口
        if( this.isCalcEleOut ){
           for( int i=0; i<eleOut.length; i++ )
           {
              super.rm.setRecord( super.getConnectionNode( R_NODE ),
                    RECORD_A_eleOut[i], this.name,
                    AirFormat.df_2( this.eleOut[i].getActivePower() ));
              super.rm.setRecord( super.getConnectionNode( R_NODE ),
                    RECORD_R_eleOut[i], this.name,
                    AirFormat.df_2( this.eleOut[i].getReactivePower() ));
           }
        }
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
              RECORD_A_eleOutSum, this.name, AirFormat.df_2( this.activePowerOut ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
              RECORD_smoothPowerOutG, this.name, AirFormat.df_2( this.eleOutSmooth.getActivePower() ));
     }

     if( this.isRecordALL || CheckPrintModule.isPrintStateWy ){
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
              RECORD_chargedWatt, this.name, AirFormat.df_2( this.chargedWatt ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
              RECORD_b_chargedWatt, this.name, AirFormat.df_2( this.b_chargedWatt ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
              RECORD_chargedWS, this.name, AirFormat.df_2( this.chargedWattSeconds ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
              RECORD_chargedWSSum, this.name, AirFormat.df_2( this.chargedWattSecondsSum ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
              RECORD_lossWSs, this.name, AirFormat.df_2( this.lossWattSeconds ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
```

```
            RECORD_lossW, this.name, AirFormat.df_2( this.lossWattSeconds / this.timeInterval ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_chargedRate, this.name, AirFormat.df_2( -this.operating_ratio ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_min_operating_ratio, this.name, AirFormat.df_2( this.min_operating_ratio ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_max_operating_ratio, this.name, AirFormat.df_2( this.max_operating_ratio ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_powerStandby, this.name, AirFormat.df_2( this.powerStandby ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_avePowerOutG1, this.name, AirFormat.df_2( this.aveAdjustPowerInG1 ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_avePowerOutG2, this.name, AirFormat.df_2( this.aveAdjustPowerInG2 ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_avePowerOutG3, this.name, AirFormat.df_2( this.aveAdjustPowerInG3 ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_davePowerOutG1, this.name,
            AirFormat.df_2( this.activePowerInG - this.aveAdjustPowerInG1 ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_davePowerOutG2, this.name,
            AirFormat.df_2( this.activePowerInG - this.aveAdjustPowerInG2 ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_davePowerOutG3, this.name,
            AirFormat.df_2( this.activePowerInG - this.aveAdjustPowerInG3 ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_maxChargeableW2CA, this.name, AirFormat.df_2( this.maxChargeableW2CA ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_discharge_power_des_W, this.name, AirFormat.df_2( this.discharge_power_des_W ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_discharge_power_base_W, this.name, AirFormat.df_2( -this.discharge_power_base_W ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_heatLossW, this.name, AirFormat.df_2( this.heatLossW ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_powerGenUnconsumed, this.name, AirFormat.df_2( this.powerGenUnconsumed ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_powerDischargeUnconsumed, this.name, AirFormat.df_2( this.powerDischargeUnconsumed ));
    }

    if( this.isRecordALL || CheckPrintModule.isPrintStateIn ){
        //入口
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_A_eleIn, this.name,
            AirFormat.df_2( this.eleIn.getActivePower() ));
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_R_eleIn, this.name,
            AirFormat.df_2( this.eleIn.getReactivePower() ));
        //発電入口
        if( this.isCalEleInGen ){
            for( int i=0; i<eleInGen.length; i++ )
            {
                super.rm.setRecord( super.getConnectionNode( R_NODE ),
                    RECORD_A_eleInG[i], this.name,
                    AirFormat.df_2( this.eleInGen[i].getActivePower() ));
                super.rm.setRecord( super.getConnectionNode( R_NODE ),
                    RECORD_R_eleInG[i], this.name,
                    AirFormat.df_2( this.eleInGen[i].getReactivePower() ));
            }
        }
        super.rm.setRecord( super.getConnectionNode( R_NODE ),
            RECORD_A_eleInGSum, this.name,
            AirFormat.df_2( this.activePowerInG ));
    }

    if( this.isRecordALL || CheckPrintModule.isPrintAdjust ){
        //
        if( this.isAdjust2012 ){
            //記録ノードに室外機調整能力を設定
            super.rm.setRecord(super.getConnectionNode(R_NODE),
                RECORD_capacity_des_Ws_Adjust, this.name,
                -this.capacity_des_Ws );//"RB_調整蓄電容量#Ws#-"
            super.rm.setRecord(super.getConnectionNode(R_NODE),
                RECORD_discharge_power_des_W_Adjust, this.name,
```

```
                this.discharge_power_des_W );//"RB_調整放電能力#W#-"
        }
    }

}



public void update() {

}


/**
 * 余剰電力の充電計算
 */
private void calcSurplusCharge(){

    //充電可能量[W]  2CA特性より
    //double maxChargeableW2CA
    //= this.capacity_max_Ws * this.getChargeRatio( this.timeInterval, -this.operating_ratio, this.max_operating_ratio,
this.chargingModel );//充電可能量[Ws]
    //double maxChargeableW2CA;
    if( this.isConstPower ){
        maxChargeableW2CA
        = this.capacity_max_Ws
        * this.getChargeRatio( timeInterval, -this.operating_ratio,
              this.max_operating_ratio * this.maintenance_factor, this.chargingModel );//充電可能量[Ws]
    }else{
        maxChargeableW2CA
        = this.capacity_max_Ws
        * this.getChargeRatio( timeInterval, -this.operating_ratio,
              this.max_operating_ratio * this.maintenance_factor, this.chargingModel );//充電可能量[Ws]
    }
    maxChargeableW2CA /= this.timeInterval;//[Ws]-->[W]

    //充電電力係数からの最大充電電力と比較
    double maxChargeableWnow;

    maxChargeableWnow
    = ( (maxChargeableW2CA / this.charging_factor) < this.charge_power_max_W )?
          (maxChargeableW2CA / this.charging_factor): this.charge_power_max_W;//ロス込で比較

    //スケジュール運転制御の値が>0の時
    if( this.charge_power_valInECR > 0 ){
        maxChargeableWnow
        = ( maxChargeableWnow < this.charge_power_valInECR )?
              maxChargeableWnow: this.charge_power_valInECR;
    }

    //charge_power_valInECRとの大小チェック
    //if( !this.isConstPower ){
    // maxChargeableWnow = ( maxChargeableWnow < this.charge_power_valInECR )? maxChargeableWnow:
this.charge_power_valInECR;
    //}

    //変換効率を考慮する
    //maxChargeableWnow /= this.charging_factor;  //20140725間違い

    //充電に回せる余剰電力
    //double surplusW = activePowerInG - Math.min( this.target_power, activePowerOut );
    double surplusW
    = this.activePowerInG - ( this.target_power + this.des_powerStandby_charge );//20170608充電時の待機時消費
    //double surplusWre = reactivePowerInG - Math.min( this.target_power, reactivePowerOut );
    //double surplusWre = this.reactivePowerInG - this.target_power;
    double surplusWre
    = this.reactivePowerInG - ( this.target_power + this.des_powerStandby_charge )
          * this.reactivePowerInG / this.activePowerInG;//20170530  //20170608充電時の待機時消費

    //System.out.println( "maxChargeableW2CA="+maxChargeableW2CA+" charge_power_max="+this.charge_power_max+"
maxChargeableWnow="+maxChargeableWnow);
    //余剰発電電力[w]
```

```
        double amariW   = surplusW - maxChargeableWnow;
        //double amariWre = surplusWre - maxChargeableWnow;
        double amariWre = surplusWre - maxChargeableWnow * this.reactivePowerInG / this.activePowerInG;//20170530;

        //蓄電池システムとして充電量[W]
        double chargedW = 0;


        //上流からの供給必要量

        if( amariW > 0 ){
           //最大パワーで充電可能
           chargedW = maxChargeableWnow;

           if( this.isCtrlPowerChangeRate ){
              //出力変動抑制制御　は逆潮流に関係なく上流＝０
              this.activePowerIn = 0;
              this.reactivePowerIn = 0;
              //未消費発電電力　20170531
              if( this.activePowerOut < this.target_power + this.des_powerStandby_charge ){
                 this.powerGenUnconsumed
                 = amariW + this.target_power + this.des_powerStandby_charge - this.activePowerOut;//20170608充電時の待機時
消費
              }else{
                 this.powerGenUnconsumed = amariW;
              }
           }else if( this.isBackwardFlow ){
              //逆潮流あり
              this.activePowerIn = -amariW;
              this.reactivePowerIn = -amariWre;
              //未消費発電電力　20170531
              if( this.activePowerOut < this.target_power + this.des_powerStandby_charge ){
                 this.powerGenUnconsumed
                 = this.target_power + this.des_powerStandby_charge - this.activePowerOut;//20170608充電時の待機時消費
              }else{
                 this.powerGenUnconsumed = 0;
              }
           }else{
              //逆潮流なし
              this.activePowerIn   = 0;
              this.reactivePowerIn = 0;
              //未消費発電電力　20170531
              if( this.activePowerOut < this.target_power + this.des_powerStandby_charge ){
                 this.powerGenUnconsumed
                 = amariW  + this.target_power + this.des_powerStandby_charge - this.activePowerOut;//20170608充電時の待機
時消費
              }else{
                 this.powerGenUnconsumed = amariW;
              }
           }
        }else{
           //部分パワーで充電
           boolean isComCharging = false;
           if( isComCharging ){
              //系統から充電あり
              chargedW = maxChargeableWnow;
              this.activePowerIn = this.activePowerOut - amariW + this.des_powerStandby_charge;//20170608
              this.reactivePowerIn = this.reactivePowerOut - amariWre + this.des_powerStandby_charge
                    * Math.pow( 1/this.powerFactorStandby/this.powerFactorStandby - 1., 0.5 );//20170608
           }else{
              //系統からは充電しない
              chargedW = surplusW;
              this.activePowerIn = this.activePowerOut - this.activePowerInG + surplusW;
              this.reactivePowerIn = this.reactivePowerOut - this.reactivePowerInG + surplusWre;//20170530

              //System.out.println( "************** activePowerOut= "+activePowerOut + "
activePowerInG="+this.activePowerInG +" surPlusW="+surplusW );
              //System.out.println( "************** reactivePowerOut= "+reactivePowerOut + "
reactivePowerInG="+this.reactivePowerInG +" surPlusWre="+surplusWre );
              //System.out.println( "************** activePowerIn= "+activePowerIn + " reactiveP="+this.reactivePowerIn );

              if( !this.isBackwardFlow ){
```

```
            if( this.activePowerIn < 0 ){
               this.powerGenUnconsumed = -this.activePowerIn;
            }
            //逆潮流なし
            this.activePowerIn   = 0;
            this.reactivePowerIn = 0;
         }
      }
   }

   //待機時消費　充電時 //20170608
   this.powerStandby = this.des_powerStandby_charge;

   this.chargedWatt = chargedW + this.des_powerStandby_charge;//20170608 充電時の待機時消費を加える
   this.b_chargedWatt = chargedW * this.charging_factor;

   //this.lossWattSeconds = chargedW * this.timeInterval * ( 1. - this.charging_factor ) / this.charging_factor;
   //this.lossWattSeconds = chargedW * this.timeInterval * ( 1. - this.charging_factor );
   this.lossWattSeconds = ( this.chargedWatt - this.b_chargedWatt ) * this.timeInterval;

   this.chargedWattSeconds = -this.b_chargedWatt * this.timeInterval;
   this.chargedWattSecondsSum += this.chargedWattSeconds;

   if( this.reactivePowerIn < 0 ){
      this.reactivePowerIn = 0;
   }

   if( chargedW == 0 ){
      //充電時間帯に充電が０の時は待機電力の計算を行う
      this.calcStandby();
   }

   this.eleIn.setActivePower( this.activePowerIn );
   this.eleIn.setReactivePower( this.reactivePowerIn );

}


/**
 * 充電計算
 */
private void calcCharge(){
   // 集計
   //入口のactivePower,reactivePower
   //System.out.println( "eleOut length=" + this.eleOut.length );

   //充電可能量[W]  2CA特性より
   //double maxChargeableW2CA;
   if( this.isConstPower ){
      maxChargeableW2CA
      = this.capacity_max_Ws
      * this.getChargeRatio( timeInterval, -this.operating_ratio,
            this.valInOpeSetRate.getValue() * this.maintenance_factor, this.chargingModel );//充電可能量[Ws]
      this.message.append( "valInOpeSetRate="+valInOpeSetRate.getValue());
   }else{
      maxChargeableW2CA
      = this.capacity_max_Ws
      * this.getChargeRatio( timeInterval, -this.operating_ratio,
            this.max_operating_ratio * this.maintenance_factor, this.chargingModel );//充電可能量[Ws]
      this.message.append( "max_operating_ratio="+max_operating_ratio);
   }
   maxChargeableW2CA /= this.timeInterval;//[Ws]-->[W]
   this.message.append( "maxChargeableW2CA[W]="+maxChargeableW2CA);

   //充電電力係数からの最大充電電力と比較
   double maxChargeableWnow;

   maxChargeableWnow = ( ( maxChargeableW2CA / this.charging_factor ) < this.charge_power_max_W )?
         ( maxChargeableW2CA / this.charging_factor ): this.charge_power_max_W;//ロス込で比較

   //スケジュール運転制御の値が>0の時
   if( this.charge_power_valInECR > 0 ){
```

```
            maxChargeableWnow
            = ( maxChargeableWnow < this.charge_power_valInECR )? maxChargeableWnow: this.charge_power_valInECR;
        }

        //charge_power_valInECRとの大小チェック
        //if( !this.isConstPower ){
        // maxChargeableWnow = ( maxChargeableWnow < this.charge_power_valInECR )? maxChargeableWnow:
this.charge_power_valInECR;
        //}

        //変換効率を考慮する
        //maxChargeableWnow /= this.charging_factor;  //20140725間違い

        if( this.isPeakCutALL ){
            //this.activePowerOut - this.activePowerInG
            //maxChargeableWnow = ( maxChargeableWnow > this.target_power )? this.target_power: maxChargeableWnow;
            //maxChargeableWnow
            //= ( maxChargeableWnow > ( this.target_power - this.activePowerOut + this.activePowerInG ) )?
            //    ( this.target_power - this.activePowerOut + this.activePowerInG ): maxChargeableWnow;
            maxChargeableWnow
            = ( maxChargeableWnow
            > ( this.target_power - this.activePowerOut + this.activePowerInG - this.des_powerStandby_charge) )?
                ( this.target_power - this.activePowerOut + this.activePowerInG - this.des_powerStandby_charge )
                : maxChargeableWnow;
        }

        //System.out.println( "maxChargeableW2CA="+maxChargeableW2CA+" charge_power_max="+this.charge_power_max+"
maxChargeableWnow="+maxChargeableWnow);
        //余剰発電電力[w]
        double amariW   = this.activePowerInG   - maxChargeableWnow;
    // double amariWre = this.reactivePowerInG - maxChargeableWnow;
        double amariWre
        = this.reactivePowerInG - maxChargeableWnow * this.reactivePowerInG / this.activePowerInG;//20170530;

        //蓄電池システムとしてみた充電量[W]
        double chargedW = 0;


        //上流からの供給必要量
        if( this.isCtrlPowerChangeRate ){
            //出力変動抑制制御の場合は  上流からの供給はない
            this.activePowerIn = 0;
            this.reactivePowerIn = 0;

            chargedW = ( this.activePowerInG - this.target_power - this.des_powerStandby_charge ) / this.charging_factor;

            this.eleOutSmooth.setActivePower( this.target_power );

        }else if( amariW > 0 ){
            //発電の余りがあり→充電はすべて発電から
            chargedW = maxChargeableWnow;

            if( this.isBackwardFlow ){
                //逆潮流あり  余剰分を負荷処理に使いさらに余った分は上位へ逆潮する
                this.activePowerIn   = this.activePowerOut - amariW + this.des_powerStandby_charge;
                this.reactivePowerIn = this.reactivePowerOut - amariWre + this.des_powerStandby_charge
                      * Math.pow( 1/this.powerFactorStandby/this.powerFactorStandby - 1., 0.5 );
            }else{
                //逆潮流なし  余剰分を負荷処理に使いさらに余った分は未消費処理する
                this.activePowerIn   = this.activePowerOut - amariW + this.des_powerStandby_charge ;
                this.reactivePowerIn = this.reactivePowerOut - amariWre + this.des_powerStandby_charge
                      * Math.pow( 1/this.powerFactorStandby/this.powerFactorStandby - 1., 0.5 );
                if( this.activePowerIn < 0 ){
                   this.powerGenUnconsumed = -this.activePowerIn;
                   this.activePowerIn = 0;
                   this.reactivePowerIn = 0;
                }
            }
        }else{
            //発電の余りがなし
            boolean isComCharging = true;
            if( isComCharging ){
```

```
            //系統から充電あり
            chargedW = maxChargeableWnow;
            this.activePowerIn = this.activePowerOut - amariW + this.des_powerStandby_charge;
            this.reactivePowerIn = this.reactivePowerOut - amariWre + this.des_powerStandby_charge
                  * Math.pow( 1/this.powerFactorStandby/this.powerFactorStandby - 1., 0.5 );
         }else{
            //系統からは充電しない
            chargedW = this.activePowerInG;

            this.activePowerIn = this.activePowerOut + this.des_powerStandby_charge;
            this.reactivePowerIn = this.reactivePowerOut + this.des_powerStandby_charge
                  * Math.pow( 1/this.powerFactorStandby/this.powerFactorStandby - 1., 0.5 );
         }
      }

      //待機時消費　充電時 //20170608
      this.powerStandby = this.des_powerStandby_charge;

      this.chargedWatt = chargedW + this.des_powerStandby_charge;//20170608 充電時の待機時消費を加える
      this.b_chargedWatt = chargedW * this.charging_factor;

      //this.lossWattSeconds = chargedW * this.timeInterval * ( 1. - this.charging_factor ) / this.charging_factor;
      //this.lossWattSeconds = chargedW * this.timeInterval * ( 1. - this.charging_factor );
      this.lossWattSeconds = ( this.chargedWatt - this.b_chargedWatt ) * this.timeInterval;

      this.chargedWattSeconds = -this.b_chargedWatt * this.timeInterval;
      this.chargedWattSecondsSum += this.chargedWattSeconds;

      if( this.reactivePowerIn < 0 ){
         this.reactivePowerIn = 0;
      }

      if( chargedW == 0 ){
         //充電時間帯に充電が０の時は待機電力の計算を行う
         this.calcStandby();
      }

      this.eleIn.setActivePower( this.activePowerIn );
      this.eleIn.setReactivePower( this.reactivePowerIn );

   }


   /**
    * 放電時の計算
    */
   private void calcDischarge(){

      //蓄電池システムとしてみた放電量[W]
      double dischargedW = 0;//有効
      double dischargedWre = 0;//無効20170530

      //最大放電能力(有効)
      double maxdischargeW;

      //放電下限充電率による利用可能量による制限
      maxdischargeW
      = ( -this.chargedWattSecondsSum - this.chargedWattSecondsSum_min_Ws ) / this.timeInterval
      * this.discharging_factor;
      //System.out.println( "①maxdischargeW="+maxdischargeW);

      if( maxdischargeW > this.discharge_power_max_W ){
         //定格放電の上限
//       this.calcAdjustMaxDischargePower( maxdischargeW );

          maxdischargeW = this.discharge_power_max_W;
         //System.out.println( "②maxdischargeW="+maxdischargeW);
      }

      double valZero = maxdischargeW * this.valInZeroRate.getValue();
      //System.out.println( "①valZero="+valZero +" valInZeroRate.getValue()"+valInZeroRate.getValue() + "
/="+(1./valInZeroRate.getValue()));
```

```
        //デマンド放電量[W]
        double demand_dischargeW = 0;

        if( this.isPeakShift ){//"0_ピークシフト制御"
           if( this.isRecord ){
              this.message.append("_ピークシフト");
           }

           //System.out.println( "③maxdischargeW="+maxdischargeW + "
discharge_power_max_shift_W="+discharge_power_max_shift_W);
           if( this.isPeakCut || this.isPeakCutALL ){//"1_ピークカット制御"
              if( this.isRecord ){
                 this.message.append("_ピークカット_target="+this.target_power );
              }
              if( this.activePowerOut - this.activePowerInG - this.target_power > 0 ){//20151007
                 //目標値を超える時　ピークカット
                 this.calcAdjust( this.activePowerOut - this.activePowerInG - this.target_power,  maxdischargeW );
                 demand_dischargeW
                 = ( this.activePowerOut - this.activePowerInG - this.target_power < maxdischargeW )?
                       this.activePowerOut - this.activePowerInG - this.target_power: maxdischargeW;
                 demand_dischargeW
                 = ( this.activePowerOut - this.activePowerInG - this.target_power < maxdischargeW )?
                       this.activePowerOut - this.activePowerInG - this.target_power: maxdischargeW;
                 demand_dischargeW = ( demand_dischargeW < this.discharge_power_base_W )?
                       this.discharge_power_base_W: demand_dischargeW;
              }else if(  this.activePowerOut + this.des_powerStandby - this.activePowerInG - this.target_power > 0 ){
                 //目標値を超える時　ピークカット
                 this.calcAdjust( this.activePowerOut + this.des_powerStandby - this.activePowerInG - this.target_power,
maxdischargeW );
                 demand_dischargeW
                 = ( this.activePowerOut + this.des_powerStandby - this.activePowerInG - this.target_power <
maxdischargeW )?
                       this.activePowerOut + this.des_powerStandby - this.activePowerInG - this.target_power:
maxdischargeW;
                 demand_dischargeW = ( demand_dischargeW < this.discharge_power_base_W )?
                       this.discharge_power_base_W: demand_dischargeW;
              }else{
                 //目標値以下の時はベース放電電力　ピークシフト
                 demand_dischargeW = ( this.discharge_power_base_W < maxdischargeW )?
                       this.discharge_power_base_W: maxdischargeW;

              }
              demand_dischargeW = ( this.activePowerOut - this.activePowerInG < demand_dischargeW )?
                    this.activePowerOut- this.activePowerInG: demand_dischargeW;//[W]
              //System.out.println( "④maxdischargeW="+maxdischargeW);
              demand_dischargeW = ( demand_dischargeW > maxdischargeW )? maxdischargeW: demand_dischargeW;//[W]
              if( demand_dischargeW < 0 ){
                 demand_dischargeW = 0;
              }
           }else{
              //ピークシフトのデマンド
              this.calcAdjust( this.discharge_power_base_W,  maxdischargeW );
              maxdischargeW = ( this.discharge_power_base_W < maxdischargeW )?
                    this.discharge_power_base_W: maxdischargeW;
              demand_dischargeW = ( this.activePowerOut - this.activePowerInG < maxdischargeW )?
                    this.activePowerOut- this.activePowerInG: maxdischargeW;//[W]
              if( demand_dischargeW < 0 ){
                 demand_dischargeW = 0;
              }


           }

           //System.out.println( "⑤demand_dischargeW="+demand_dischargeW);

           //使い切り
           if( valZero > 0 ){
              if( valZero > demand_dischargeW ){
                 demand_dischargeW = Math.min( valZero, Math.max( 0, this.activePowerOut - this.activePowerInG ) );
                 maxdischargeW = demand_dischargeW;
```

```
            //System.out.println( "⑦valZero="+valZero+" demand="+(this.activePowerOut - this.activePowerInG));
        }
    }

    //スケジュール運転制御の値が>0の時　　スケジュール運転制御はピークシフト時のみ適用する
    if( this.discharge_power_valInEDCR > 0 ){
        demand_dischargeW = ( demand_dischargeW < this.discharge_power_valInEDCR )?
              demand_dischargeW: this.discharge_power_valInEDCR;
    }

    //このピークシフトでは、充電となることはない
    if( demand_dischargeW < 0 ){
        demand_dischargeW = 0;
    }

}else if( this.isPeakCut || this.isPeakCutALL ){//"1_ピークカット制御"　ピークカット単独の場合
    if( this.isRecord ){
        this.message.append("_ピークカット_target="+this.target_power );
    }

    if( this.activePowerOut - this.activePowerInG - this.target_power > 0.){
        demand_dischargeW
        = this.activePowerOut - this.activePowerInG - this.target_power;//20151007

    }else if( this.activePowerOut + this.des_powerStandby - this.activePowerInG - this.target_power > 0. ){
        demand_dischargeW
        = this.activePowerOut + this.des_powerStandby - this.activePowerInG - this.target_power;//20151007

    }else{
        demand_dischargeW = 0;//20151007

    }
    this.calcAdjust( demand_dischargeW,  maxdischargeW );
    demand_dischargeW = ( demand_dischargeW > maxdischargeW )? maxdischargeW: demand_dischargeW;//[W]
    if( demand_dischargeW < 0 ){
        demand_dischargeW = 0;
    }

}else if( this.isConstPower ){//"2_出力補正制御"
    if( this.isRecord ){
        this.message.append("_発電電力平準化");
    }
    demand_dischargeW = this.target_power - this.activePowerInG;//[W]
    this.calcAdjust( demand_dischargeW,  maxdischargeW );
    //System.out.println( "⑧demand_dischargeW="+demand_dischargeW);

}else if( this.isCtrlPowerChangeRate ){//"2_出力補正制御"
    if( this.isRecord ){
        this.message.append("_出力変動抑制　検討中");
    }
    demand_dischargeW = this.target_power - this.activePowerInG;//[W]
    //demand_dischargeW = this.target_power - this.activePowerInG;//[W]

    this.eleOutSmooth.setActivePower( this.target_power );

}else{

}

if( this.isRecord ){
    this.message.append("_dmande="+ AirFormat.df_2( demand_dischargeW ));
}

if( demand_dischargeW > maxdischargeW ){
    //放電量が不足
//    this.calcAdjust( demand_dischargeW,  maxdischargeW );
    //
    dischargedW = maxdischargeW;
    dischargedWre = maxdischargeW * Math.pow( 1/this.powerFactor/this.powerFactor - 1., 0.5 );//20170530
    this.activePowerIn   = this.activePowerOut - activePowerInG - dischargedW;
    this.reactivePowerIn = this.reactivePowerOut- this.reactivePowerInG - dischargedWre;//20170530
}else{
```

```
            //放電量が足りている
            dischargedW = demand_dischargeW;
            dischargedWre = demand_dischargeW * Math.pow( 1/this.powerFactor/this.powerFactor - 1., 0.5 );//20170530
            this.activePowerIn   = (  this.activePowerOut -   this.activePowerInG - dischargedW > 0 )?
                  this.activePowerOut -   this.activePowerInG - dischargedW: 0;
            this.reactivePowerIn = (this.reactivePowerOut - this.reactivePowerInG - dischargedWre > 0 )?
                  this.reactivePowerOut - this.reactivePowerInG - dischargedWre: 0;//20170530
         }
         if( this.isRecord ){
            this.message.append("_discharge="+ AirFormat.df_2( dischargedW ));
         }

         this.chargedWatt = -dischargedW;
         this.b_chargedWatt = -dischargedW / this.discharging_factor;

         //this.lossWattSeconds = dischargedW * this.timeInterval * ( 1. - this.discharging_factor ) /
this.discharging_factor;
         //this.lossWattSeconds = dischargedW * this.timeInterval * ( 1. - this.discharging_factor );
         this.lossWattSeconds = ( this.chargedWatt - this.b_chargedWatt ) * this.timeInterval;

         this.chargedWattSeconds = -this.b_chargedWatt * this.timeInterval;
         this.chargedWattSecondsSum += this.chargedWattSeconds;
         //System.out.println( "demand_dischargeW ="+demand_dischargeW+" dischargedW="+dischargedW+"
activePowerIn="+activePowerIn+" chargedWattSecondsSum="+ chargedWattSecondsSum);

         if( this.activePowerIn < 0 ){
            this.activePowerIn = 0;
         }
         if( this.reactivePowerIn < 0 ){
            this.reactivePowerIn = 0;
         }

         if( this.activePowerOut < dischargedW ){
            this.powerDischargeUnconsumed = dischargedW - this.activePowerOut + this.activePowerInG;
         }

         this.eleIn.setActivePower( this.activePowerIn );
         this.eleIn.setReactivePower( this.reactivePowerIn );

         if( dischargedW == 0 ){
            //放電時間帯に放電が０の時は待機電力の計算を行う
            this.calcStandby();
         }
      }


      /**
       * 待機時の処理
       */
      private void calcStandby(){

         this.maxChargeableW2CA = 0;

         this.powerStandby = this.des_powerStandby;

         this.activePowerIn   = this.activePowerOut - this.activePowerInG + this.powerStandby;
         this.reactivePowerIn = this.reactivePowerOut - this.reactivePowerInG + this.powerStandby
               * Math.pow( 1/this.powerFactorStandby/this.powerFactorStandby - 1., 0.5 );//20170530;

         if( this.activePowerIn < 0 ){
            this.activePowerIn = 0;
         }
         if( this.reactivePowerIn < 0 ){
            this.reactivePowerIn = 0;
         }

         this.chargedWatt = this.powerStandby;//20140726待機時電力を充電電力として扱う
         this.b_chargedWatt = 0.;
         this.chargedWattSeconds = 0.;

         this.lossWattSeconds = this.powerStandby * this.timeInterval;
```

```
        this.eleIn.setActivePower( this.activePowerIn );
        this.eleIn.setReactivePower( this.reactivePowerIn );

    }


/* private double getMaxDischargeW(){
        return 1.;
    }
*/
    /**
     * @param timeInterval  計算時間間隔[s]
     * @param ratio 充電率[-]
     * @param maxRatio 充電停止最大充電率[-]
     * @return 充電率ratioの時の計算ステップ時間の可能充電率（充電率の増加量）[-]
     */
    private double getChargeRatio( double timeInterval,  double ratio, double maxRatio, int ChargingModel ){
        double h0 = 0;
        double h1 = timeInterval / 3600.;
        double efficiency = 1.0;
        double r=0;

        if( ratio >= 1. ){
            r = 0.;
        }else{
            switch( ChargingModel ){
            case RBTYPE_LIB://リチウムイオン電池
                //近似式
                //LIM50E 2CA
                //2CAでの充電時間と充電率の特性近似式
                //  ratio = 1 - 1 / (1+2* hr ) / (1+2* hr )
                //    ratio  充電率[0～1]
                //    hr  充電時間[hour]

                //特性近似式における充電率の時刻を求める
                h0 = 0.5 / Math.pow( 1.- ratio, 0.5 ) - 0.5;
                //計算ステップ時間を加算する
                h1 += h0;
                //近似式で計算ステップ時間経過後の充電率を求める
                r = 1. - 1. / Math.pow(( 1.+ 2. * h1), 2.);
                //上限チェック
                r = ( r < maxRatio)? r: maxRatio;

                //充電率の可能増加量[-]
                r = ( r - ratio ) / efficiency;
                break;
            case RBTYPE_PBB://鉛
                //20141016
                //鉛電池は０～１００%までリニアに充電可能とする
                //最大充電可能電力は定格放電電力
                //
                //最大蓄電容量  capacity_max_Ws
                //定格放電電力  discharge_power_max_W
                //充電特性
                // 充電率  r = ratio +  定格放電電力×計算時間間隔／最大蓄電容量
                //    ratio  現在の充電率[0～1]
                //計算ステップ時間経過後の充電率を求める
                r = ratio + discharge_power_max_W * timeInterval / capacity_max_Ws;
                //上限チェック
                r = ( r < maxRatio)? r: maxRatio;

                //充電率の可能増加量[-]
                r = ( r - ratio ) / efficiency;
                break;
            case RBTYPE_NAS://NAS
                //20141016
                //NAS電池は０～１００%までリニアに充電可能とする
                //最大充電可能電力は定格放電電力
                //
                //最大蓄電容量  capacity_max_Ws
                //定格放電電力  discharge_power_max_W
                //充電特性
```

```
            // 充電率  r = ratio +  定格放電電力×計算時間間隔／最大蓄電容量
            //    ratio  現在の充電率[0～1]
            //計算ステップ時間経過後の充電率を求める
            r = ratio + discharge_power_max_W * timeInterval / capacity_max_Ws;
            //上限チェック
            r = ( r < maxRatio)? r: maxRatio;

            //充電率の可能増加量[-]
            r = ( r - ratio ) / efficiency;
            //System.out.println("r="+r+" ratio="+ratio+" discharge_power_max_W="+discharge_power_max_W+"
capacity_max_Ws="+capacity_max_Ws+" h1="+h1);
            break;
         case RBTYPE_FLI://時間比例
            r = maxRatio - ratio;
            r = ( r < this.charge_hour_factor )? r : this.charge_hour_factor;
            break;
         default:
            r = 0;
            break;
      }
      }

      if( r < 0 ){
         r = 0;//20140522
      }
      //System.out.println( "ratio="+ratio+" r="+r);

      return r;
   }


   public Object viewInternal(TestCommand cmd) {
      // とりあえずクラス変数のリストを返します。
      java.util.List<Object> result = new java.util.ArrayList<Object>();

      //接続ノード
      result.add(super.sm);
      result.add(super.cm);

      //外部定義項目
      result.add(capacity_max_Ws);
      result.add( this.eleLoss.getActivePower() );
      result.add( this.airIn.getTempDB());
      result.add( this.operating_ratio );
      result.add( this.maintenance_factor );
      if( eleOut != null ){
         result.add( eleOut );
      }

      return result;
   }
}
```

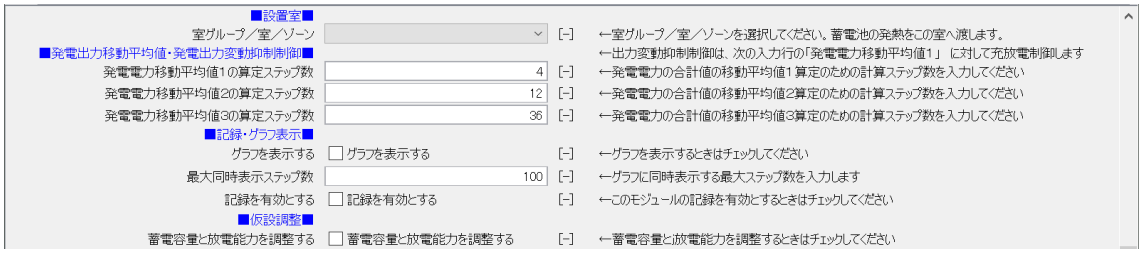
「蓄電池 G2018」（場所：設備 2015／電気設備 2015／）

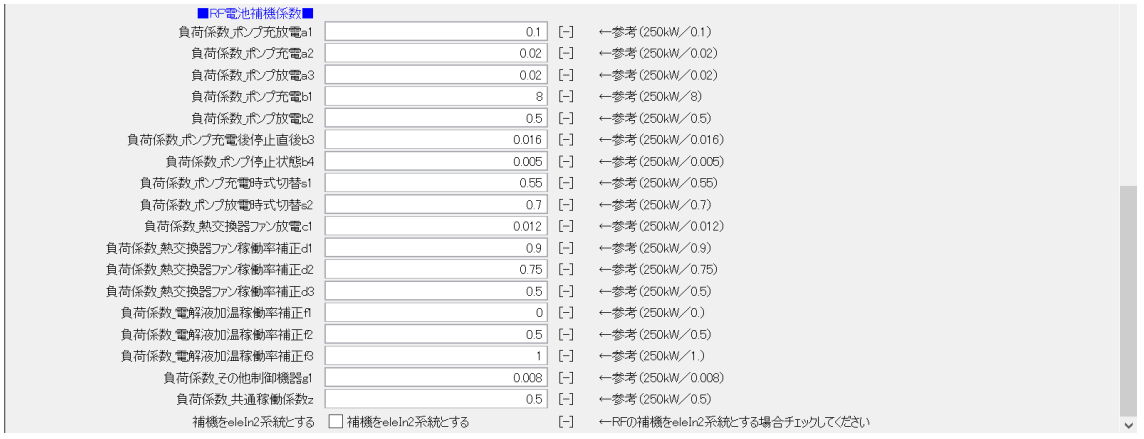
2021.12.6 更新　2022.10.5 更新

モジュール名	蓄電池 G2018
クラス	RechargeableBatteryModule2018

（1）入力画面

・スペック（画面を更新）





（2）モジュールの概要

本モジュールは、蓄電池本体および PCS(パワーコンディショナー)で構成され、蓄電池用制御モジュールと組み合わせて使用する。

蓄電池のタイプ別に充放電特性を用意しており、次のタイプの計算が可能である。

0_リチウムイオン電池

1_鉛電池

2_NAS 電池

3_充電時間比例

4_レドックスフロー電池

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	―	―		
	■蓄電池■								
1	定格蓄電容量	double		10	[kWh]	―	0		
3	初期蓄電容量	double		0	[kWh]	―	0		←計算開始時の蓄電容量
4	容量保持率	double		0.8	[-]	1	0		←参考（Li=0.8、鉛＝0.8、NAS＝0.72）
5	定格放電電力	double		1	[kW]	―	0		
6	ベース放電電力			1	[kW]	―	0		←ピークシフト制御時のベース放電電力を入力してください
	■充放電の上下限の設定等■								
7	放電停止下限充足率	double			[-]				←放電を停止とする充電率の下限
8	充電停止上限充足率	double			[-]				←充電を停止とする充電率の上限
9	充電時間率	double			[-]	24	0		←充電時間率＝ 定格蓄電容量 ÷ 1 時間当たりの目標充電容量
10	充電特性	String		0_リチウムイオン電池、1_鉛電池、2_NAS 電池、3_充電時間比例、4_レドックスフロー電池	[-]		0		←充電特性を指定
	■蓄電池システムの充放電・待機時の効率■								
11	ＰＣＳ充電時の効率	double		0.95	[-]				←参考（Li=0.95、鉛＝0.95、NAS＝0.95）
12	ＰＣＳ放電時の効率	double		0.95	[-]				←参考（Li=0.95、鉛＝0.95、NAS＝0.95）
13	蓄電池本体の効率	double		0.95	[-]	1	0		←参考（Li=0.95、鉛＝0.85、NAS＝0.90）
14	その他（補機等）の効率放電時	double		1	[-]	1	0		←参考（Li=1.00、鉛＝1.00、NAS＝1.00）
15	その他（補機等）の効率充電時	double		1	[-]	1	0		←参考（Li=1.00、鉛＝1.00、NAS＝1.00）

16	待機時の効率非充電時	double		1	[-]	1	0		←参考（Li=1.00、鉛＝1.00、NAS＝0.86）
17	待機時の効率充電時	double		1	[-]	1	0		←参考（Li=1.00、鉛＝1.00、NAS＝0.86）
	■蓄電池システムの発熱■								
18	熱損失係数	double		1	[-]				←電力損失の熱への変換率
	■供給先系統数・倍率■								
19	eleOut[]出口接続ノード数	int		1	[-]		1		←電力の供給先系統数を整数で入力して下さい
	■発電・CGS 太陽光風力など■								
20	eleInG[]発電入口接続ノード数	int		3	[-]		1		←発電の受入系統数を整数で入力して下さい
21	逆潮流する			FALSE	[-]				←逆潮流するときはチェックしてください
	■設置室■								
22	室グループ／室／ゾーン	String		#RoomGroup	[-]				←室グループ／室／ゾーンを選択してください。蓄電池の発熱をこの室へ渡します。
	■発電出力移動平均値・発電出力変動抑制制御■								←出力変動抑制制御は、次の入力行の「発電電力移動平均値１」 に対して充放電制御します
23	発電電力移動平均値１の算定ステップ数	int		4	[-]		0		←発電電力の合計値の移動平均値１算定のための計算ステップ数を入力してください
24	発電電力移動平均値２の算定ステップ数	int		12	[-]		0		←発電電力の合計値の移動平均値２算定のための計算ステップ数を入力してください
25	発電電力移動平均値３の算定ステップ数	int		36	[-]		0		←発電電力の合計値の移動平均値３算定のための計算ステップ数を入力してください
	■記録・グラフ表示■								
26	グラフを表示する	boolean		FALSE	[-]				←グラフを表示するときはチェックしてください
27	最大同時表示ステップ数	int		100	[-]		1		←グラフに同時表示する最大ステップ数を入力します
28	記録を有効とする	boolean	isRecorde	FALSE	[-]	—	—		←このモジュールの記録を有効と

									するときはチェックしてください
	■仮設調整■								
29	蓄電容量と放電能力を調整する	boolean		FALSE					←蓄電容量とｊ放電能力を調整するときはチェックしてください
	■RF 電池補機係数■								
30	負荷係数_ポンプ充放電 a1	double		0.1	[-]		0		←参考（250kW／0.1）
31	負荷係数_ポンプ充電 a2	double		0.02	[-]		0		←参考（250kW／0.02）
32	負荷係数_ポンプ放電 a3	double		0.02	[-]		0		←参考（250kW／0.02）
33	負荷係数_ポンプ充電 b1	double		8	[-]		0		←参考（250kW／8）
34	負荷係数_ポンプ放電 b2	double		0.5	[-]		0		←参考（250kW／0.5）
35	負荷係数_ポンプ充電後停止直後 b3	double		0.016	[-]		0		←参考（250kW／0.016）
36	負荷係数_ポンプ停止状態 b4	double		0.005	[-]		0		←参考（250kW／0.005）
37	負荷係数_ポンプ充電時式切替 s1	double		0.55	[-]		0		←参考（250kW／0.55）
38	負荷係数_ポンプ放電時式切替 s2	double		0.7	[-]		0		←参考（250kW／0.7）
39	負荷係数_熱交換器ファン放電 c1	double		0.012	[-]		0		←参考（250kW／0.012）
40	負荷係数_熱交換器ファン稼働率補正 d1	double		0.9	[-]		0		←参考（250kW／0.9）
41	負荷係数_熱交換器ファン稼働率補正 d2	double		0.75	[-]		0		←参考（250kW／0.75）
42	負荷係数_熱交換器ファン稼働率補正 d3	double		0.5	[-]		0		←参考（250kW／0.5）
43	負荷係数_電解液加温稼働率補正 f1	double		0	[-]		0		←参考（250kW／0.）
44	負荷係数_電解液加温稼働率補正 f2	double		0.5	[-]		0		←参考（250kW／0.5）
45	負荷係数_電解液加温稼働率補正 f3	double		1	[-]		0		←参考（250kW／1.）
46	負荷係数_その他制御機器 g1	double		0.008	[-]		0		←参考（250kW／0.008）
47	負荷係数_共通稼働係数 z	double		0.5	[-]		0		←参考（250kW／0.5）
48	補機を eleIn2 系統とする	double		FALSE	[-]		0		←RF の補機を eleIn2 系統とする場合チェックしてください

（５）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	REC	-	記録	メモリ	出口	空調記録モジュールへ出力する。
2	発停	L1_swcIn	swcIn	-	制御	発停	入口	上位からの発停信号
3	発停　充電	L1_swcInECharge	swcInECharge	-	制御	発停	入口	蓄電池充放電制御モジュールからの充電発停信号
4	発停　放電	L1_swcInEDischarge	swcInEDischarge	-	制御	発停	入口	蓄電池充放電制御モジュールからの放電発停信号
5	運転モード	L1_modIn	modIn	-	制御	制御モード	入口	上位からの運転モード信号
6	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	上位からの電力入口
7	電力出口	L0_eleOut[]	eleOut[]	-	状態	電力	出口	下位（電力負荷）への電力出口
8	発電電力入口	L0_eleInGen[]	eleInGen[]	-	状態	電力	入口	発電電力入口
9	周囲空気	L0_airIn	airIn	-	状態	空気	入口	蓄電池の周囲温度
10	放熱	L0_heaOut	heaOut	-	状態	熱	出口	（未使用）
11	電力損失	L0_eleObsLoss	eleObsLoss	-	状態	電力	観察	蓄電池の電力損失
12	電力　充電	L0_eleObsCharge	eleObsCharge	-	状態	電力	観察	充電電力
13	電力　放電	L0_eleObsDischarge	eleObsDischarge	-	状態	電力	観察	放電電力
14	電力　平準化	L0_eleOutSmooth	eleOutSmooth	-	状態	電力	出口	平準化電力
15	値：充電率入口	L0_valInEChargeRate	valInEChargeRate	-	状態	値	入口	蓄電池充放電制御モジュールから充電率を受けとる
16	値：放電率入口	L0_valInEDischargeRate	valInEDischargeRate	-	状態	値	入口	蓄電池充放電制御モジュールから放電率を受けとる
17	値：電力目標値	L0_valInTargetW	valInTargetW	-	状態	値	入口	蓄電池充放電制御モジュールから目標値を受けとる
18	値：使い切り放電率	L0_valInZeroRate	valInZeroRate	-	状態	値	入口	蓄電池充放電制御モジュールから使い切りのための放電率を受けとる
19	値：運転目標値	L0_valInOpeSetRate	valInOpeSetRate	-	状態	値	入口	蓄電池充放電制御モジュールから運転目標値を受けとる

（6）記録項目

No.	項目	出力内容	単位	分類
1	RB_Message#-#-	メッセージ	ー	メッセージ
2	RB_入口有効電力#W#-		W	入口
3	RB_入口無効電力#Var#-		Var	入口
4	RB_入口 2 有効電力#W#-		W	出口
5	RB_入口 2 無効電力#Var#-		Var	出口
6	RB_蓄放電電力#W#-		W	出口
7	RB_電池部分蓄放電電力#W#-		W	出口
8	RB_待機時電力#W#-		W	出口
9	RB_蓄電容量変化量#Ws/Step#-		Ws/Step	出口
10	RB_蓄電容量#Ws#-		Ws	
11	RB_損失合計#Ws/Step#-		Ws/Step	
12	RB_損失合計#W#-		W	
13	RB_発熱量#W#-		W	
14	RB_充電率#-#-		-	
15	RB_放電停止下限充足率#-#-		-	
16	RB_充電停止上限充足率#-#-		-	
17	RB_充電特性充電可能電力#W#-		W	
18	RB_定格放電電力#W#-		W	
19	RB_ベース放電電力#W#-		W	
20	RB_発電電力合計有効電力#W#-		W	
21	RB_発電電力移動平均値 1#W#-		W	
22	RB_発電電力移動平均値 2#W#-		W	
23	RB_発電電力移動平均値 3#W#-		W	
24	RB_発電電力と移動平均値 1 の差#W#-		W	
25	RB_発電電力と移動平均値 2 の差#W#-		W	
26	RB_発電電力と移動平均値 3 の差#W#-		W	
27	RB_発電電力出力変動抑制後#W#-		W	

28	RB_出側合計有効電力#W#-		W	
29	RB_未消費発電電力#W#-		W	
30	RB_未消費放電電力#W#-		W	
31	RB_eleOut[*]有効電力#W#有効電力		W	
32	RB_eleOut[*]無効電力#Var#無効電力		Var	
33	RB_eleInGen[*]有効電力#W#有効電力		W	
34	RB_eleInGen[*]無効電力#Var#無効電力		Var	
35	RB_調整蓄電容量#Ws#-		W	
36	RB_調整放電能力#W#-		W	
37	RB_RF 補機消費電力 Pump#W#-		W	
38	RB_RF 補機消費電力 HEXFan#W#-		W	
39	RB_RF 補機消費電力 Heater#W#-		W	
40	RB_RF 補機消費電力 Ctrl#W#-		W	

（７）計算フロー・計算内容

省略

・制御方法の切替

蓄電池の充放電の制御は、「蓄電池充放電制御_標準 2013」モジュールで設定し、蓄電池の充電と放電の計算は制御モジュールと接続した次の接続ノードの信号で切り替える。

充電：L1_swcInECharge

放電：L1_swcInEDischarge

また、制御モジュールと接続した以下の接続ノードから各種運転目標値等を受取り、充放電を行う。

充電率：L0_valInEChargeRate

放電率：L0_valInEDischargeRate

電力目標値：L0_valInTargetW

使い切り放電率：L0_valInZeroRate

運転目標値：L0_valInOpeSetRate

・PCS 効率

PCS 効率は固定効率と負荷率によって変化する変動効率の 2 種類から計算可能である。

PCS 効率を固定効率で計算する場合

PCS 効率を固定効率で計算する場合は、入力項目の「ＰＣＳ充電時の効率」と「ＰＣＳ放電時の効率」に入力された値で計算する。固定効率の場合は入力する数値は 1 個である。

PCS 効率を変動効率で計算する場合

PCS 効率を変動効率で計算する場合は、次に示す特性式の分母の 2 次式の 3 個の係数 a,b,c を半角の空白で区切って入力する。

特性式

$$PCS効率 = \frac{x}{ax^2 + bx + c}$$

ここで x は負荷率（0～1）

（定数の説明）

二次の項の係数 a は，インバータ出力電流に依存する損失分であり 0 以上の値

一次の項の係数 b は，素子の電圧降下分に依存する係数であり 1 以上の値

定数項 c は，その他の回路の限界効率等に依存する値で 0 以上の値

参考値：a=0.014, b=1.03, c=0.0093

（８）データ範囲と範囲外の取扱い

特になし。

2022.10.5 更新事項

1．NAS 電池の PCS 充・放電時の効率、蓄電池本体の効率のデフォルト値

入力画面で参考表示している値を調査実測事例をもとに次のように変更した。

PCS 充電時の効率 NAS=0.95→0.96

PCS 放電時の効率 NAS=0.95→0.96

蓄電池本体の効率 NAS=0.9→0.88

2．その他（補機等）の効率

その他（補機等）の効率を充電時と放電時に分けて扱うこととする。

その他（補機等）充電時の効率

その他（補機等）放電時の効率

これまでは、充電効率の算定に「その他（補機等）の効率」を使用

充電効率 ＝ PCS 充電時効率 × その他(補機等)の効率 × 蓄電池の効率

・蓄電池システムへの充電時電力

＝ 蓄電池への電力 ＋ 待機時電力（充電時） ＋ RF の補機電力

・待機時電力（充電時）＝ 定格放電電力×（１－待機時の効率（充電時））

・蓄電池部分に充電される電力 ＝ 蓄電池への電力 × 充電効率

放電効率 ＝ PCS 放電時効率

・蓄電池システムからの放電時電力

＝ 蓄電池からの電力

・蓄電池部分の放電される電力 ＝ 蓄電池からの電力 ／ 放電効率 － RF の補機電力

次のように修正する。

充電効率 ＝ PCS 充電時効率 × その他(補機等)の効率(充電時) × 蓄電池の効率

放電効率 ＝ PCS 放電時効率 × その他(補機等)の効率(放電時)

3．入力項目の名称変更

待機時の効率→待機時の効率非充電時　と変更した。

2021.12.6 更新事項

1．シーケンス接続の表の備考欄の補足説明を青色文字で追記した。

「蓄電池充放電制御_標準 2013」（場所：設備 2015／電気設備 2015／）

モジュール名	蓄電池充放電制御_標準 2013
クラス	ControlEChargeStandardModule201312

（1）入力画面

・スペック

（2）モジュールの概要

蓄電池の充放電を制御するモジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

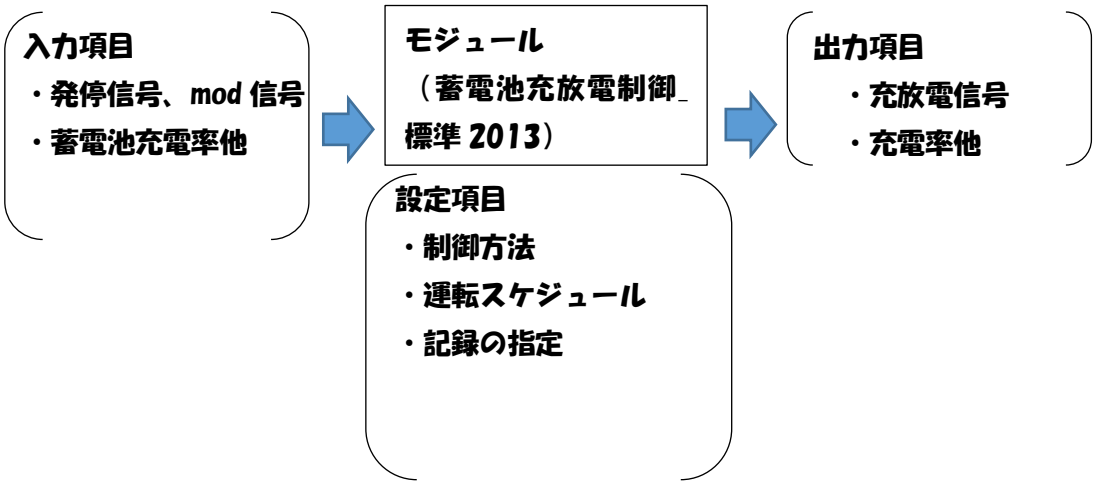


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	Name	ー	[-]	ー	ー		
1	■DR 制御・条件■					ー	ー		
2	OPE1_DR 制御を実施する	boolean	isDRCtrl[0]	FALSE	[-]	ー	ー		←上位からの swcIn 指令により OPE1_DR 制御を実施する時はチェックをしてください。
3	OPE1_下げ DR 制御時の設定放電電力リスト	String	drSetValue[0]	0 50 100 200 300	[kW][kW]・・	ー	ー		←OPE1_下げ DR 制御するレベル別に放電電力を半角のスペースで区切って入力してください。
4	OPE1_上げ DR 制御時の設定充電電力リスト	String	drSetValue[0]	0 50 100 200 300	[kW][kW]・・	ー	ー		←OPE1_上げ DR 制御するレベル別に充電電力を半角のスペースで区切って入力してください。
5	OPE2_DR 制御を実施する	boolean	isDRCtrl[1]	FALSE	[-]	ー	ー		←上位からの swcIn 指令により OPE2_DR 制御を実施する時はチェックをしてください。
6	OPE2_下げ DR 制御時の設定放電電力リスト	String	drSetValue[1]	0 50 100 200 300	[kW][kW]・・	ー	ー		←OPE2_下げ DR 制御するレベル別に放電電力を半角のスペースで区切って入力してください。
7	OPE2_上げ DR 制御時の設定充電電力リスト	String	drSetValue[1]	0 50 100 200 300	[kW][kW]・・	ー	ー		←OPE2_上げ DR 制御するレベル別に充電電力を半角のスペースで区切って入力してください。
8	OPE3_DR 制御を実施する	boolean	isDRCtrl[2]	FALSE	[-]	ー	ー		←上位からの swcIn 指令により OPE3_DR 制御を実施する時はチェックをしてください。
9	OPE3_下げ DR 制御時の設定放電電力リスト	String	drSetValue[2]	0 50 100 200 300	[kW][kW]・・	ー	ー		←OPE3_下げ DR 制御するレベル別に放電電力を半角のスペースで区切って入力してください。
10	OPE3_上げ DR 制御時の設定充電電力リスト	String	drSetValue[2]	0 50 100 200 300	[kW][kW]・	ー	ー		←OPE3_上げ DR 制御するレベル別に充電電力を半角のスペースで区

					.				切って入力してください。
11	■運転スケジュール■					―	―		
12	このスケジュールを使用する	boolean	isUseThisOpe	TRUE	[-]	―	―		←上位コントローラのスケジュールを使う場合はチェックをはずして下さい。
13	■DailyAnnualSchedule の指定■					―	―		
14	充電 DailyAnnualSchedule を使用する	boolean	isUseChargeDailyAnnualSchedule	FALSE	[-]	―	―		←充電 DailyAnnualSchedule を使用する場合はチェックして下さい。
15	充電 DailyAnnualSchedule	String	schNameCharge	#Userm、時刻変動スケジュール	[-]	―	―		
17	放電 DailyAnnualSchedule を使用する	boolean	isUseDischargeDailyAnnualSchedule	FALSE	[-]	―	―		←放電 DailyAnnualSchedule を使用する場合はチェックして下さい。
18	放電 DailyAnnualSchedule 名	String	schNameDischarge	#Userm、時刻変動スケジュール	[-]	―	―		
20	■期間・曜日別の運転スケジュールの設定■					―	―		
21	■OPE・運転スケジュール■					―	―		
22	この OPE/期間別の充放電スケジュールを使用する	boolean	isUseThisMode	TRUE	[-]	―	―		←上位コントローラの期間別スケジュールを使う場合はチェックをはずして下さい。
23	OPE1/夏期　放電　開始月日ー終了月日	String	OPE1EC_START_END	6/1-9/30	[月/日]-[月/日]	―	―		←入力例[5/1-11/30]　月と日を半角の[/]で、開始と終了を半角の[-]で区切る。
24	OPE2/冬期　放電　開始月日ー終了月日	String	OPE2EC_START_END	12/1-3/31	[月/日]-[月/日]	―	―		←入力例[12/1-4/30]　月と日を半角の[/]で、開始と終了を半角の[-]で区切る。
25	OPE3/中間期　放電　開始月日ー終了月日	String	OPE3EC_START_END	4/1-5/31 / 10/1-11/30	[月/日]-[月/日]	―	―		←入力例[12/1-4/30]　月と日を半角の[/]で、開始と終了を半角の[-]で区切る。

26	OPE1/夏期　蓄電　開始月日ー終了月日	String	OPE1EC_START_END	6/1-9/30	[月/日]-[月/日]	ー	ー		←入力例[5/1-11/30]　月と日を半角の[/]で、開始と終了を半角の[-]で区切る。
27	OPE2/冬期　蓄電　開始月日ー終了月日	String	OPE2EC_START_END	12/1-3/31	[月/日]-[月/日]	ー	ー		←入力例[12/1-4/30]　月と日を半角の[/]で、開始と終了を半角の[-]で区切る。
28	OPE3/中間期　蓄電　開始月日ー終了月日	String	OPE3EC_START_END	4/1-5/31 / 10/1-11/30	[月/日]-[月/日]	ー	ー		←入力例[12/1-4/30]　月と日を半角の[/]で、開始と終了を半角の[-]で区切る。
29	OPE1/夏期の運用	String	controlOperationModCodes[0]	0_停止、1_ピークシフト制御、2_ピークシフト＋ピークカット（放電）制御、3_ピークシフト＋ピークカット（充放電）制御、4_ピークカット（放電）制御、5_ピークカット（充放電）制御、6_発電電力平準化（一定値）制御、7_出力変動抑制制御、8_負荷追従制御		ー	ー		
30	OPE2/冬期の運用	String	controlOperationModCodes[1]	0_停止、1_ピークシフト制御、2_ピークシフト＋ピークカット（放電）制御、3_ピークシフト＋ピークカット（充放電）制御、4_ピークカット（放電）制御、5_ピークカット（充放電）制御、6_発電電力平準化（一定値）制御、7_出力変動抑制制御、8_負荷追従制御		ー	ー		
31	OPE3/中間期の運用	String	controlOperationModCodes[2]	0_停止、1_ピークシフト制御、2_ピークシフト＋ピークカット（放電）制御、3_ピークシフト＋ピークカット（充放電）制御、4_ピークカット（放電）制御、5_ピークカッ		ー	ー		

				ト（充放電）制御、6_発電電力平準化（一定値）制御、7_出力変動抑制制御、8_負荷追従制御					
32	OPE4/未指定日の運用	String	~~controlOperationModCodes[3]~~ airOpeSch.OPE4	0_停止、1_ピークシフト制御、2_ピークシフト＋ピークカット（放電）制御、3_ピークシフト＋ピークカット（充放電）制御、4_ピークカット（放電）制御、5_ピークカット（充放電）制御、6_発電電力平準化（一定値）制御、7_出力変動抑制制御、8_負荷追従制御		―	―		
33	■運転スケジュール■					―	―		←入力例[22:00-8:00 / 8:00-20:00 / 8:00-20:00] 時と分を半角の[:]で、開始と終了を半角の[-]で区切る。
34	■OPE1/夏期■					―	―		最初の時間帯は充電運転swcOutVCharge、中間は予備（未使用）、最後は放電運転swcOutVDischargeです。
35	OPE1_日曜日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]- [時:分]	―	―		運転しない場合は[0:00-0:00]とする。 時間帯と時間帯はスペースを含む" / "の半角３文字で区切る。
36	OPE1_月曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]- [時:分]	―	―		
37	OPE1_火曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]- [時:分]	―	―		
38	OPE1_水曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]- [時:分]	―	―		

39	OPE1_木曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	－	－		
40	OPE1_金曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	－	－		
41	OPE1_土曜日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]-[時:分]	－	－		
42	OPE1_祝日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]-[時:分]	－	－		
43	OPE1_特別日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]-[時:分]	－	－		
45	■OPE2/冬期■					－	－		
46	OPE2_日曜日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]-[時:分]	－	－		
47	OPE2_月曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	－	－		
48	OPE2_火曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	－	－		
49	OPE2_水曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	－	－		
50	OPE2_木曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	－	－		
51	OPE2_金曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	－	－		

52	OPE2_土曜日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]-[時:分]	―	―		
53	OPE2_祝日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]-[時:分]	―	―		
54	OPE2_特別日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]-[時:分]	―	―		
56	■OPE3、OPE4/中間期■					―	―		
57	OPE3_日曜日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]-[時:分]	―	―		
58	OPE3_月曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	―	―		
59	OPE3_火曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	―	―		
60	OPE3_水曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	―	―		
61	OPE3_木曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	―	―		
62	OPE3_金曜日	String	airOpeSch.controlOperationWeekCodes[]	22:00-8:00 / 0:00-0:00 / 8:00-22:00	[時:分]-[時:分]	―	―		
63	OPE3_土曜日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]-[時:分]	―	―		
64	OPE3_祝日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]-[時:分]	―	―		

65	OPE3_特別日	String	airOpeSch.controlOperationWeekCodes[]	0:00-0:00 / 0:00-0:00 / 0:00-0:00	[時:分]-[時:分]	―	―		
67	■■ピークシフト制御のオプション■					―	―		
68	■ピークカット制御・条件■					―	―		
70	ピークカットの目標値を月別設定する	boolean	isChange_PC	FALSE	[-]	―	―		←ピークカットの目標値を次の月別に可変とする場合はチェックしてください。
71	ピークカット目標値[年間と月別]	String	targetPower_PC[i]	13 1 2 3 4 5 6 7 8 9 10 11 12	[kW] [kW]・・	―	―		←ピークカット目標値（データは「年間固定値、1 月、2 月、～、12 月」の 13 個を半角スペースで区切る。）
72	■使い切り制御・条件■					―	―		
73	使い切り制御	String	zeroRemains	0_なし、1_使い切り制御（前詰め）、2_使い切り制御（平均）	[-]	―	―		←使い切り制御を指定して下さい。
74	使い切り制御開始時刻	String	startZRemainsH startZRemainsM	16:00	[時:分]	―	―		←使い切り制御の開始時刻を設定してください。
75	使い切り制御の時間	String	secondsZRemains	60	[分間]	―	―		←使い切り制御の放電時間を設定指定ください。
76	■■発電電力平準化（一定値）制御のオプション■					―	―		
77	■発電電力平準化（一定値）制御・条件■					―	―		
78	商用充電上限充足率	String	opeSetRate	0.5	[-]	―	―		←商用充電時の充電上限充足率を設定してください。
79	発電電力平準化（一定値）制御の目標値を月別設定する	boolean	isChangeConstPower	FALSE	[-]	―	―		←発電電力平準化（一定値）制御の目標値を次の月別に可変とする場合はチェックしてください。
80	発電電力平準化（一定値）制御の目標値[年間と月別]	String	targetConstPower[i]	13 1 2 3 4 5 6 7 8 9 10 11 12	[kW] [kW]・・	―	―		←発電電力平準化（一定値）制御の目標値 （データは「年間固定値、1 月、2

									月、～、12 月」の 13 個を半角スペースで区切る。）
84	■記録・グラフ表示■					－	－		
85	グラフを表示する	boolean	isGVisible	FALSE	[-]	－	－		←グラフを表示するときはチェックしてください
86	最大同時表示ステップ数	int	maxItemCount	100	[-]	－	－		←グラフに同時表示する最大ステップ数を入力します
87	記録を有効とする	boolean	isRecord	FALSE	[-]	－	－		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録	L2_recOut	recOut	-	記録	メモリ	出口	空調記録モジュールへ接続する
2	発停信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	上位からの発停信号
3	モード信号	L1_modIn	modIn	-	制御	制御モード	入口	上位からのモード信号
4	充電信号	L1_swcOutECharge	swcOutECharge	-	制御	OnOff 信号	出口	充電信号の出力
5	放電信号	L1_swcOutEDischarge	swcOutEDischarge	-	制御	OnOff 信号	出口	放電信号の出力
6	モード信号	L1_modOut	modOut	-	制御	制御モード	出口	モード信号の出力
7	充電率入口	L0_valInSTRate	valInSTRate	-	値	値	入口	蓄電池充電率の入口
8	充電率入口	L0_valOutEChargeRate	valOutEChargeRate	-	値	値	出口	充電率の入口
9	放電率入口	L0_valOutEDischargeRate	valOutEDischargeRate	-	値	値	出口	放電率の入口
10	目標電力の値の入口	L0_valOutTargetW	valOutTargetW	-	値	値	出口	目標電力の値の入口
11	使い切りのための放電率の入口	L0_valOutZeroRate	valOutZeroRate	-	値	値	出口	使い切りのための放電率の入口
12	平準化商用充電の上限充足率の入口	L0_valOutOpeSetRate	valOutOpeSetRate	-	値	値	出口	平準化商用充電の上限充足率の入口

（6）記録項目

No.	項目	出力内容	単位	分類
1	CtrEC_Message#-#-	メッセージ	—	メッセージ
2	CtrEC_swcIn#-#-	上位からの発停信号	—	入口
3	CtrEC_modIn#-#-	上位からのモード信号	—	入口
4	CtrEC_swcOutMain#-#-	発停信号—主	—	出口
5	CtrEC_swcOutECharge#-#-	発停信号—充電	—	出口
6	CtrEC_swcOutEDischarge#-#-	発停信号—放電	—	出口
7	CtrEC_valInEChargeRate#-#-	上位からの充電率	—	入口
8	CtrEC_valOutEChargeRate#-#-	充電率（出口）	—	出口
9	CtrEC_valOutEDischargeRate#-#-	放電率（出口）	—	出口
10	CtrEC_valOutTargetW#W#-	ピークカットなど目標値[W]	W	出口
11	CtrEC_valOutZeroRate#-#-	使い切りのための時間比例で求めた放電量率[-]	—	出口
12	CtrEC_valOutOpeSetRate#-#-	平準化時の商用充電上限充足率[-]	—	出口
13	CtrEC_modOut#-#-	モード信号		出口

（7）計算フロー・計算内容

省略

（8）データ範囲と範囲外の取扱い

省略

参考　ソース

```
package jp.or.ibec.best.domain.sample.air;

import java.io.BufferedReader;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;

import jp.or.ibec.best.DO.BestValue;
import jp.or.ibec.best.control.schedule.BestTimeManager;
import jp.or.ibec.best.control.schedule.IBestDomainSchedule;
import jp.or.ibec.best.domain.building.manager.ScheduleManager;
import jp.or.ibec.best.domain.building.schedule.DailyAnnualSchedule;
import jp.or.ibec.best.essentials.AbstractBestModule;
import jp.or.ibec.best.essentials.BestSpecs;
import jp.or.ibec.best.essentials.IBestControlMessage;
import jp.or.ibec.best.essentials.IBestMetaModule;
import jp.or.ibec.best.essentials.IBestRecordMessage;
import jp.or.ibec.best.essentials.IBestStateMessage;


/**
 * @author HIROSHI NINOMIYA /20120921
 *
 * 標準蓄熱制御　をもとに　蓄電池用コントローラを試作3
 *
 * 2013110充電、放電スケジュール
 */
public class ControlEChargeStandardModule201312 extends AbstractBestModule implements IBestMetaModule{

   private final String moduleName = "(ControlEChargeStandardModule201312) ";

   //接続ノード

   //制御ノード    swc**  運転状態(0:停止、1:運転)
   private final String C_NODE_swcIn  = "L1_swcIn" ;  //熱原運転状態

   private final String C_NODE_modIn  = "L1_modIn" ;  //熱原運転モード(0:停止,1:冷房,2:暖房)

   private final String C_NODE_swcOutEleCharge   = "L1_swcOutECharge";//蓄電
   private final String C_NODE_swcOutEleDischarge= "L1_swcOutEDischarge";//放電

   private final String C_NODE_modOut = "L1_modOut";//蓄電池運転モード(  停止,充電,放電  )

   private final String S_NODE_valInEChargeRate     = "L0_valInSTRate";//蓄電率
   private final String S_NODE_valOutEChargeRate    = "L0_valOutEChargeRate";//蓄電率
   private final String S_NODE_valOutEDischargeRate = "L0_valOutEDischargeRate";//放電率
   private final String S_NODE_valOutTargetW        = "L0_valOutTargetW";//ピークカットなど目標値[W]
   private final String S_NODE_valOutZeroRate       = "L0_valOutZeroRate";//使い切りのための時間比例で求めた放電量率[-]
   private final String S_NODE_valOutOpeSetRate     = "L0_valOutOpeSetRate";//平準化時の商用充電上限充足率[-]

   //記録ノード
   private final String R_NODE = "L2_recOut";

   private final String RECORD_Message  = "CtrEC_Message#-#-";
   private final String RECORD_swcIn    = "CtrEC_swcIn#-#-";
   private final String RECORD_modIn    = "CtrEC_modIn#-#-";

   private final String RECORD_swcOutMain       = "CtrEC_swcOutMain#-#-";
   private final String RECORD_swcOutECharge    = "CtrEC_swcOutECharge#-#-";
   private final String RECORD_swcOutEDischarge = "CtrEC_swcOutEDischarge#-#-";

// private final String RECORD_swcOutECandD     = "CtrEC_swcOutECandD#-#-";

   private final String RECORD_valInEChargeRate     = "CtrEC_valInEChargeRate#-#-";//蓄電率
   private final String RECORD_valOutEChargeRate    = "CtrEC_valOutEChargeRate#-#-";//蓄電率
   private final String RECORD_valOutEDischargeRate = "CtrEC_valOutEDischargeRate#-#-";//放電率
   private final String RECORD_valOutTargetW        = "CtrEC_valOutTargetW#W#-";//ピークカットなど目標値[W]
   private final String RECORD_valOutZeroRate       = "CtrEC_valOutZeroRate#-#-";//使い切りのための時間比例で求めた放電量
```

```
率[-]
    private final String RECORD_valOutOpeSetRate     = "CtrEC_valOutOpeSetRate#-#-";//平準化時の商用充電上限充足率[-]

    private final String RECORD_modOut           = "CtrEC_modOut#-#-";

    //private final String RECORD_dayload2Sum = "CtrEC_２次側日負荷合計#J#負荷" ;

    //外部定義項目
    //仕様
    private final String SPEC_name      = "名称";

    private static final String SPEC_isOPE1_DRCtrl = "isOPE1_DR制御実施";
    private static final String SPEC_isOPE2_DRCtrl = "isOPE2_DR制御実施";
    private static final String SPEC_isOPE3_DRCtrl = "isOPE3_DR制御実施";

    private static final String SPEC_OPE1_DRSetValueList = "OPE1_DR設定値リスト[-]";
    private static final String SPEC_OPE2_DRSetValueList = "OPE2_DR設定値リスト[-]";
    private static final String SPEC_OPE3_DRSetValueList = "OPE3_DR設定値リスト[-]";

    private final String SPEC_isUseThisOpe  = "このスケジュールを使用する";
    private final String SPEC_isUseThisMode = "この充放電期間を使用する";

    //20130322
    private final String SPEC_isUseChargeDailyAnnualSchedule    = "充電DailyAnnualScheduleを使用する";
    private final String SPEC_isUseDischargeDailyAnnualSchedule = "放電DailyAnnualScheduleを使用する";
    private final String SPEC_schNameCharge    = "充電スケジュール名[-]";
    private final String SPEC_schNameDischarge = "放電スケジュール名[-]";

    //20130111
    private AirOpeSchedule20130101 airOpeSch = new AirOpeSchedule20130101();//放電側スケジュール

// private final String SPEC_isPeakCut      = "ピークカットする";//false;//
    private final String SPEC_isChange_PC    = "ピークカットの目標値を月別設定する";
    private final String SPEC_targetPower_PC = "ピークカット目標値月別[kW]";

    private final String SPEC_ZeroRemains  = "使い切り制御[-]";//false;//
    private final String SPEC_startZRemains  = "使い切り制御開始時刻[-]";
    private final String SPEC_secondsZRemains= "使い切り制御の時間[s]";

    private final String SPEC_opeSetRate         = "商用充電上限充足率[-]";
    private final String SPEC_isChangeConstPower = "発電電力平準化（一定値）制御の目標値を月別設定する";
    private final String SPEC_targetConstPower   = "発電電力平準化（一定値）制御の目標値月別[kW]";

    /*******************************************************/
    private final String SPEC_OPE1EC_START_END  = "OPE1蓄電開始月日ー終了月日[-]";
    private final String SPEC_OPE2EC_START_END  = "OPE2蓄電開始月日ー終了月日[-]";
    private final String SPEC_OPE3EC_START_END  = "OPE3蓄電開始月日ー終了月日[-]";

    private final String SPEC_isGVisible   = "グラフを表示する";
    private final String SPEC_maxItemCount = "最大同時表示ステップ数";
    private final String SPEC_isRecord     = "記録を有効とする";
    //
    private StringBuffer message = new StringBuffer();

    //仕様など
    private String  name = null;

    //DR
    protected boolean[] isDRCtrl = {false, false, false};//"isRD制御実施";
    protected double[][] drSetValue = new double[3][];//"RD設定値リスト[-]";

    /**
     * このモジュールのスケジュールを使用する場合＝true
     * 上位からのスケジュールを使用する場合＝false
     */
    private boolean isUseThisOpe;
    private boolean isUseThisMode;

    private boolean isUseChargeDailyAnnualSchedule;
    private boolean isUseDischargeDailyAnnualSchedule;
    private String schNameCharge = null;
    private String schNameDischarge = null;
```

```
private BestValue valInEChargeRate = null;//20120719nino

private BestValue valOutEChargeRate = null;//"LO_valOutEChargeRate";//蓄電率
private BestValue valOutEDischargeRate = null;//"LO_valOutEChargeRate";//蓄電率
private BestValue valOutTargetW = null;//"LO_valOutTargetW";//ピークカットなど目標値
private BestValue valOutZeroRate = null;//"LO_valOutZeroRate";//使い切りのための時間比例で求めた放電量率
private BestValue valOutOpeSetRate = null;//"LO_valOutOpeSetRate";//平準化時の商用充電上限充足率

private double setValOutEChargeRate = 0;
private double setValOutEDischargeRate = 0;

/**
 * ピークカットする
 */
private boolean isPeakCut      = false;
/**
 * 充放電ピークカットする
 */
private boolean isPeakCutALL   = false;
/**
 * ピークカットの目標値を月別設定する
 */
private boolean isChange_PC    = false;
/**
 *   ピークカット目標値月別[kW]：[0]年間固定値、[1]～[12]１月から12月
 */
private double[] targetPower_PC = new double[13];
/**
 * 放電時のピークカット目標値を0kWに変更するための係数
 */
private double   targetPower_PCeff = 1;
private String   zeroRemains  = null;//"使い切り制御";
private int zeroRemainsType = 0;//"使い切り制御"のタイプ

private String startZRemains  = null;//"使い切り制御開始時刻[-]";
private double secondsZRemains;//= "使い切り制御の時間[s]";
private double opeSetRate;//= "平準化の時の商用充電上限充足率[-]";

private double secondsCountDown;

private boolean isConstPower = false;
private boolean isChangeConstPower = false;//"発電電力平準化（一定値）制御の目標値を月別設定する";
private double[] targetConstPower =  new double[13];//"発電電力平準化（一定値）制御の目標値月別[kW]";

private boolean isCtrlPowerChangeRate = false;//出力変動抑制制御
private boolean isLoadFollowingControl = false;//負荷追従制御

private boolean isPeakShift =false;


private int startZRemainsH;//時
private int startZRemainsM;//分

/**
 * 上位からのon/off信号入力
 */
private int[] swcIn = new int[1];

/**
 * 上位からのmode信号入力
 */
private int[] modIn = new int[1];

private int[] modOut = new int[1];

private int[] swcOut = new int[3];
private int swcOutMain;
private int swcOutCharge;
private int swcOutDischarge;

private String[] name_swcIn  = { "上位からのswcIn" };
```

```
    private String[] name_modIn  = { "上位からのmodIn" };
    private String[] name_swcOut = { "swcOutMain","swcOutCharge","swcOutDischarge" };
    private String[] name_modOut = { "modOut" };
    //

    /**
     * OPE1の蓄電開始月日-終了月日の入力文字列
     */
    private String OPE1EC_START_END    = null;

    /**
     * OPE2の蓄電開始月日-終了月日の入力文字列
     */
    private String OPE2EC_START_END    = null;

    /**
     * OPE3の蓄電開始月日-終了月日の入力文字列
     */
    private String OPE3EC_START_END    = null;

    /*************************************************************/

    private boolean isGVisible = false;
    private int  maxItemCount = 100;
    private boolean isRecord   = false;

    /**
     * "冷房時熱源への限界送水温度";以下の時=true
     */
    private boolean isCoolLimitTempHSin;//"冷房時熱源への限界送水温度";以下の時=true
    /**
     * "暖房時熱源への限界送水温度";以上の時=true
     */
    private boolean isHeatLimitTempHSin;//"暖房時熱源への限界送水温度";以上の時=true
    /**
     * "冷房時２次側への限界送水温度";以上の時=true
     */
    //private boolean isCoolLimitTempDischarge;//"冷房時２次側への限界送水温度";以上の時=true
    /**
     * "暖房時２次側への限界送水温度";以下の時=true
     */
    //private boolean isHeatLimitTempDischarge;//"暖房時２次側への限界送水温度";以下の時=true
    //
    //private double hsRestartInterval = 3600.;//"熱源の再起動までの停止時間[s]";//熱源の再起動までの停止時間[s]

    /**
     * "負荷予測の指定"
     */
// private String loadPredictType;// = "負荷予測の指定";
    private int loadPredictTypeNum = 0;//負荷予測のタイプ番号

    //内部変数

    private String[] controlOperationModCodesStrs = null;//蓄熱運転[COOL_START_END, HEAT_START_END]

    private ControlOperationModCode[] controlOperationModCodes = null;

// private int modeMy;//合成

// private int swcMy;

// private int swcMyST;

// private int swcMy2;

    //グラフ表示など
    private GraphCommonJFrame2015 gGCJF = null;
    private GraphCommonData2015[] gData = null;

    private BufferedReader br = null;
    private HashMap<Integer, Double> loadDataValue = new HashMap<Integer, Double>();
```

```
private DailyAnnualSchedule dasCharge = null;

private DailyAnnualSchedule dasDischarge = null;

//
protected final static int i_Stop = 0;                    //0_停止
protected final static int i_PeakShift = 1;                  //1_ピークシフト制御
protected final static int i_PeakShiftCutDisCharge = 2;      //2_ピークシフト＋ピークカット（放電）制御
protected final static int i_PeakShiftCutAll = 3;            //3_ピークシフト＋ピークカット（充放電）制御
protected final static int i_PeakCutDisCharge = 4;           //4_ピークカット（放電）制御
protected final static int i_PeakCutAll = 5;              //5_ピークカット（充放電）制御
protected final static int i_LevelGenPower = 6;              //6_発電電力平準化（一定値）制御
protected final static int i_SuppressGenPowerFluctuation = 7; //7_出力変動抑制制御
protected final static int i_LoadFollowing = 8;              //8_負荷追従制御


@Override
public void setProfile(BestSpecs spec) {
   if( spec == null ){
      return;
   }
   Map<String, String> map = spec.getSpec();
   if( map == null ){
      return;
   }

   //名称を取得
   this.name = spec.getSpecValue( this.SPEC_name, this.moduleName );

   //デマンドレスポンス制御　実施する＝true
   //private final String SPEC_isOPE1_DRCtrl = "isOPE1_DR制御実施";
   this.isDRCtrl[0] = spec.getSpecValue( SPEC_isOPE1_DRCtrl, false );
   //private final String SPEC_isOPE2_DRCtrl = "isOPE2_DR制御実施";
   this.isDRCtrl[1] = spec.getSpecValue( SPEC_isOPE2_DRCtrl, false );
   //private final String SPEC_isOPE3_DRCtrl = "isOPE3_DR制御実施";
   this.isDRCtrl[2] = spec.getSpecValue( SPEC_isOPE3_DRCtrl, false );

   //デマンドレスポンス制御　設定値リスト
   //private final String SPEC_RDGenPowerList = "DR設定値リスト[-]";
   String str = spec.getSpecValue( SPEC_OPE1_DRSetValueList, "10" );
   str = str.replace( "_", " " );
   String[] strs = null;
   strs = str.split( "[ /]" );
   this.drSetValue[0] = new double[strs.length];
   for( int i=0; i<strs.length; i++ ){
      this.drSetValue[0][i] = Double.parseDouble( strs[i] );
      //System.out.println( "drSetValue[0]["+i+"]="+this.drSetValue[0][i] );
      this.message.append("()drSetValue[0]["+i+"]="+this.drSetValue[0][i] );
   }

   str = spec.getSpecValue( SPEC_OPE2_DRSetValueList, "10" );
   str = str.replace( "_", " " );
   strs = null;
   strs = str.split( "[ /]" );
   this.drSetValue[1] = new double[strs.length];
   for( int i=0; i<strs.length; i++ ){
      this.drSetValue[1][i] = Double.parseDouble( strs[i] );
      //System.out.println( "drSetValue[1]["+i+"]="+this.drSetValue[1][i] );
      this.message.append("()drSetValue[1]["+i+"]="+this.drSetValue[1][i] );
   }

   str = spec.getSpecValue( SPEC_OPE3_DRSetValueList, "10" );
   str = str.replace( "_", " " );
   strs = null;
   strs = str.split( "[ /]" );
   this.drSetValue[2] = new double[strs.length];
   for( int i=0; i<strs.length; i++ ){
      this.drSetValue[2][i] = Double.parseDouble( strs[i] );
      //System.out.println( "drSetValue[2]["+i+"]="+this.drSetValue[2][i] );
      this.message.append("()drSetValue[2]["+i+"]="+this.drSetValue[2][i] );
   }
```

```
        //
        this.isUseThisOpe = spec.getSpecValue( this.SPEC_isUseThisOpe, true );
        //
        this.isUseThisMode = spec.getSpecValue( this.SPEC_isUseThisMode, true );

        //20130322
        this.isUseChargeDailyAnnualSchedule = spec.getSpecValue( this.SPEC_isUseChargeDailyAnnualSchedule, false );

        this.isUseDischargeDailyAnnualSchedule = spec.getSpecValue( this.SPEC_isUseDischargeDailyAnnualSchedule, false );

        this.schNameCharge = spec.getSpecValue( this.SPEC_schNameCharge, "" );

        this.schNameDischarge = spec.getSpecValue( this.SPEC_schNameDischarge, "" );

        if( this.isUseChargeDailyAnnualSchedule ){
           if( this.schNameCharge.equals("" )){
              this.isUseChargeDailyAnnualSchedule = false;
           }else{
              this.dasCharge = ScheduleManager.getDailyScheduleManager().getDailyAnnualSchedule( this.schNameCharge );
           }
        }
        //System.out.println( "isUseChargeDailyAnnualSchedule="+isUseChargeDailyAnnualSchedule+"
schNameCharge="+schNameCharge);

        if( this.isUseDischargeDailyAnnualSchedule ){
           if( this.schNameDischarge.equals("" )){
              this.isUseDischargeDailyAnnualSchedule = false;
           }else{
              this.dasDischarge =
ScheduleManager.getDailyScheduleManager().getDailyAnnualSchedule( this.schNameDischarge );
           }
        }

        //20130111
        if( this.isUseThisOpe ){
           this.airOpeSch.setProfile(spec);
        }

        //"OPE1蓄熱開始月日ー終了月日";
        this.OPE1EC_START_END = spec.getSpecValue( this.SPEC_OPE1EC_START_END, "7/1-9/30" );
        this.OPE1EC_START_END = this.OPE1EC_START_END.replace( '_', ' ' );//20120628nino

        //"OPE2蓄熱開始月日ー終了月日";
        this.OPE2EC_START_END = spec.getSpecValue( this.SPEC_OPE2EC_START_END, "12/1-4/30" );
        this.OPE2EC_START_END = this.OPE2EC_START_END.replace( '_', ' ' );//20120628nino

        //"OPE3蓄熱開始月日ー終了月日";
        this.OPE3EC_START_END = spec.getSpecValue( this.SPEC_OPE3EC_START_END, "5/1-6/30 / 10/1-11/30" );
        this.OPE3EC_START_END = this.OPE3EC_START_END.replace( '_', ' ' );//20120628nino

        //isGVisibleを取得
        this.isGVisible = spec.getSpecValue( this.SPEC_isGVisible, false );

        // 最大同時表示ステップ数
        this.maxItemCount = spec.getSpecValue( this.SPEC_maxItemCount, 100 );

        //isRecordを取得
        this.isRecord = spec.getSpecValue( this.SPEC_isRecord, false );

        // "HOL", "SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT", "SPC"
        //OPE1,2,3蓄電期間
        this.controlOperationModCodesStrs = new String[3];
        this.controlOperationModCodesStrs[0] = this.OPE1EC_START_END;
        this.controlOperationModCodesStrs[1] = this.OPE2EC_START_END;
        this.controlOperationModCodesStrs[2] = this.OPE3EC_START_END;

        this.controlOperationModCodes = new ControlOperationModCode[3];
        this.controlOperationModCodes[0] = new ControlOperationModCode( this.airOpeSch.OPE1_mod,
this.controlOperationModCodesStrs[0] );
        this.controlOperationModCodes[1] = new ControlOperationModCode( this.airOpeSch.OPE2_mod,
this.controlOperationModCodesStrs[1] );
        this.controlOperationModCodes[2] = new ControlOperationModCode( this.airOpeSch.OPE3_mod,
```

```
this.controlOperationModCodesStrs[2] );

      //private final String SPEC_isPeakCut      = "ピークカットする";//false;//
   // this.isPeakCut = spec.getSpecValue( this.SPEC_isPeakCut, false );

      //SPEC_isChange_PC    = "ピークカットの目標値を月別設定する";
      this.isChange_PC = spec.getSpecValue( this.SPEC_isChange_PC, false );

      //SPEC_targetPower_PC = "ピークカット目標値月別[kW]";
      str = spec.getSpecValue( this.SPEC_targetPower_PC, "13 1 2 3 4 5 6 7 8 9 10 11 12" );
      strs = null;
      str = str.replace( '_', ' ' );//20120628nino
      strs = str.split( " " );
      int len = strs.length;
      for( int i=0; i<13; i++ ){
         //System.out.println( " " +  Double.parseDouble( strs[i]) );
         if( len > i ){
            this.targetPower_PC[i] = Double.parseDouble( strs[i] ) * 1000.;//[kW]-->[W]
         }else{
            this.targetPower_PC[i] = Double.parseDouble( strs[1] ) * 1000.;//[kW]-->[W]
         }
      }

      //SPEC_ZeroRemains  = "使い切り制御";//false;//
      //0_なし、1_使い切り制御（前詰め）、2_使い切り制御（平均）
      this.zeroRemains = spec.getSpecValue( this.SPEC_ZeroRemains, "0_なし" );
      if( this.zeroRemains.equals( "0_なし") ){
         this.zeroRemainsType = 0;
      }else if( this.zeroRemains.equals( "1_使い切り制御（前詰め）")){
         this.zeroRemainsType = 1;
      }else if( this.zeroRemains.equals( "2_使い切り制御（平均）")){
         this.zeroRemainsType = 2;
      }else{
         this.zeroRemainsType = 0;
      }

      //SPEC_startZRemains  = "使い切り制御開始時刻[-]";
      this.startZRemains = spec.getSpecValue( this.SPEC_startZRemains, "17:00" );

      this.startZRemainsH = Integer.parseInt( this.startZRemains.split( ":" )[0] );
      this.startZRemainsM = Integer.parseInt( this.startZRemains.split( ":" )[1] );

      //SPEC_secondsZRemains= "使い切り制御の時間[s]";
      this.secondsZRemains = spec.getSpecValue( this.SPEC_secondsZRemains, 60. );//[s]

      //SPEC_opeSetRate         = "商用充電上限充足率[-]";
      this.opeSetRate = spec.getSpecValue( this.SPEC_opeSetRate, 0.5 );

      //SPEC_isChangeConstPower = "発電電力平準化（一定値）制御の目標値を月別設定する";
      this.isChangeConstPower = spec.getSpecValue( this.SPEC_isChangeConstPower, false );

      //String SPEC_targetConstPower   = "発電電力平準化（一定値）制御の目標値月別[kW]";
      str = spec.getSpecValue( this.SPEC_targetConstPower, "13 1 2 3 4 5 6 7 8 9 10 11 12" );
      str = str.replace( '_', ' ' );//20120628nino
      strs = str.split( " " );
      len = strs.length;
      for( int i=0; i<13; i++ ){
         //System.out.println( " " +  Double.parseDouble( strs[i]) );
         if( len > i ){
            this.targetConstPower[i] = Double.parseDouble( strs[i] ) * 1000.;//[kW]-->[W]
         }else{
            this.targetConstPower[i] = Double.parseDouble( strs[1] ) * 1000.;//[kW]-->[W]
         }
      }

   }


   public void initialize(IBestStateMessage stateNodes,
         IBestControlMessage commandNodes, IBestRecordMessage recordNodes,
         IBestDomainSchedule schedule) {
      super.sm = stateNodes   ;     //状態ノードを取得
```

```
        super.cm = commandNodes ;     //制御ノードを取得
        super.rm = recordNodes  ;     //記録ノードを取得

        //S_NODE_valInSTRate
        this.valInEChargeRate = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valInEChargeRate);

        //private BestValue valOutEChargeRate = null;//"LO_valOutEChargeRate";//蓄電率
        this.valOutEChargeRate = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valOutEChargeRate );
        this.valOutEDischargeRate = BestValue.bindnode( super.transferMapState, super.sm,
this.S_NODE_valOutEDischargeRate );
        this.valOutTargetW        = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valOutTargetW );
        this.valOutZeroRate       = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valOutZeroRate );
        this.valOutOpeSetRate     = BestValue.bindnode( super.transferMapState, super.sm, this.S_NODE_valOutOpeSetRate );

        super.cm.setCommand( super.getConnectionNode(this.C_NODE_swcIn), 0 );
        super.cm.setCommand( super.getConnectionNode(this.C_NODE_modIn), 0 );
        super.cm.setCommand( super.getConnectionNode(this.C_NODE_modOut), 0 );

        super.cm.setCommand( super.getConnectionNode(this.C_NODE_swcOutEleDischarge), 0 );
        super.cm.setCommand( super.getConnectionNode(this.C_NODE_swcOutEleCharge), 0 );


        for( int i=0; i<this.dayload2.length; i++ ){
           this.dayload2[i] =  new LinkedList<Double>();
           this.dayload2[i].add( 0. );
           this.dayload2[i].add( 0. );
           this.dayload2[i].add( 0. );
           this.dayload2[i].add( 0. );
        }
        for( int i=0; i<this.dayloadHS.length; i++ ){
           this.dayloadHS[i] =  new LinkedList<Double>();
           this.dayloadHS[i].add( 0. );
           this.dayloadHS[i].add( 0. );
           this.dayloadHS[i].add( 0. );
           this.dayloadHS[i].add( 0. );
        }

        //グラフ表示の準備
        if( this.isGVisible ){

           this.gData = new GraphCommonData2015[12];

           int i = 0;

           this.gData[i++] = new GraphCommonData2015( "swcIn[-]", this.maxItemCount,
                 -1, 40, 0 );
           this.gData[i++] = new GraphCommonData2015( "modIn[-]", this.maxItemCount,
                 -1, 0.03, 0 );

           this.gData[i++] = new GraphCommonData2015( "swcOutMain[-]", this.maxItemCount,
                 0, 1000, 1 );
           this.gData[i++] = new GraphCommonData2015( "swcOutCharge[-]", this.maxItemCount,
                 0, 1000, 1 );
           this.gData[i++] = new GraphCommonData2015( "swcOutDischarge[-]", this.maxItemCount,
                 0, 1000, 1 );
           this.gData[i++] = new GraphCommonData2015( "modOut[-]", this.maxItemCount,
                 0, 1000, 1 );

           this.gData[i++] = new GraphCommonData2015( "V_valInEChargeRate[-]", this.maxItemCount,
                 0, 1000, 2 );
           this.gData[i++] = new GraphCommonData2015( "V_valOutEChargeRate[-]", this.maxItemCount,
                 0, 1000, 2 );
           this.gData[i++] = new GraphCommonData2015( "V_valOutEDischargeRate[-]", this.maxItemCount,
                 0, 1000, 2 );
           this.gData[i++] = new GraphCommonData2015( "V_valOutOpeSetRate[-]", this.maxItemCount,
                 0, 1000, 2 );
           this.gData[i++] = new GraphCommonData2015( "V_valOutZeroRate[-]", this.maxItemCount,
                 0, 1000, 2 );

           this.gData[i++] = new GraphCommonData2015( "V_valOutTargetW[W]", this.maxItemCount,
                 0, 1000, 3 );
```

```
        String[] yName = { "swc,modIn[-]","swc,modOut[-]","valIn,Out[-]","Target[W]" };

        gGCJF = new GraphCommonJFrame2015( this.name+"_CtrRB", this.gData, yName );

      }

      //
   /* if( this.loadPredictTypeNum == 4 ){
        //4_負荷計算予測
        String rss;
        String[] rssn;
        String sname =  this.name.concat( "_".concat( this.RECORD_dayload2Sum ) );
        int nnn =0;
        try{
           //System.out.println( "22:00" );

           //20120523nino
           String str = System.getenv( "BEST_RESULT");//work¥Files_ObjectInfo¥Object001¥Result
           String str2 = str.replace( "Result", "simulator" );
           br = new BufferedReader( new FileReader( str2+"¥¥best_result.csv") );
           //br = new BufferedReader( new
FileReader( "work¥¥Files_ObjectInfo¥¥Object001¥¥simulator¥¥best_result.csv") );

           rss = br.readLine();
           rssn = rss.split( "," );

           for( int i=0; i<rss.length(); i++ ){
              if( rssn[i].equals( sname )){
                 nnn= i;
                 break;
              }
           }

           while(( rss = br.readLine() ) != null ){
              rssn = rss.split( "," );

              if( Integer.parseInt( rssn[3])==22 && Integer.parseInt( rssn[4])==0 ){
                 Integer dateNumber = Integer.parseInt( rssn[0] ) * 10000+ Integer.parseInt( rssn[1] ) *100 +
Integer.parseInt( rssn[2] );
                 Double value = Double.parseDouble( rssn[ nnn ]);
                 loadDataValue.put( dateNumber, value );
                 //System.out.println( "dateNumber="+dateNumber+" 22:00" + rssn[ nnn ] + "/"+value);
              }
           }
        }

        catch( Exception e ){
           System.out.println( "ファイルがありません"+e );

        }
      }
   */
   }

   private int[] dateTime; //0:年、1:月、2:日、3:年間通算日、4:曜日、5:時、6:分、7:秒
   private int dayOfWeek;
   private int nextdayOfWeek; //翌日
   private int timeInterval;

   /**
    * 充電時間帯に全充電された
    */
   private boolean isRBFullCharging = false;//充電時間帯に全充電された

   private LinkedList<Double>[] dayload2  = new LinkedList[9];//２次側の曜日別の日積算負荷[J]
   private LinkedList<Double>[] dayloadHS = new LinkedList[9];//熱源の曜日別の日積算製造熱量[J]


   public void outputs(){
      if(super.sm == null || super.cm == null ){
         this.message.append( "(E) sm cm がない");
         return;
```

```
        }

        //年月日時刻曜日など
        this.dateTime = BestTimeManager.getDateWeatherTime();
        this.dayOfWeek = ScheduleManager.getHolidayManager().getDayOfWeek(); //20080116
        this.nextdayOfWeek = ScheduleManager.getHolidayManager().getDayOfWeekTomorrow();//20090910
        this.timeInterval = BestTimeManager.getCurrentInterval();

        //valInSTRate
        this.valInEChargeRate = (BestValue)super.sm.getState( super.getConnectionNode( this.S_NODE_valInEChargeRate ));



        //前のステップの熱量の計算

        //
        this.setValOutEChargeRate = 0;
        this.setValOutEDischargeRate = 0;


//      if( )

        //20111130
        if( this.isUseThisOpe ){
           //このスケジュールを使用する場合

           int swcOPE1234   = 0;//OPE1,2,3,4の共通フラッグ
           int swcOPE1234Charge = 0;//OPE1,2,3,4STの共通フラッグ

           if( this.dayOfWeek<0 ){//20110127
              this.dayOfWeek = 0;
           }
           if( this.nextdayOfWeek<0 ){
              this.nextdayOfWeek = 0;
           }

           //熱源再起動可能か？

           //放電swc
           int swcDisCharge = 0;

           //充電swc
           int swcCharge = 0;

           //２次側swc
      //   int swc2 = 0;

           this.isPeakShift  = false;
           this.isPeakCut    = false;
           this.isPeakCutALL = false;
           this.isConstPower = false;

           this.modOut[0] = 0;

           //    this.swcMy = Airswc.offON( this.swcMy);//swcMyをoff
      //   this.swcMy2= Airswc.offON( this.swcMy2);//swcMyをoff

           //■■放電
           boolean isOPE123 = false;//OPE1,2,3の放電指定期間か？


           //OPE1～4の該当する放電時間帯の判定
           for( int iope=0; iope<3; iope++ ){
              if( ControlOperationModCode.checkMODsOffOn( this.airOpeSch.controlOperationModCodes[iope].getMyNumber(),
this.dateTime )){
                 isOPE123 = true;
                 switch( iope ){
                    case 0:
                       swcOPE1234 = Airswc.onOPE1( swcOPE1234 );
                       break;
                    case 1:
                       swcOPE1234 = Airswc.onOPE2( swcOPE1234 );
```

```
                    break;
                case 2:
                    swcOPE1234 = Airswc.onOPE3( swcOPE1234 );
                    break;
                default:
                    break;
            }
            if( this.airOpeSch.controlOperationModCodes[iope].getOperationName().equals(
                    "0_停止" ) ){

            }else{
                if( this.airOpeSch.controlOperationModCodes[iope].getOperationName().equals(
                    "1_ピークシフト制御" ) ){
                    //充放電スケジュールの場合
                    this.isPeakShift = true;
                }else if( this.airOpeSch.controlOperationModCodes[iope].getOperationName().equals(
                    "2_ピークシフト＋ピークカット（放電）制御" ) ){
                    this.isPeakShift = true;
                    this.isPeakCut = true;
                }else if( this.airOpeSch.controlOperationModCodes[iope].getOperationName().equals(
                    "3_ピークシフト＋ピークカット（充放電）制御" ) ){
                    this.isPeakShift = true;
                    this.isPeakCutALL = true;
                }else if( this.airOpeSch.controlOperationModCodes[iope].getOperationName().equals(
                    "4_ピークカット（放電）制御" ) ){
                    this.isPeakCut = true;
                }else if( this.airOpeSch.controlOperationModCodes[iope].getOperationName().equals(
                    "5_ピークカット（充放電）制御" ) ){
                    this.isPeakCutALL = true;
                }else if( this.airOpeSch.controlOperationModCodes[iope].getOperationName().equals(
                    "6_発電電力平準化（一定値）制御" ) ){
                    this.isConstPower = true;
                }else if( this.airOpeSch.controlOperationModCodes[iope].getOperationName().equals(
                    "7_出力変動抑制制御" ) ){
                    this.isCtrlPowerChangeRate = true;
                }else if( this.airOpeSch.controlOperationModCodes[iope].getOperationName().equals(
                    "8_負荷追従制御" ) ){
                    this.isLoadFollowingControl = true;
                    this.isPeakCut = true;
                    this.isPeakCutALL = true;
                    this.targetPower_PCeff = 0;//放電時のピークカット目標値を0kWに変更するための係数
                }
                //放電
                if( this.isUseDischargeDailyAnnualSchedule ){
                    //isUseDailyAnnualScheduleを優先してチェック
                    this.setValOutEDischargeRate = this.dasDischarge.getSchedule();
                    if( this.setValOutEDischargeRate > 0 ){
                        swcDisCharge = 1;
                    }else{
                        this.setValOutEDischargeRate = 0;
                    }
                }else{
                    int iend =
this.airOpeSch.controlOperationWeekCodes[iope].getOperationCodes(this.dayOfWeek).getSwcsOffOn().length-1;
                    //放電運転時間の場合
                    if( this.airOpeSch.controlOperationWeekCodes[iope].isSWCon( this.dayOfWeek, this.dateTime, iend,
iend ) ){
                        swcDisCharge = 1;
                        this.setValOutEDischargeRate = 0;
                    }
                }
            }
        }

        //OPE1,2,3の期間でない時
        if( !isOPE123 ){
            //OPE4 未指定期間の場合　のモード指定のチェック
            swcOPE1234 = Airswc.onOPE4( swcOPE1234 );
            if( this.airOpeSch.OPE4_mod.equals( "4_換気" )){
                //OPE4 未指定期間が換気指定の場合
                this.modOut[0] = Airmod.onVENTILATE( this.modOut[0] );
```

```
            }
        }

        //昼間の熱源運転のチェック
        if( swcDisCharge == 1 ){//熱源の場合->換気の時は熱源停止
    //      this.swcMy = Airswc.onON( this.swcMy );
        }else{
    //      this.swcMy = Airswc.onOFF( this.swcMy );
        }


        if( this.isPeakCut || this.isPeakCutALL ){
            int im;
            im = ( this.isChange_PC )?    this.dateTime[1] : 0;
            this.valOutTargetW.setValue( this.targetPower_PC[ im ] * this.targetPower_PCeff );//20170511
        }else if( this.isConstPower ){
            //発電電力平準化（一定値）制御
            int im;
            im = ( this.isChangeConstPower )?   this.dateTime[1] : 0;
            this.valOutTargetW.setValue( this.targetConstPower[ im ] );
        }


        //■■蓄電運転
        boolean isOPE123ST = false;//OPE1,2,3のST指定期間か？
    //  boolean isOPE123STCandD = false;//OPE1,2,3でSTCOOLか？

        for( int iope=0; iope<3; iope++ ){
            if( Airmod.checkMODsOffOn( this.controlOperationModCodes[iope].getModsOffOn(), this.dateTime )){
                isOPE123ST = true;
                //isOPE123 = true;
                switch( iope ){
                    case 0:
                        swcOPE1234Charge = Airswc.onOPE1( swcOPE1234Charge );
                        break;
                    case 1:
                        swcOPE1234Charge = Airswc.onOPE2( swcOPE1234Charge );
                        break;
                    case 2:
                        swcOPE1234Charge = Airswc.onOPE3( swcOPE1234Charge );
                        break;
                    default:
                        break;
                }
                if( this.controlOperationModCodes[iope].getOperationName().equals( "0_停止" ) ){
                    //System.out.println("0_停止");

                }else if( this.controlOperationModCodes[iope].getOperationName().equals( "1_ピークシフト制御" ) ){
                    //充放電スケジュールの場合
                    //System.out.println("1_ピークシフト制御");
                    this.isPeakShift = true;
                }else if( this.controlOperationModCodes[iope].getOperationName().equals( "2_ピークシフト＋ピークカット（放
電）制御" ) ){
                    //System.out.println("2_ピークシフト＋ピークカット（放電）制御");
                    this.isPeakShift = true;
                    this.isPeakCut = true;
                }else if( this.controlOperationModCodes[iope].getOperationName().equals( "3_ピークシフト＋ピークカット（充
放電）制御" ) ){
                    //System.out.println("3_ピークシフト＋ピークカット（充放電）制御");
                    this.isPeakShift = true;
                    this.isPeakCutALL = true;
                }else if( this.controlOperationModCodes[iope].getOperationName().equals( "4_ピークカット（放電）制御" ) ){
                    //System.out.println("4_ピークカット（放電）制御");
                    this.isPeakCut = true;
                }else if( this.controlOperationModCodes[iope].getOperationName().equals( "5_ピークカット（充放電）制御
" ) ){
                    //System.out.println("5_ピークカット（充放電）制御");
                    this.isPeakCutALL = true;
                }else if( this.controlOperationModCodes[iope].getOperationName().equals( "6_発電電力平準化（一定値）制御
" ) ){
                    //System.out.println("6_発電電力平準化（一定値）制御");
                    this.isConstPower = true;
```

```
                }else if( this.controlOperationModCodes[iope].getOperationName().equals( "7_出力変動抑制制御" ) ){
                   //System.out.println("7_出力変動抑制制御");
                   this.isCtrlPowerChangeRate = true;
                }else if( this.controlOperationModCodes[iope].getOperationName().equals( "8_負荷追従制御" ) ){
                   //System.out.println("8_負荷追従制御");
                   this.isLoadFollowingControl = true;
                   this.isPeakCut = true;
                   this.targetPower_PCeff = 0;
                }

                if( this.isPeakShift || this.isConstPower || this.isPeakCut || this.isPeakCutALL ){
                   //充放電期間の場合
                   //充電
                   if( this.isUseChargeDailyAnnualSchedule ){
                      //isUseDailyAnnualScheduleを優先してチェック
                      this.setValOutEChargeRate = this.dasCharge.getSchedule();
                      //System.out.println("setValOutEChargeRate="+setValOutEChargeRate);
                      if( this.setValOutEChargeRate > 0 ){
                         this.message.append( "(C)swcCharge=1" );
                         swcCharge = 1;
                      }else{
                         this.setValOutEChargeRate = 0;
                      }

                   }else{
                      //充電運転時間の場合
                      if( this.airOpeSch.controlOperationWeekCodes[ iope ].isSWCon( this.dayOfWeek, this.dateTime, 0,
0 ) ){
                         //当日の充電運転時間帯である場合
                         if( this.airOpeSch.controlOperationWeekCodes[ iope ].isOverlap( this.dayOfWeek, 0 ) ){
                            //２日にオーバーラップしている場合
                            if( this.dateTime[5]*60+this.dateTime[6] <

   this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.dayOfWeek).getSwcsOffOn()[0][1][0] * 60
                                  +
this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.dayOfWeek).getSwcsOffOn()[0][1][1] ){
                               //時刻が終了時刻より前の場合　充電運転
                               swcCharge = 1;
                               this.setValOutEChargeRate = 0;
                            }
                         }else{
                            //オーバーラップしていない場合
                            if( this.dateTime[5]*60+this.dateTime[6] >=

   this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.dayOfWeek).getSwcsOffOn()[0][0][0] * 60
                                  +
this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.dayOfWeek).getSwcsOffOn()[0][0][1] &&
                                   this.dateTime[5]*60+this.dateTime[6] <

   this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.dayOfWeek).getSwcsOffOn()[0][1][0] * 60
                                  +
this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.dayOfWeek).getSwcsOffOn()[0][1][1]){
                               //時刻が開始時刻と終了時刻の間の場合
                               swcCharge = 1;
                               this.setValOutEChargeRate = 0;
                               //20111213nino
                               if( this.dateTime[5]*60+this.dateTime[6] ==

   this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.dayOfWeek).getSwcsOffOn()[0][0][0] * 60
                                    +
this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.dayOfWeek).getSwcsOffOn()[0][0][1]     ){
                                  //充電開始時刻の時
                               // this.isTimeSTLoadPredict = true;
                               }
                            }
                         }
                      }
                      if( this.airOpeSch.controlOperationWeekCodes[ iope ].isSWCon( this.nextdayOfWeek, this.dateTime, 0,
0 ) &&
                           this.airOpeSch.controlOperationWeekCodes[ iope ].isOverlap( this.nextdayOfWeek, 0 ) ){
                         //翌日が充電運転時間であり２日にオーバーラップしている場合
                         if( this.dateTime[5]*60+this.dateTime[6] >=
```

```
   this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.nextdayOfWeek).getSwcsOffOn()[0][0][0] * 60
                              +
this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.nextdayOfWeek).getSwcsOffOn()[0][0][1] ) {
                           //時刻が充電開始時刻より後の場合
                           swcCharge = 1;
                           this.setValOutEChargeRate = 0;
                           //20111213nino
                           if( this.dateTime[5]*60+this.dateTime[6] ==

   this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.nextdayOfWeek).getSwcsOffOn()[0][0][0] * 60
                                 +
this.airOpeSch.controlOperationWeekCodes[ iope ].getOperationCodes(this.nextdayOfWeek).getSwcsOffOn()[0][0][1] ) {
                              //充電開始時刻の時
                           // this.isTimeSTLoadPredict = true;
                           }
                        }
                     }
                  }

                  if( swcCharge == 1 ){
                     //充電期間で充電運転時間の時
                     this.modOut[0] = Airmod.onE_CHARGE( this.modOut[0] );//modOutの充電modをon
                     //System.out.println("■■＞充電modOut");
                  }
               }
            }

            if( !isOPE123ST ){
               swcOPE1234Charge = Airswc.onOPE4( swcOPE1234Charge );
               if( this.airOpeSch.OPE4_mod.equals( "4_換気" )){
                  //冷暖房の未指定期間＝中間期/換気が換気指定の場合
                  this.modOut[0] = Airmod.onVENTILATE( this.modOut[0] );
                  if( this.airOpeSch.controlOperationWeekCodes[ 2 ].isSWCon( this.nextdayOfWeek, this.dateTime ) ){
                  // swcV = 1;
                  }
               }
            }
         }

         //全蓄熱となった場合のフラグ
         if( this.isCoolLimitTempHSin ){
            if( swcCharge == 1 ){
               this.isRBFullCharging = true;
            }
         }else{
            if( swcCharge != 1 ){
               this.isRBFullCharging = false;
            }
         }

         //蓄熱運転のチェック
         if( ( swcCharge == 1 && !this.isRBFullCharging ) ){//熱源の場合->換気の時は熱源停止
   //       this.swcMyST = Airswc.onON( this.swcMyST );
         }else{
   //       this.swcMyST = Airswc.onOFF( this.swcMyST );
         }

         /*****************************************************************************/


         //
         //蓄熱開始時刻でチェック用データの更新

         //前回の運転時間数による簡易制御

         //再起動停止時間によるチェック

         //限界温度　によるチェック

         //OPE
         this.swcOutMain      = swcOPE1234 | swcOPE1234Charge;
```

```
        this.swcOutCharge    = swcOPE1234 | swcOPE1234Charge;
        this.swcOutDischarge = swcOPE1234;

        //
        this.swcOutMain  = ( swcCharge==1 || swcDisCharge==1 )?
                                Airswc.onON( this.swcOutMain) : Airswc.offON(this.swcOutMain);
        this.swcOutCharge    = ( swcCharge==1 )? Airswc.onON(this.swcOutCharge) : Airswc.offON(this.swcOutCharge);
        this.swcOutDischarge = ( swcDisCharge==1 )? Airswc.onON(this.swcOutDischarge) :
Airswc.offON(this.swcOutDischarge);
        //
      // this.swcOutCharge = Airswc.onREMOTE( this.swcOutCharge );//遠隔操作

      // System.out.println( "BestTimeManager.getCurrentInterval()"+seconds + " secondsWorkedC="+secondsWorkedC+ "
secondsWorkedDirectC="+secondsWorkedDirectC+ " secondsWorkedChargeC=" +secondsWorkedChargeC
      //       +"    secondsWorkingC="+secondsWorkingC + " secondsWorkingDirectC="+secondsWorkingDirectC+ "
secondsWorkingChargeC="+secondsWorkingChargeC);

        this.modOut[0] = ( this.isPeakShift )? Airmod.onPEAKSHIFT( this.modOut[0]) :
Airmod.offPEAKSHIFT( this.modOut[0]);
        this.modOut[0] = ( this.isPeakCut )? Airmod.onPEAKCUT( this.modOut[0]) : Airmod.offPEAKCUT( this.modOut[0]);
        this.modOut[0] = ( this.isPeakCutALL )? Airmod.onPEAKCUTALL( this.modOut[0]) :
Airmod.offPEAKCUTALL( this.modOut[0]);
        this.modOut[0] = ( this.isConstPower )? Airmod.onCONSTPOWER( this.modOut[0]) :
Airmod.offCONSTPOWER( this.modOut[0]);
        this.modOut[0] = ( this.isCtrlPowerChangeRate )? Airmod.onSMOOTH( this.modOut[0]) :
Airmod.offSMOOTH( this.modOut[0]);
        this.modOut[0] = ( this.isLoadFollowingControl )? Airmod.onLOAD_FOLLOWING_CONTROL( this.modOut[0]) :
Airmod.offLOAD_FOLLOWING_CONTROL( this.modOut[0]);


        super.cm.setCommand( super.getConnectionNode(this.C_NODE_swcOutEleCharge)
              , this.swcOutCharge );
        super.cm.setCommand( super.getConnectionNode(this.C_NODE_swcOutEleDischarge)
              , this.swcOutDischarge );

        //modeOut
        super.cm.setCommand( super.getConnectionNode(this.C_NODE_modOut)
              , this.modOut[0] );

      }else{
        //親からの制御swc,mode
        this.swcIn[0] = super.cm.getCommand( super.getConnectionNode( this.C_NODE_swcIn ));
        this.modIn[0] = super.cm.getCommand( super.getConnectionNode( this.C_NODE_modIn ));
        //swcOut

        super.cm.setCommand( super.getConnectionNode(this.C_NODE_swcOutEleCharge)
              , this.swcOutCharge = this.swcIn[0] );
        super.cm.setCommand( super.getConnectionNode(this.C_NODE_swcOutEleDischarge)
              , this.swcOutDischarge = this.swcIn[0] );

        //modeOut
        super.cm.setCommand( super.getConnectionNode(this.C_NODE_modOut)
              , this.modOut[0] = this.modIn[0] );

        //
        if( Airswc.isON( this.swcOutCharge )){
          this.setValOutEChargeRate = 0.2;
        }else{
          this.setValOutEChargeRate = 0;
        }
        if( Airswc.isON( this.swcOutDischarge )){
          this.setValOutEDischargeRate = 0.2;
        }else{
          this.setValOutEDischargeRate = 0;
        }
      }


      this.valOutEChargeRate.setValue( this.setValOutEChargeRate );
      this.valOutEDischargeRate.setValue( this.setValOutEDischargeRate );

      //使い切り制御     ピークシフトの時のみ適用
```

```
      if( this.zeroRemainsType==1 && this.isPeakShift ){
         //System.out.println( "dateTime[5]="+dateTime[5]+" startZRemainsH="+startZRemainsH+" dateTime[6]="+dateTime[6]+"
startZRemainsM="+startZRemainsM );
         if( this.dateTime[5]== this.startZRemainsH && this.dateTime[6] == this.startZRemainsM ){
            this.secondsCountDown = this.secondsZRemains;
         }else{
            this.secondsCountDown -= this.timeInterval;
         }
         if( this.secondsCountDown > 0 ){
            this.valOutZeroRate.setValue( 1. );
            //System.out.println( " timeInterval="+timeInterval+" secondsCountDown="+secondsCountDown+" valOutZeroRate="+
            //    (this.timeInterval / this.secondsCountDown)) ;
         }else{
            this.valOutZeroRate.setValue( 0 );
         }
      }else if( this.zeroRemainsType==2 && this.isPeakShift ){
         //System.out.println( "dateTime[5]="+dateTime[5]+" startZRemainsH="+startZRemainsH+" dateTime[6]="+dateTime[6]+"
startZRemainsM="+startZRemainsM );
         if( this.dateTime[5]== this.startZRemainsH && this.dateTime[6] == this.startZRemainsM ){
            this.secondsCountDown = this.secondsZRemains;
         }else{
            this.secondsCountDown -= this.timeInterval;
         }
         if( this.secondsCountDown > 0 ){
            this.valOutZeroRate.setValue( this.timeInterval / this.secondsCountDown );
            //System.out.println( " timeInterval="+timeInterval+" secondsCountDown="+secondsCountDown+" valOutZeroRate="+
            //    (this.timeInterval / this.secondsCountDown)) ;
         }else{
            this.valOutZeroRate.setValue( 0 );
         }
      }else{
         this.valOutZeroRate.setValue( 0 );
      }

      this.valOutOpeSetRate.setValue( this.opeSetRate );

      super.sm.setState( super.getConnectionNode( this.S_NODE_valOutEChargeRate), this.valOutEChargeRate );
      super.sm.setState( super.getConnectionNode( this.S_NODE_valOutEDischargeRate), this.valOutEDischargeRate );
      super.sm.setState( super.getConnectionNode( this.S_NODE_valOutTargetW), this.valOutTargetW );
      super.sm.setState( super.getConnectionNode( this.S_NODE_valOutZeroRate), this.valOutZeroRate );
      super.sm.setState( super.getConnectionNode( this.S_NODE_valOutOpeSetRate), this.valOutOpeSetRate );

      //20111208nino
/*    super.sm.setState( super.getConnectionNode(this.S_NODE_valOutSP_T_watOutCH_HS), this.valOutSP_T_watOutCH_HS );
      super.sm.setState( super.getConnectionNode(this.S_NODE_valOutSP_T_watInCH_HS), this.valOutSP_T_watInCH_HS );
      super.sm.setState( super.getConnectionNode(this.S_NODE_valOutCapCtrl), this.valOutCapCtrl );
      super.sm.setState( super.getConnectionNode(this.S_NODE_valOutSP_T_watOutCH_2), this.valOutSP_T_watOutCH_2 );
*/
      if( this.isRecord ){
         this.message.append(
               "(C)swcIn="+swcIn[0]+"/modeIn="+modIn[0] );
      }

      this.swcOut[0] = this.swcOutMain;
      this.swcOut[1] = this.swcOutCharge;
      this.swcOut[2] = this.swcOutDischarge;

      //記録
      //グラフデータ追加
      if( this.isGVisible ){

         double[] gData = new double[12];

      /*
         this.gData[i++] = new GraphCommonData2015( "swcIn[-]", this.maxItemCount,
               -10, 40, 0 );
         this.gData[i++] = new GraphCommonData2015( "modIn[-]", this.maxItemCount,
               -0.01, 0.03, 0 );

         this.gData[i++] = new GraphCommonData2015( "swcOutMain[-]", this.maxItemCount,
               0, 1000, 1 );
         this.gData[i++] = new GraphCommonData2015( "swcOutCharge[-]", this.maxItemCount,
```

```
            0, 1000, 1 );
        this.gData[i++] = new GraphCommonData2015( "swcOutDischarge[-]", this.maxItemCount,
            0, 1000, 1 );
        this.gData[i++] = new GraphCommonData2015( "modOut[-]", this.maxItemCount,
            0, 1000, 1 );

        this.gData[i++] = new GraphCommonData2015( "V_valInEChargeRate[-]", this.maxItemCount,
            0, 1000, 2 );
        this.gData[i++] = new GraphCommonData2015( "V_valOutEChargeRate[-]", this.maxItemCount,
            0, 1000, 2 );
        this.gData[i++] = new GraphCommonData2015( "V_valOutEDischargeRate[-]", this.maxItemCount,
            0, 1000, 2 );
        this.gData[i++] = new GraphCommonData2015( "V_valOutOpeSetRate[-]", this.maxItemCount,
            0, 1000, 2 );
        this.gData[i++] = new GraphCommonData2015( "V_valOutZeroRate[-]", this.maxItemCount,
            0, 1000, 2 );

        this.gData[i++] = new GraphCommonData2015( "V_valOutTargetW[W]", this.maxItemCount,
            0, 1000, 3 );

    */

        int i=0;
        gData[i++] = this.swcIn[0];// 上位からのswcIn[-]
        gData[i++] = this.modIn[0];// 上位からのmodIn[-]

        gData[i++] = this.swcOutMain;// swcOutMain[-]
        gData[i++] = this.swcOutCharge;// swcOutCharge[-]
        gData[i++] = this.swcOutDischarge;// swcOutDisCharge[-]
        gData[i++] = this.modOut[0];// modOut[-]

        gData[i++] = this.valInEChargeRate.getValue();// 蓄電率[-]

        gData[i++] = this.valOutEChargeRate.getValue();// 蓄電率[-]
        gData[i++] = this.valOutEDischargeRate.getValue();// 放電率[-]
        gData[i++] = this.valOutOpeSetRate.getValue();// 平準化時の商用充電上限充足率[-]
        gData[i++] = this.valOutZeroRate.getValue();// 使い切りのための時間比例で求めた放電量率[-]

        gData[i++] = this.valOutTargetW.getValue();// 目標値[W]

        this.gGCJF.addData( BestTimeManager.getDateWeatherTime(), gData );

    }

    //記録ノード
    if( this.isRecord  && super.rm != null ){
        this.record();
    }

    message.setLength(0);
}

/**
 * 記録
 */
private void record(){
    //記録
    if( super.rm != null ){
    if( CheckPrintModule.isPrintMessage ){
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
            this.RECORD_Message, this.name, this.message.toString() );
    }

    if( CheckPrintModule.isPrintStateIn ){
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
            this.RECORD_swcIn, this.name, this.swcIn[0] );
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
            this.RECORD_modIn, this.name, this.modIn[0] );
        super.rm.setRecord(super.getConnectionNode(this.R_NODE),
            this.RECORD_valInEChargeRate, this.name, this.valInEChargeRate.getValue() );
    }
```

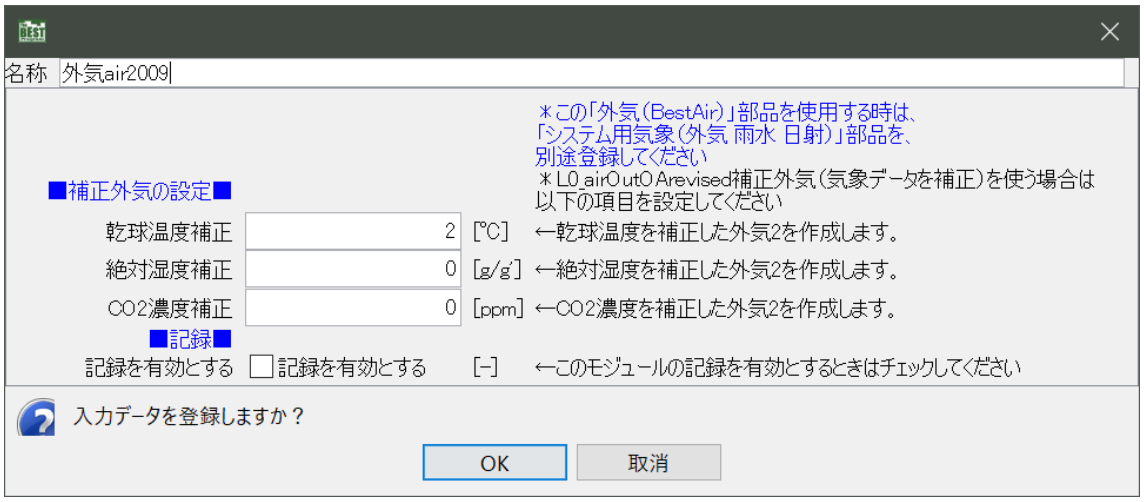
```
        if( CheckPrintModule.isPrintStateOut ){
           super.rm.setRecord(super.getConnectionNode(this.R_NODE),
              this.RECORD_swcOutMain, this.name, this.swcOutMain );
           super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                 this.RECORD_swcOutECharge, this.name, this.swcOutCharge );
           super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                 this.RECORD_swcOutEDischarge, this.name, this.swcOutDischarge );
           super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                 this.RECORD_modOut, this.name, this.modOut[0] );
           super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                 this.RECORD_valOutEChargeRate, this.name, this.valOutEChargeRate.getValue() );
           super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                 this.RECORD_valOutEDischargeRate, this.name, this.valOutEDischargeRate.getValue() );
           super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                 this.RECORD_valOutOpeSetRate, this.name, this.valOutOpeSetRate.getValue() );
           super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                 this.RECORD_valOutTargetW, this.name, this.valOutTargetW.getValue() );
           super.rm.setRecord(super.getConnectionNode(this.R_NODE),
                 this.RECORD_valOutZeroRate, this.name, this.valOutZeroRate.getValue() );
        }

        //if( CheckPrintModule.isPrintStateMy ){
        //}

        }

    }

    public void update() {
    }

}
```

「外気 air」（場所：設備 2015／媒体境界条件 2015／）

モジュール名	外気 air
クラス	OutsideAirModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、上位の設備モジュールから受け取った BestAir 媒体のフィールド変数のうち、乾球温度、絶対湿度および CO2 濃度を入力値で補正したものを下位の設備モジュールへ渡すものである。

例えば、分散型熱源の室外機置き場などで、室外機の冷房時の吸い込み外気温度が気象データの値より高くなるショートサーキット問題を、補正した外気で計算することで能力不足の発生の事前検討に利用できる。

モジュール名が「外気 air」となっているが、上位モジュールからの接続 air 媒体を「外気」に限定するものではない。空調機の給気や還気に接続すれば、給気や還気の air の状態を補正することができる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

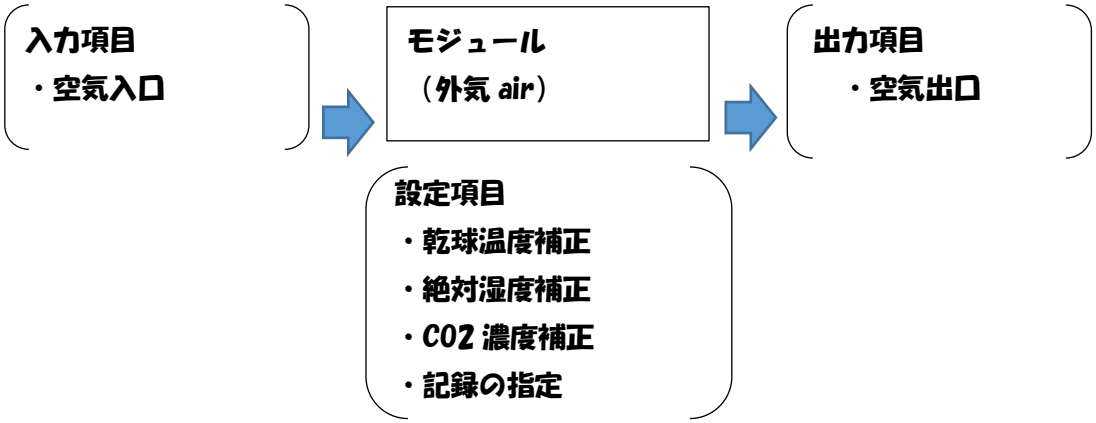


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	乾球温度補正	double	addTemp	2	[℃]	ー	ー		←乾球温度を補正した外気 2 を作成します。
2	絶対湿度補正	double	addHumi	0	[g/g’]	ー	0		←絶対湿度を補正した外気 2 を作成します。
3	CO2 濃度補正	double	addCO2ppm	0	[ppm]	ー	0		←CO2 濃度を補正した外気 2 を作成します。
4	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図 1 における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	LO_recOut	R_NODE	-	記録	メモリ	出口	空気の入口
2	空気入口	LO_airInOA	airInOA	-	状態	空気	入口	空気の入口
3	空気出口	LO_airOutOA	airOutOA	-	状態	空気	出口	空気の出口
4	補正空気出口	LO_airOutOArevised	airOutOArevised	-	状態	空気	出口	補正空気の出口

（6）記録項目

No.	項目	出力内容	単位	分類
1	OA_Message#-#-	メッセージ	ー	メッセージ
2	OA_乾球温度#℃#温度	LO_airOutOA の乾球温度	℃	出口
3	OA_絶対湿度#g/g’#絶対湿度	LO_airOutOA 絶対湿度	g/g ‘	出口
4	OA_CO2 濃度#ppm#濃度	LO_airOutOA CO2 濃度	ppm	出口
5	OA_補正後乾球温度#℃#温度	LO_airOutOArevised の乾球温度	℃	出口
6	OA_補正後絶対湿度#g/g’#絶対湿度	LO_airOutOArevised の絶対湿度	g/g’	出口
7	OA_補正後 CO2 濃度#ppm#濃度	LO_airOutOArevised の CO2 濃度	ppm	出口

（7）計算フロー・計算内容

・L0_airOutOA の空気

入口接続ノード L0_airInOA からの空気の状態値を、出口接続ノード L0_airOutOA の空気の状態値として渡す。

・L0_airOutOArevised の補正空気

入口接続ノード L0_airInOA からの空気の状態値に、ユーザー入力の補正値を加算したものを出口接続ノード L0_airOutOArevised の空気の状態値として渡す。

airOutOArevised 補正出口の乾球温度[°C]

＝　airInOA 入口の乾球温度[°C]　＋　乾球温度補正[°C]

airOutOArevised 補正出口の絶対湿度[g/g']

＝　airInOA 入口の絶対湿度[g/g']　＋　絶対湿度補正[°C]

airOutOArevised 補正出口の CO2 濃度[°C]

＝　airInOA 入口の CO2 濃度[ppm]　＋　CO2 濃度補正[ppm]

その他の補正空気のフィールド変数は、補正後の乾球温度と絶対湿度と空気線図関数から値が更新される。

（8）データ範囲と範囲外の取扱い

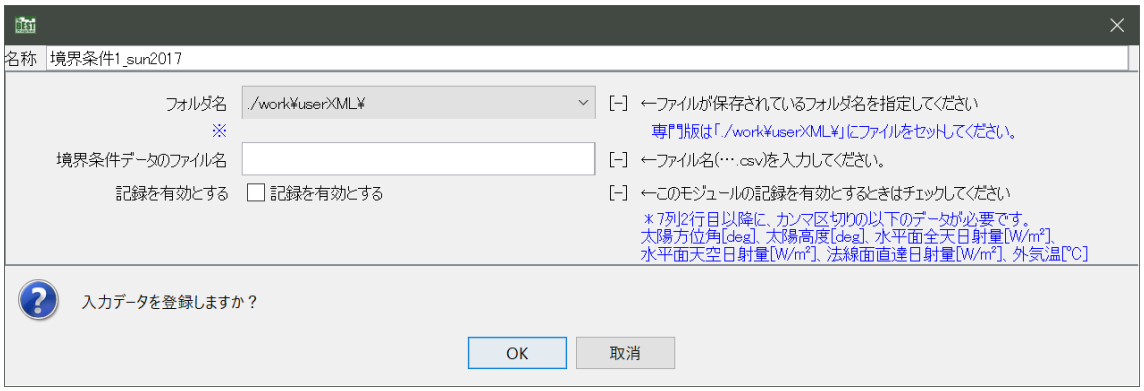
特になし。

「境界条件 1_sun2017」（場所：設備 2015／媒体境界条件 2015／）

モジュール名	境界条件 1_sun2017
クラス	FileReadPrintxSun1Module201701

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、外部ファイル定義の外気と太陽日射情報を BestAir 媒体と BestSun 媒体に取り込み、設備モジュールへ渡すものである。

取り込める外気と太陽日射データは 1 セット分である。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

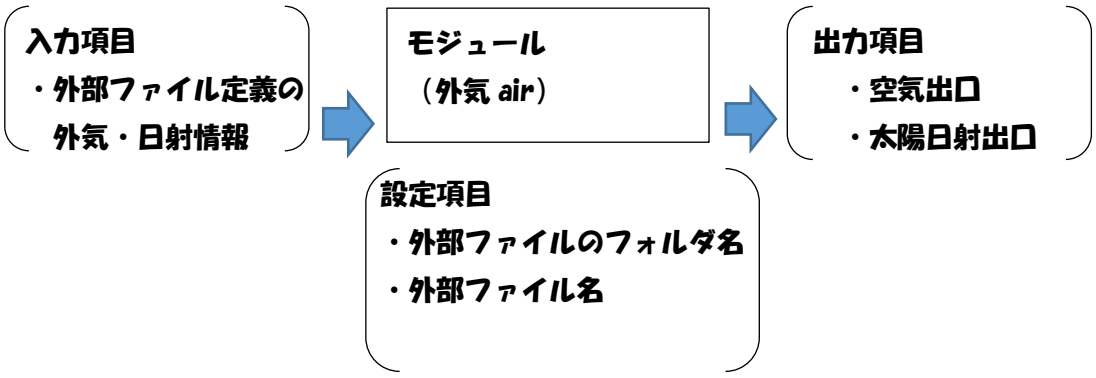


図 1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	フォルダ名	String	folderName	-	[-]	ー	ー		←ファイルが保存されているフォルダ名を指定してください
2	境界条件データのファイル名	String	fileName	-	[-]	ー	ー		←ファイル名(….csv)を入力してください。

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	太陽日射出口	L0_sunOut	sunOut1	-	状態	太陽日射	出口	太陽日射の出口
3	空気出口	L0_airOut	airOut1	-	状態	空気	出口	空気の出口

（6）記録項目

No.	項目	出力内容	単位	分類
1	太陽方位角#deg#-	太陽方位角	Deg	出口
2	太陽高度#deg#-	太陽高度	Deg	出口
3	水平面全天日射量#W/m2#-	水平面全天日射量	W/m2	出口
4	水平面天空日射量#W/m2#-	水平面天空日射量	W/m2	出口
5	法線面直達日射量#W/m2#-	法線面直達日射量	W/m2	出口
6	気温#℃#-	気温	℃	出口
7	絶対湿度#g/g'#-	絶対湿度	g/g'	出口

（7）計算フロー・計算内容

・L0_sunOut の太陽日射媒体

外部ファイルに定義された「太陽方位角」、「太陽高度」、「水平面全天日射量」、「水平面天空日射量」、「法線面直達日射量」を、計算ステップごとに値を取得し L0_sunOut の太陽日射媒体の状態値として渡す。

sunOut の太陽方位角

＝　外部ファイルの 7 列目の値[deg]

sunOut の太陽高度

＝　外部ファイルの 8 列目の値[deg]

sunOut の水平面全天日射量

＝　外部ファイルの 9 列目の値[W/m2]

sunOut の水平面天空日射量

＝　外部ファイルの 10 列目の値[W/m2]

sunOut の水平面直達日射量

＝　外部ファイルの 11 列目の値[W/m2]

・L0_airOut の空気媒体

外部ファイルに定義された「乾球温度」と「絶対湿度」を、計算ステップごとに値を取得し L0_airOut の空気媒体の状態値として渡す。

airOut の乾球温度[°C]

＝　外部ファイルの 12 列目の値[°C]

airOut の絶対湿度[g/g']

＝　外部ファイルの 13 列目の値[g/g']

CO2 濃度以外の空気のフィールド変数は、乾球温度と絶対湿度と空気線図関数から値がセットされる。

☞　値が数値でない時は、値＝ 0 をセットする。

☞　外部ファイルの定義データの行数が計算期間の計算ステップ数に満たない場合、オーバーした計算ステップでは sunOut と airOut のフィールド変数にはすべて値=0 をセットする。

なお、外部ファイルの 1 から 6 列目は順に「年」「月」「日」「時」「分」「曜日」の 6 項目である。

（8）データ範囲と範囲外の取扱い

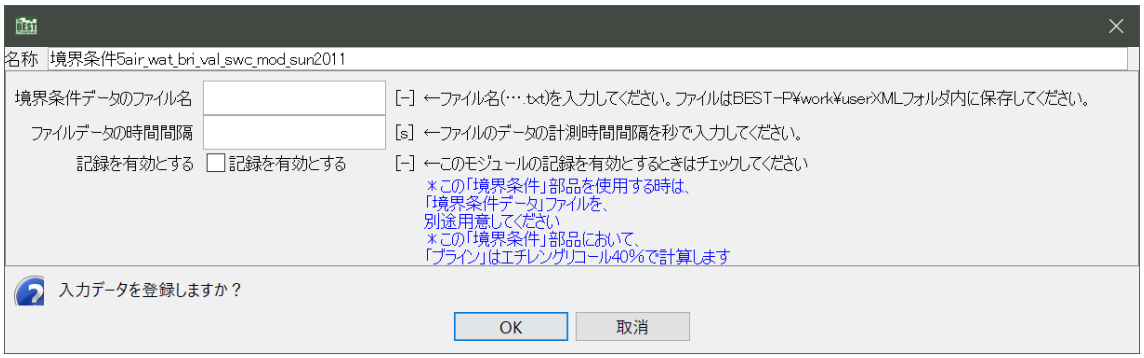
特になし。

「境界条件 5air_wat_bri_val_swc_mod_sun2011」(場所：設備 2015／媒体境界条件 2015／)

モジュール名	境界条件 5air_wat_bri_val_swc_mod_sun2011
クラス	FileReadPrintx5Module20110601

（1）入力画面

・スペック



・「境界条件データのファイル名」は、使用する実測データなどのファイル名を入力する。このファイルは csv 形式のテキストデータファイルとし、ファイルは「¥work¥userXML」フォルダへ保存しておく。ファイル名は拡張子「.csv」を含めて入力する。

・「ファイルデータの時間間隔」には、実測した時間間隔を秒単位で入力する。←新項目

例えば、10 分計測データの場合は 600[s]、 1 時間計測の場合は 3600[s]を入力する。

☞計測データは　固定の計測時間間隔であることが条件である。

また、実測データ時間間隔＞＝BEST の計算時間間隔　であることも条件である。

モジュールは、BEST の計算時間間隔に置き換えて値を取り込む。

（２）モジュールの概要

このモジュールは、外部ファイルに csv 形式で用意した空気、水、ブライン、値、on/off 信号、モード信号、日射などを読込み、BestAir、BestWater、BestBrine、BestValue、swc、mod、BestSun 媒体に置き換える。置換えた Best＊＊媒体の値は、接続ノード L0_airOut[*]や L0_watOut[*]などから他のモジュールへ受渡しができる。

各媒体はそれぞれ５個の条件をデータファイルに設定できる。

（３）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	境界条件データのファイル名	String	fileName	-	[-]	ー	ー		←（例：EleData.csv）ファイルは「¥work¥userXML」フォルダへセットする。
2	ファイルデータの時間間隔	int	dataTimeInterval	-	[s]	ー	ー		←ファイルのデータの計測時間間隔を秒で入力する。
3	記録を有効とする	boolean	isRecord	false	[-]	ー	ー		

（5）シーケンス接続（図 1 における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	空気 1-空気 5	L0_airOut1-airOut5	airOut1-airOut5	-	状態	空気	出口	空気の出口 1-5
3	水 1-水 5	L0_watOut1-watOut5	watOut1-watOut5	-	状態	水	出口	水の出口 1-5
4	ブライン 1-ブライン 5	L0_briOut1-briOut5	briOut1-briOut5	-	状態	ブライン	出口	ブラインの出口 1-5
5	値 1-値 5	L0_valOut1-valOut5	valOut1-valOut5	-	状態	値	出口	値の出口 1-5
6	太陽日射 1-太陽日射 5	L0_sunOut1-sunOut5	sunOut1-sunOut5	-	状態	太陽日射	出口	太陽日射の出口 1-5
7	発停 1-発停 5	L1_swcOut1-swcOut5	swcOut1-swcOut5	-	制御	On/Off 信号	出口	発停の出口 1-5
8	モード 1-モード 5	L1_modOut1-modOut5	modOut1-modOut5	-	制御	制御モード	出口	モードの出口 1-5

（6）記録項目

No.	項目	出力内容	単位	分類
1	FRele_Message#-#-	メッセージ	-	メッセージ
2	FRmed 空気* 乾球温度#℃#温度	L0_air0ut* の乾球温度	℃	出口
3	FRmed 空気* 絶対湿度#g/g'#絶対湿度	L0_air0ut* の絶対湿度	g/g'	出口
4	FRmed 空気* 風量#g/s#質量流量	L0_air0ut* の風量	g/s	出口
5	FRmed 水* 温度#℃#温度	L0_wat0ut* の温度	℃	出口
6	FRmed 水* 流量#g/s#質量流量	L0_wat0ut* の流量	g/s	出口
7	FRmed ブライン* 温度#℃#温度	L0_bri0ut* の温度	℃	出口
8	FRmed ブライン* 流量#g/s#質量流量	L0_bri0ut* の流量	g/s	出口
9	FRmed 値*#-#-	L0_val0ut* の値	-	出口
10	FRmed 制御 swc*#-#-	L1_swc0ut* の値	-	出口
11	FRmed 制御 mod*#-#-	L1_mod0ut* の値	-	出口
12	FRmed sun* 太陽方位角#deg#-	L0_sun0ut* の太陽方位角	deg	出口
13	FRmed sun* 太陽高度#deg#-	L0_sun0ut* の太陽高度	deg	出口
14	FRmed sun* 水平面全天日射量#W/m2#-	L0_sun0ut* の水平面全天日射量	W/m2	出口
15	FRmed sun* 水平面天空日射量#W/m2#-	L0_sun0ut* の水平面天空部日射量	W/m2	出口
16	FRmed sun* 法線面直達日射量#W/m2#-	L0_sun0ut* の法線面直達日射量	W/m2	出口
	☞項目名の中の ＊ は 1 から 5 の数値に置き換える	☞接続温―ド名の中の ＊ は 1 から 5 の数値に置き換える		

（7）計算フロー・計算内容

・外部の境界条件データのファイルのフォーマット

フォーマットは次の通りである。

＊データはｃｓｖ形式で次のフォーマットで作成する。

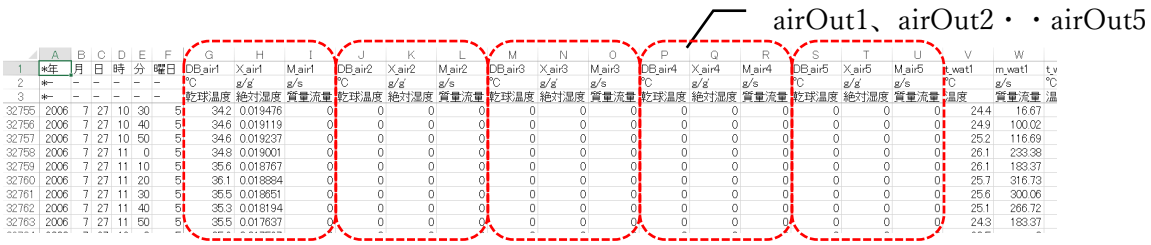
＊１列目の最初の文字が「*」の行は、コメント行として扱う。

＊1～6 列（A～F 列）は自由（推奨例：*年、月、日、時、分、曜日）、7 列目（G 列）以降が媒体のデータです。（A~F 列の値は計算には使用しない）

⇒計算期間を実測データの日時に調整して計算実行する必要がある。

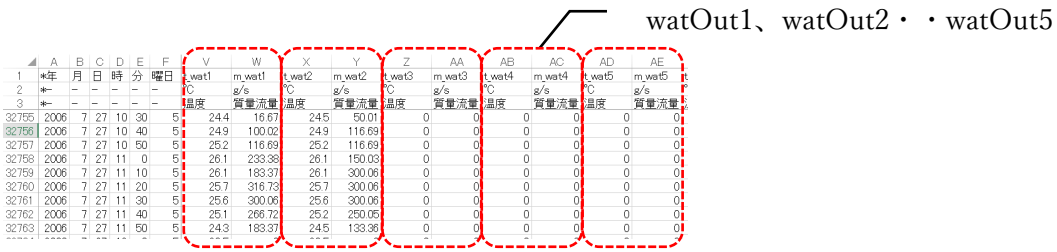
＊G~CC 列の所定の場所に各媒体のデータを用意する。

＊G～U 列：5 個の BestAir 媒体のデータで一つの BestAir 媒体を乾球温度[°C]、絶対湿度[g/g']、質量流量[g/s]で定義し 3 列を使用する。接続ノードは airOut1、airOut2、・・airOut5 である。



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	*年	月	日	時	分	曜日	DB_air1	X_air1	M_air1	DB_air2	X_air2	M_air2	DB_air3	X_air3	M_air3	DB_air4	X_air4	M_air4	DB_air5	X_air5	M_air5	t_wat1	m_wat1
2	*	–	–	–	–	–	°C	g/g	g/s	°C	g/g	g/s	°C	g/g	g/s	°C	g/g	g/s	°C	g/g	g/s	°C	g/s
3	*	–	–	–	–	–	乾球温度	絶対湿度	質量流量	乾球温度	絶対湿度	質量流量	乾球温度	絶対湿度	質量流量	乾球温度	絶対湿度	質量流量	乾球温度	絶対湿度	質量流量	温度	質量流量
32755	2006	7	27	10	30	5	34.2	0.019476	0	0	0	0	0	0	0	0	0	0	0	0	0	24.4	16.67
32756	2006	7	27	10	40	5	34.6	0.019119	0	0	0	0	0	0	0	0	0	0	0	0	0	24.9	100.02
32757	2006	7	27	10	50	5	34.6	0.019237	0	0	0	0	0	0	0	0	0	0	0	0	0	25.2	116.69
32758	2006	7	27	11	0	5	34.8	0.019001	0	0	0	0	0	0	0	0	0	0	0	0	0	26.1	233.38
32759	2006	7	27	11	10	5	35.6	0.018767	0	0	0	0	0	0	0	0	0	0	0	0	0	26.1	183.37
32760	2006	7	27	11	20	5	36.1	0.018884	0	0	0	0	0	0	0	0	0	0	0	0	0	25.7	316.73
32761	2006	7	27	11	30	5	35.5	0.018651	0	0	0	0	0	0	0	0	0	0	0	0	0	25.6	300.06
32762	2006	7	27	11	40	5	35.3	0.018194	0	0	0	0	0	0	0	0	0	0	0	0	0	25.1	266.72
32763	2006	7	27	11	50	5	35.5	0.017637	0	0	0	0	0	0	0	0	0	0	0	0	0	24.3	183.37

＊V～AE 列：5 個の BestWater 媒体のデータで一つの BestWater 媒体を温度[°C]、質量流量[g/s]で定義し 2 列を使用する。接続ノードは watOut1、watOut2、・・watOut5 である。

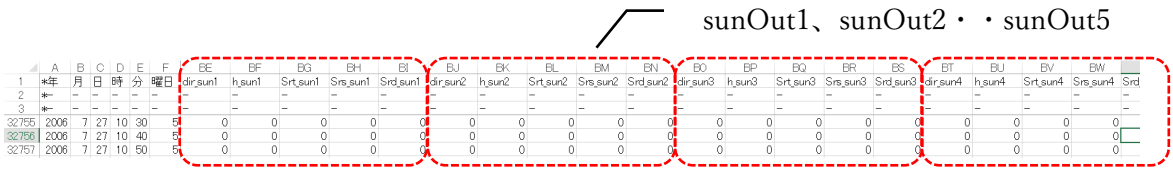


	A	B	C	D	E	F	V	W	X	Y	Z	AA	AB	AC	AD	AE
1	*年	月	日	時	分	曜日	t_wat1	m_wat1	t_wat2	m_wat2	t_wat3	m_wat3	t_wat4	m_wat4	t_wat5	m_wat5
2	*	–	–	–	–	–	°C	g/s	°C	g/s	°C	g/s	°C	g/s	°C	g/s
3	*	–	–	–	–	–	温度	質量流量	温度	質量流量	温度	質量流量	温度	質量流量	温度	質量流量
32755	2006	7	27	10	30	5	24.4	16.67	24.5	50.01	0	0	0	0	0	0
32756	2006	7	27	10	40	5	24.9	100.02	24.9	116.69	0	0	0	0	0	0
32757	2006	7	27	10	50	5	25.2	116.69	25.2	116.69	0	0	0	0	0	0
32758	2006	7	27	11	0	5	26.1	233.38	26.1	150.03	0	0	0	0	0	0
32759	2006	7	27	11	10	5	26.1	183.37	26.1	300.06	0	0	0	0	0	0
32760	2006	7	27	11	20	5	25.7	316.73	25.7	300.06	0	0	0	0	0	0
32761	2006	7	27	11	30	5	25.6	300.06	25.6	300.06	0	0	0	0	0	0
32762	2006	7	27	11	40	5	25.1	266.72	25.2	250.05	0	0	0	0	0	0
32763	2006	7	27	11	50	5	24.3	183.37	24.5	133.36	0	0	0	0	0	0

＊AF～AO 列：5 個の BestBrine 媒体のデータで一つの BestBrine 媒体を温度[℃]、質量流量[g/s]で定義し 2 列を使用する。接続ノードは briOut1、briOut2、・・briOut5 である。

＊AP～AT 列：5 個の BestValue 媒体のデータで一つの BestValue 媒体を値[-]で定義し 1 列を使用する。接続ノードは valOut1、valOut2、・・valOut5 である。
＊AU～AY 列：5 個の swc 媒体のデータで一つの swc 媒体を swc 信号[-]で定義し 1 列を使用する。接続ノードは swcOut1、swcOut2、・・swcOut5 である。
＊AZ～BD 列：5 個の mod 媒体のデータで一つの mod 媒体を mod 信号[-]で定義し 1 列を使用する。接続ノードは modOut1、modOut2、・・modOut5 である。

＊BE～CC 列：5 個の BestSun 媒体のデータで一つの BestSun 媒体を dir 太陽方位角[deg]、h 太陽高度[deg]、Srt 水平面全天日射量[W/m2]、Srs 水平面天空日射量[W/m2]、Srd 法線面直達日射量[W/m2]で定義し 5 列を使用する。接続ノードは sunOut1、sunOut2、・・sunOut5 である。



☞　値が数値でない時は、値＝０をセットする。

☞　外部ファイルの境界条件データの行数が計算期間の計算ステップ数に満たない場合、オーバーした計算ステップでは各媒体のフィールド変数は最後に取り込んだ境界条件値のままとなる。

（８）データ範囲と範囲外の取扱い

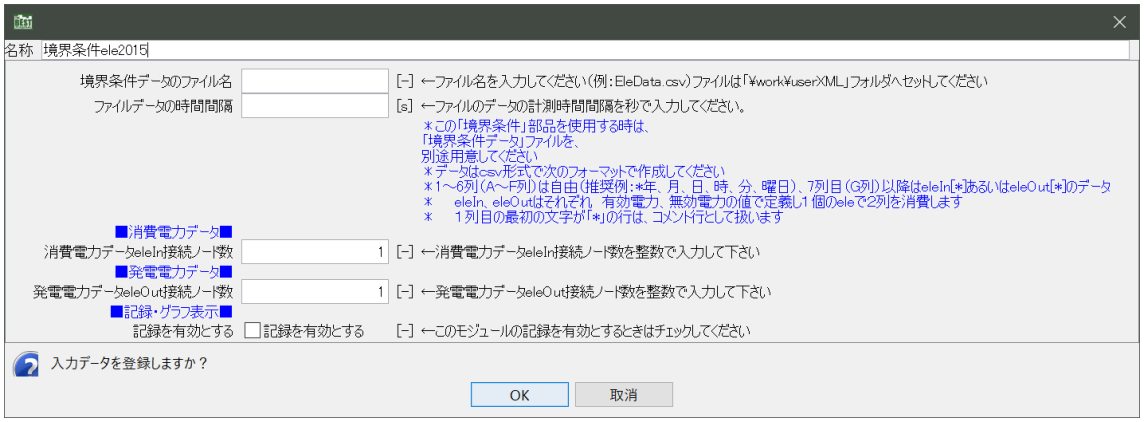
特になし。

「境界条件 ele2015」（場所：設備 2015／媒体境界条件 2015／）

モジュール名	境界条件 ele2015
クラス	FileReadPrintElex20Module2015

（1）入力画面

・スペック



・「境界条件データのファイル名」は、実測データのファイル名を入力する。
このファイルは csv 形式のテキストデータファイルとし、ファイルは「¥work¥userXML」フォルダへ保存しておく。ファイル名は拡張子「.csv」を含めて入力する。

・「ファイルデータの時間間隔」には、実測した時間間隔を秒単位で入力する。
例えば、10 分計測データの場合は 600[s]、 1 時間計測の場合は 3600[s]を入力する。
☞計測データは　固定の計測時間間隔であることが条件である。
また、実測データ時間間隔＞＝BEST の計算時間間隔　であることも条件である。
モジュールは、BEST の計算時間間隔に置き換えて値を取り込む。

・実測データファイルから取込み変換利用できる消費電力や発電電力の数は固定とせず自由度を持たせている。境界条件データファイルに用意した消費電力や発電電力の数に応じて、入力画面の「消費電力データ eleIn 接続ノード数」「発電電力データ eleOut 接続ノード数」に入力する。

（2）モジュールの概要

このモジュールは、外部ファイルに csv 形式で用意した消費電力・発電電力の実測値などを読込み、BestElectricity 媒体に置き換える。置き換えた BestElectricity 媒体の値は、接続ノード L0_eleIn[*]や L0_eleOut[*]から他のモジュールへ受渡しができる。

例えば、蓄電池の容量の検討で、実測した負荷側のデータがあればこれを L0_eleIn[*]にセットして蓄電池モジュールと接続することで、実際の負荷をもとに蓄電池の運転状況がシミュレーションできる。

また、実際の太陽光発電や風力発電などで、変動を抑制する必要が生じた場合に、発電量の実測値を L0_eleOut[*]にセットして蓄電池モジュールと接続してシミュレーションすれば、実際の負荷をもとにした変動抑制のために必要な蓄電池容量の検討ができる。

このモジュールでは、消費電力 eleIn と発電電力 eleOut のデータ数（接続ノード数）に制限は無く自由に設定できる。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

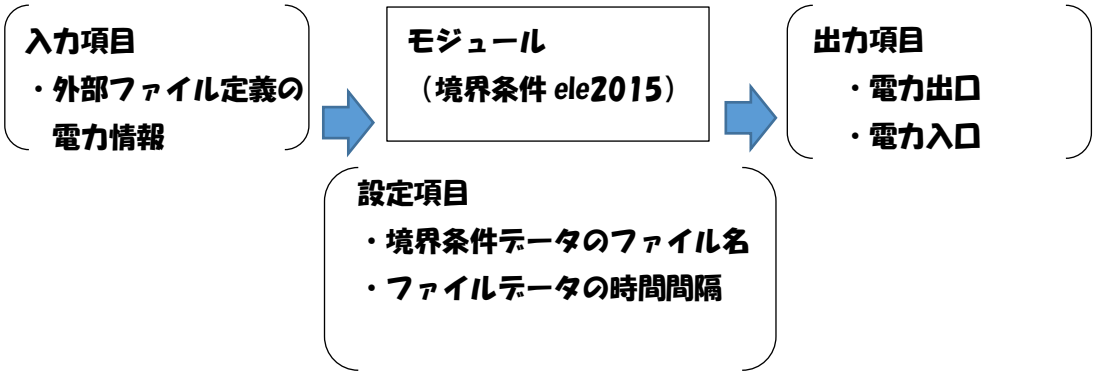


図 1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	ー	ー		
1	境界条件データのファイル名	String	fileName	-	[-]	ー	ー		←（例：EleData.csv）ファイルは「¥work¥userXML」フォルダへセットする。
2	ファイルデータの時間間隔	int	dataTimeInterval	-	[s]	ー	ー		←ファイルのデータの計測時間間隔を秒で入力する。
3	消費電力データ eleIn 接続ノード数	int	numberOfeleIn	1	[-]	ー	0		←消費電力データ eleIn 接続ノード数を整数で入力する。
4	発電電力データ eleOut 接続ノード数	int	numberOfeleOut	1	[-]	ー	0		←発電電力データ eleOut 接続ノード数を整数で入力する。
5	記録を有効とする	boolean	isRecord	false	[-]	ー	ー		

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	消費電力	L0_eleIn	eleIn[*]	-	状態	電力	入口	消費電力の入口（消費電力を上位へ渡す） 消費電力データ eleIn 接続ノード数だけ用意される。
3	発電電力	L0_eleOut	eleOut[*]	-	状態	電力	出口	発電電力の出口（発電を下位へ渡す） 発電電力データ eleOut 接続ノード数だけ用意される。

（6）記録項目

No.	項目	出力内容	単位	分類
1	FRele_Message#-#-	メッセージ	-	メッセージ
2	FRele_L0_eleIn[*]有効電力#W#電力	L0_eleIn[*]の有効電力（消費電力）	W	入口
3	FRele_L0_eleIn[*]無効電力#Var#無効電力	L0_eleIn[*]の無効電力（消費電力）	Var	入口
4	FRele_L0_eleOut[*]有効電力#W#電力	L0_eleIn[*]の有効電力（発電電力）	W	出口
5	FRele_L0_eleOut[*]無効電力#Var#無効電力	L0_eleIn[*]の無効電力（発電電力）	Var	出口

（７）計算フロー・計算内容

・外部の境界条件データのファイルのフォーマット

フォーマットは次の通りである。（入力画面の備考欄にも説明している。）

＊データはｃｓｖ形式で次のフォーマットで作成する。

＊１列目の最初の文字が「*」の行は、コメント行として扱う。

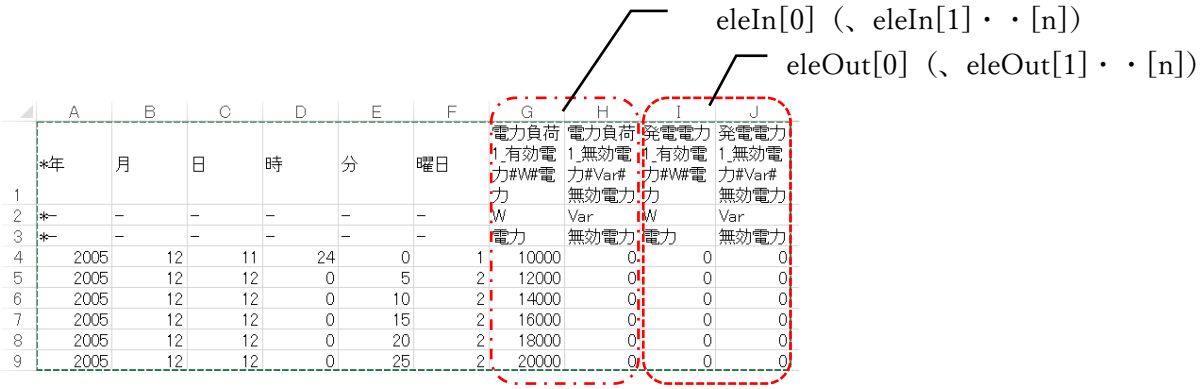
＊1～6 列（A～F 列）は自由（推奨例：*年、月、日、時、分、曜日）、7 列目（G 列）以降は eleIn[*]あるいは eleOut[*]のデータです。（A~F 列の値は計算には使用しない）

⇒計算期間を実測データの日時に調整して計算実行する必要がある。

＊eleIn、eleOut はそれぞれ有効電力、無効電力の二組の値で定義し 1 個の ele で 2 列を消費する。

＊eleIn[0]、eleIn[1]、eleIn[m-1]、eleOut[0]、eleOut[1]、eleOut[n-1]の順でデータを用意する。（m は eleIn の接続ノード数、n は eleOut の接続ノード数）

境界条件データのファイルの例



	A	B	C	D	E	F	G	H	I	J
1	*年	月	日	時	分	曜日	電力負荷1_有効電力#W#電力	電力負荷1_無効電力#Var#無効電力	発電電力1_有効電力#W#電力	発電電力1_無効電力#Var#無効電力
2	*-	-	-	-	-	-	W	Var	W	Var
3	*-	-	-	-	-	-	電力	無効電力	電力	無効電力
4	2005	12	11	24	0	1	10000	0	0	0
5	2005	12	12	0	5	2	12000	0	0	0
6	2005	12	12	0	10	2	14000	0	0	0
7	2005	12	12	0	15	2	16000	0	0	0
8	2005	12	12	0	20	2	18000	0	0	0
9	2005	12	12	0	25	2	20000	0	0	0

7 列目以降がデータ列であるが、消費電力（eleIn 接続ノード数分のデータ）→発電電力（eleOut 接続ノード数分のデータ）の順でデータを作成する。

上記の例は、消費電力データ eleIn 接続ノード数=1、発電電力データ eleOut 接続ノード数=1 の時の境界条件データファイルの例である。7、8 列目（G、H 列）が eleIn[0]、9、10 列目（I、J 列）が eleOut[0]の値である。

・L0_eleIn[0]の電力媒体

外部ファイルに定義された「有効電力」、「無効電力」を、助走計算期間を含む計算ステップごとに値を取得し消費電力入口 L0_eleIn[0]の電力媒体の状態値として渡す。

eleIn[0]の有効電力

＝　外部ファイルの 7 列目（G 列）の値[W]

eleIn[0]の無効電力

＝　外部ファイルの 8 列目（H 列）の値[Var]

・L0_eleIn[0]の電力媒体

外部ファイルに定義された「有効電力」、「無効電力」を、助走計算期間を含む計算ステップごとに値を取得し発電出口 L0_eleOut[0]の電力媒体の状態値として渡す。

eleOut[0]の有効電力

＝　外部ファイルの 9 列目（I 列）の値[W]

eleOut[0]の無効電力

＝　外部ファイルの 10 列目（J 列）の値[Var]

☞　値が数値でない時は、値＝ 0 をセットする。

☞　外部ファイルの定義データの行数が計算期間の計算ステップ数に満たない場合、オーバーした計算ステップでは eleIn[*]と eleOut[*]のフィールド変数にはすべて値=0 をセットする。

（8）データ範囲と範囲外の取扱い

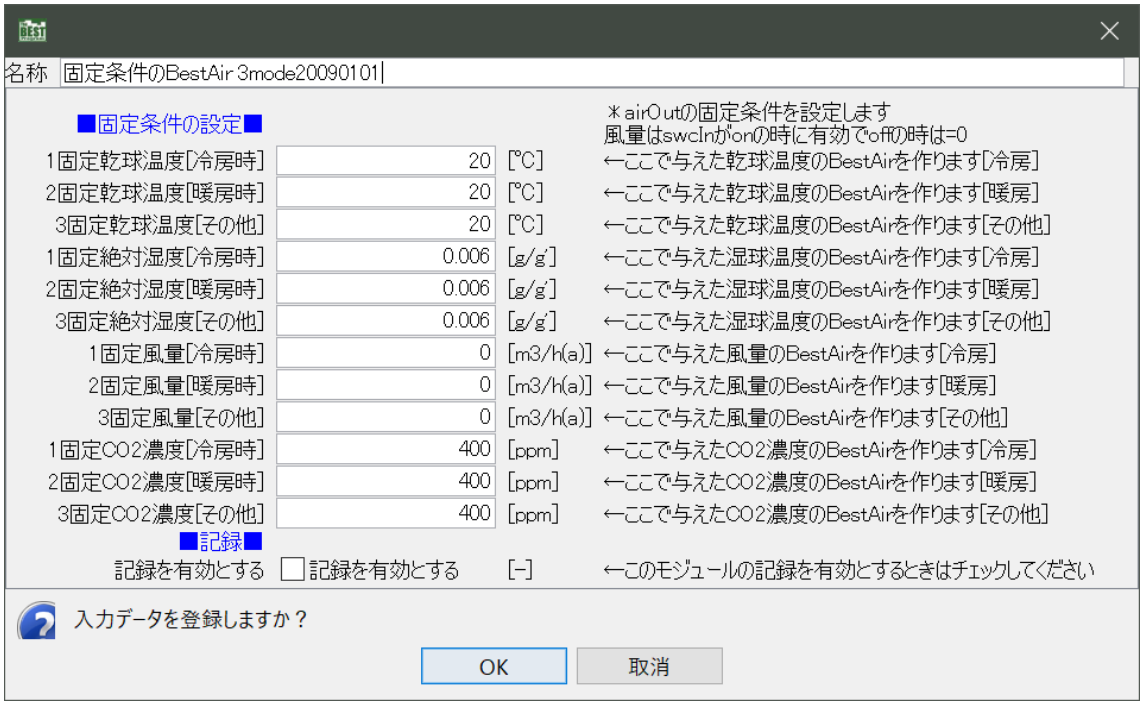
特になし。

「固定条件の BestAir 3mode」（場所：空調・換気設備／媒体 空気 水／）

モジュール名	固定条件の BestAir 3mode
クラス	ConstantAir3modeModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、空気 BestAir 媒体の境界条件を与えるもので、冷房、暖房、中間期の別に設定できるモジュールである。

L1_modIn の mod 信号により、冷房、暖房、中間期は判断し、それぞれの期間に設定された条件の BestAir を作成し、L0_airOut 接続ノードから出力する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

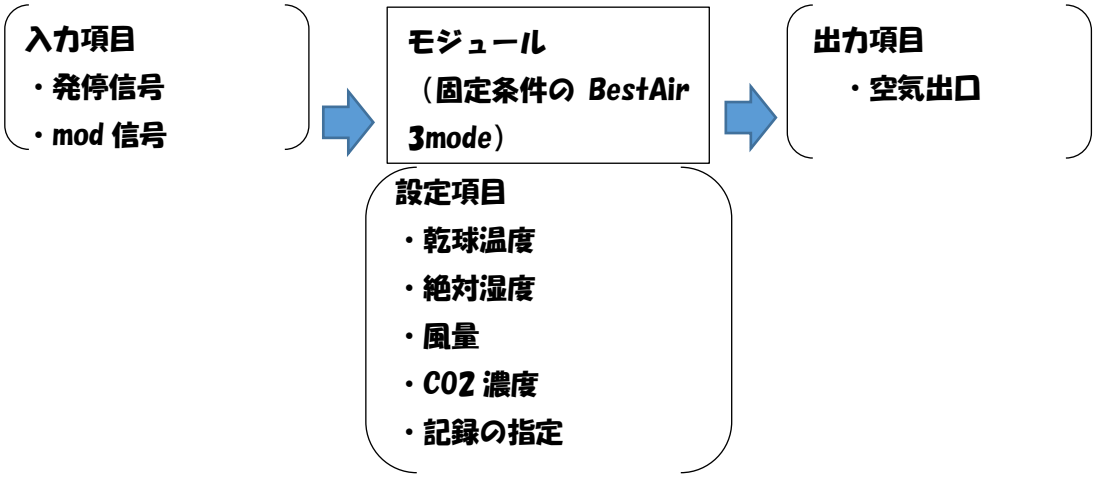


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図1における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	1固定乾球温度[冷房時]	double	c_TempDB[0]	20	[℃]	―	―		←ここで与えた乾球温度のBestAirを作ります[冷房]
2	2固定乾球温度[暖房時]	double	c_TempDB[1]	20	[℃]	―	―		←ここで与えた乾球温度のBestAirを作ります[暖房]
3	3固定乾球温度[その他]	double	c_TempDB[2]	20	[℃]	―	―		←ここで与えた乾球温度のBestAirを作ります[その他]
4	1固定絶対湿度[冷房時]	double	c_Humi[0]	0.006	[g/g']	―	0		←ここで与えた絶対湿度のBestAirを作ります[冷房]
5	2固定絶対湿度[暖房時]	double	c_Humi[1]	0.006	[g/g']	―	0		←ここで与えた絶対湿度のBestAirを作ります[暖房]
6	3固定絶対湿度[その他]	double	c_Humi[2]	0.006	[g/g']	―	0		←ここで与えた絶対湿度のBestAirを作ります[その他]
7	1固定風量[冷房時]	double	c_FlowRate[0]	0	[m3/h(a)]	―	0		←ここで与えた風量のBestAirを作ります[冷房]
8	2固定風量[暖房時]	double	c_FlowRate[1]	0	[m3/h(a)]	―	0		←ここで与えた風量のBestAirを作ります[暖房]
9	3固定風量[その他]	double	c_FlowRate[2]	0	[m3/h(a)]	―	0		←ここで与えた風量のBestAirを作ります[その他]
10	1固定CO2濃度[冷房時]	double	c_CO2ppm[0]	400	[ppm]	―	0		←ここで与えたCO2濃度のBestAirを作ります[冷房]
11	2固定CO2濃度[暖房時]	double	c_CO2ppm[1]	400	[ppm]	―	0		←ここで与えたCO2濃度のBestAirを作ります[暖房]
12	3固定CO2濃度[その他]	double	c_CO2ppm[2]	400	[ppm]	―	0		←ここで与えたCO2濃度のBestAirを作ります[その他]
13	記録を有効とする	boolean	isRecood	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	On/off 信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	Onoff 信号の入口
3	運転モード	L1_modIn	modIn	-	制御	制御モード	入口	制御モードの入口
4	空気出口	L0_airOut	airOut	-	状態	空気	出口	空気の出口

（6）記録項目

No.	項目	出力内容	単位	分類
1	固定 air_Message#-#-	メッセージ	ー	メッセージ
2	固定 air_乾球温度#℃#温度	L0_airOut の乾球温度	℃	出口
3	固定 air_絶対湿度#g/g'#絶対湿度	L0_airOut の絶対湿度	g/g ‘	出口
4	固定 air_質量風量#g/s#質量流量	L0_airOut の質量風量	g/s	出口
5	固定 air_CO2 濃度#ppm#濃度	L0_airOut の CO2 濃度	ppm	出口

（7）計算フロー・計算内容

・L0_airOut の空気

入口接続ノード L1_modIn からの制御モード信号から冷暖房中間期を判定し、それぞれの期間に応じた入力値を BestAir 媒体 airOut に設定し接続ノード L0_airOut から渡す。

L1_modIn の制御信号が冷房の時は以下のようになる。

airOut の乾球温度[℃]

＝　1 固定乾球温度[冷房時] [℃]

airOut の絶対湿度[g/g']

＝　1 固定絶対湿度[冷房時] [g/g']

airOut の CO2 濃度[℃]

＝　1 固定 CO2 濃度[冷房時] [ppm]

風量については、

L1_swcIn の発停信号が運転の時は、

airOut の風量[g/s]

＝　1 固定風量[冷房時] [g/s]

L1_swcIn の発停信号が停止の時は、

airOut の風量[g/s]

＝　0 [g/s]

とする。

その他の補正空気のフィールド変数は、補正後の乾球温度と絶対湿度と空気線図関数から値が更新される。

（8）データ範囲と範囲外の取扱い

特になし。

「固定条件の BestWater 3mode」（場所：空調・換気設備／媒体 空気 水／）

モジュール名	固定条件の BestWater 3mode
クラス	ConstantWater3modeModule20090101

（1）入力画面

・スペック

（2）モジュールの概要

本モジュールは、空気 BestWater 媒体の境界条件を与えるもので、冷房、暖房、中間期の別に設定できるモジュールである。

L1_modIn の mod 信号により、冷房、暖房、中間期を判断し、それぞれの期間に設定された条件の BestWater を作成し、L0_watOut 接続ノードから出力する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

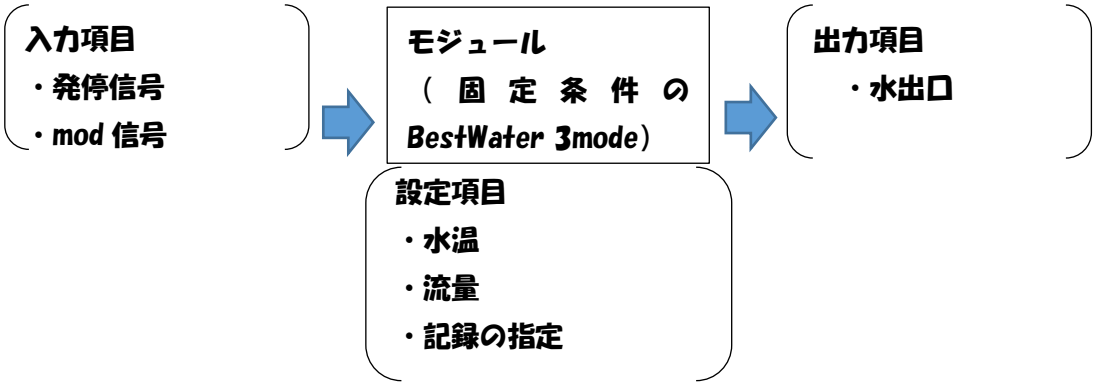


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	1固定水温[冷房時]	double	c_Temp[0]	12	[℃]	―	0		←この水温の BestWater を作成します。[冷房]
2	2固定水温[暖房時]	double	c_Temp[1]	12	[℃]	―	0		←この水温の BestWater を作成します。[暖房]
3	3固定水温[その他]	double	c_Temp[2]	12	[℃]	―	0		←この水温の BestWater を作成します。[その他]
4	1固定水量[冷房時]	double	c_FlowRate[0]	0	[L/min(w)]	―	0		←この流量の BestWater を作成します。[冷房]
5	2固定水量[暖房時]	double	c_FlowRate[1]	0	[L/min(w)]	―	0		←この流量の BestWater を作成します。[暖房]
6	3固定水量[その他]	double	c_FlowRate[2]	0	[L/min(w)]	―	0		←この流量の BestWater を作成します。[その他]
13	記録を有効とする	boolean	isRecood	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	On/off 信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	Onoff 信号の入口
3	運転モード	L1_modIn	modIn	-	制御	制御モード	入口	制御モードの入口
4	水出口	L0_watOut	watOut	-	状態	水	出口	水の出口

（6）記録項目

No.	項目	出力内容	単位	分類
1	固定 waterMessage#-#-	メッセージ	―	メッセージ
2	固定 water 温度#℃#温度	L0_watOut の温度	℃	出口
3	固定 water 質量流量#g/s#質量流量	L0_watOut の質量流量	g/s	出口

（7）計算フロー・計算内容

・L0_watrOut の水

入口接続ノード L1_modIn からの制御モード信号から冷暖房中間期を判定し、それぞれの期間に応じた入力値を BestWater 媒体 watOut に設定し接続ノード L0_watOut から渡す。

L1_modIn の制御信号が冷房の時は以下のようになる。

watOut の温度[℃]

＝　1 固定水温[冷房時] [℃]

流量については、

L1_swcIn の発停信号が運転の時は、

watOut の流量[g/s]

＝　1 固定水量[冷房時] [g/s]

L1_swcIn の発停信号が停止の時は、

watOut の流量[g/s]

＝　0 [g/s]

とする。

（8）データ範囲と範囲外の取扱い

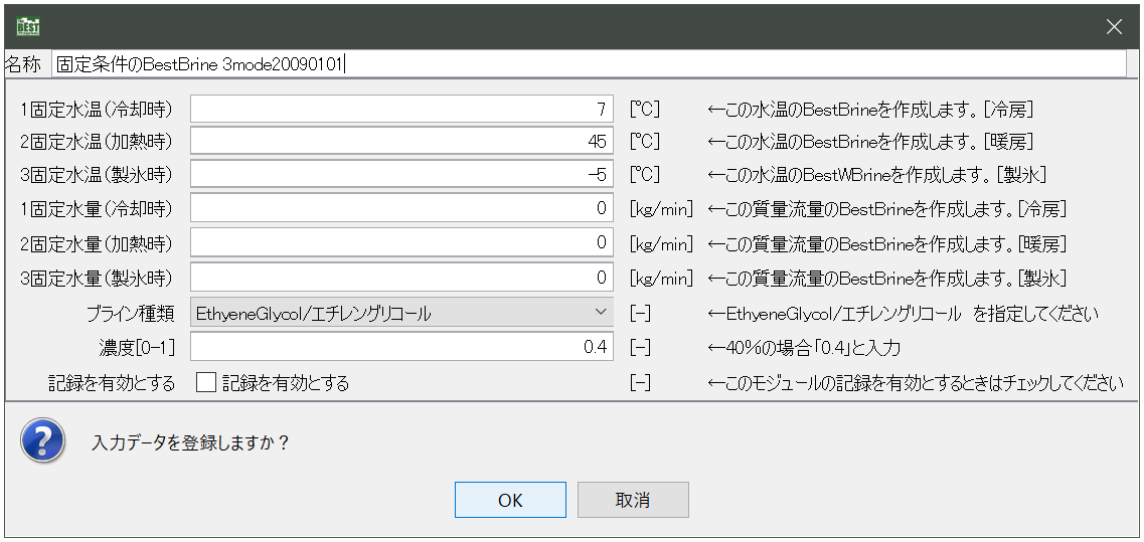
特になし。

「固定条件の BestBrine 3mode」（場所：空調・換気設備／媒体 空気 水／）

モジュール名	固定条件の BestBrine 3mode
クラス	ConstantBrine3modeModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、空気 BestBrine 媒体の境界条件を与えるもので、冷房、暖房、中間期の別に設定できるモジュールである。

L1_modIn の mod 信号により、冷房、暖房、中間期を判断し、それぞれの期間に設定された条件の BestBrine を作成し、L0_briOut 接続ノードから出力する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	1固定水温[冷房時]	double	c_Temp[0]	7	[℃]	ー	0		←この水温の BestBrine を作成します。[冷房]
2	2固定水温[暖房時]	double	c_Temp[1]	45	[℃]	ー	0		←この水温の BestBrine を作成します。[暖房]
3	3固定水温[その他]	double	c_Temp[2]	-5	[℃]	ー	0		←この水温の BestBrine を作成します。[その他]
4	1固定水量[冷房時]	double	c_FlowRate[0]	0	[L/min(w)]	ー	0		←この流量の BestBrine を作成します。[冷房]
5	2固定水量[暖房時]	double	c_FlowRate[1]	0	[L/min(w)]	ー	0		←この流量の BestBrine を作成します。[暖房]
6	3固定水量[その他]	double	c_FlowRate[2]	0	[L/min(w)]	ー	0		←この流量の BestBrine を作成します。[その他]
7	ブライン種類	String	type	EthyeneGlycol/エチレングリコール	[-]				選択リスト　CalciumChloride/塩化カルシウム EthyeneGlycol/エチレングリコール PropyleneGlycol/プロピレングリコール EthyleneChloride/エチレンクロライド
8	濃度[0-1]	Double	concentration	0.4	[-]				
9	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	On/off 信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	Onoff 信号の入口
3	運転モード	L1_modIn	modIn	-	制御	制御モード	入口	制御モードの入口
4	水出口	L0_watOut	watOut	-	状態	水	出口	水の出口

（6）記録項目

No.	項目	出力内容	単位	分類
1	固定 brineMessage#-#-	メッセージ	ー	メッセージ
2	固定 brine 温度#℃#温度	L0_briOut の温度	℃	出口
3	固定 brine 質量流量#g/s#質量流量	L0_briOut の質量流量	g/s	出口

（7）計算フロー・計算内容

・L0_briOut のブライン

入口接続ノード L1_modIn からの制御モード信号から冷暖房中間期を判定し、それぞれの期間に応じた入力値を BestBrine 媒体 briOut に設定し接続ノード L0_briOut から渡す。

L1_modIn の制御信号が冷房の時は以下のようになる。

briOut の温度[℃]

＝　1 固定水温[冷房時] [℃]

流量については、

L1_swcIn の発停信号が運転の時は、

briOut の流量[g/s]

＝　1 固定水量[冷房時] [g/s]

L1_swcIn の発停信号が停止の時は、

briOut の流量[g/s]

＝　0 [g/s]

とする。

（8）データ範囲と範囲外の取扱い

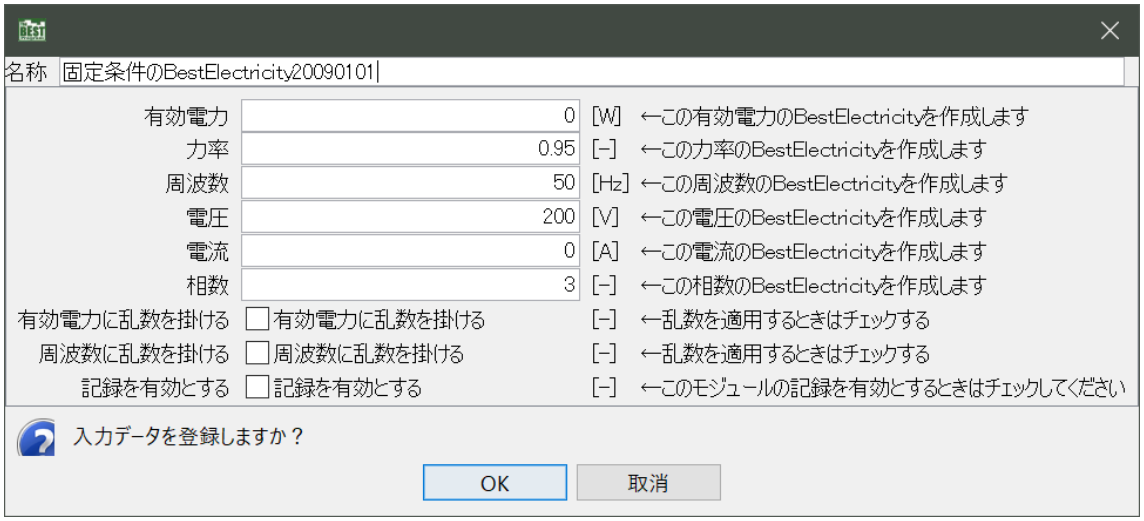
特になし。

「固定条件の BestElectricity」（場所：空調・換気設備／媒体 空気 水／）

モジュール名	固定条件の BestElectricity
クラス	ConstantElectricityInModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、電力 BestElectricity 媒体の境界条件を与えるモジュールである。

L1_swcln の onoff 信号により、設定された条件の BestElectricity を作成し、L0_eleln 接続ノードから出力する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

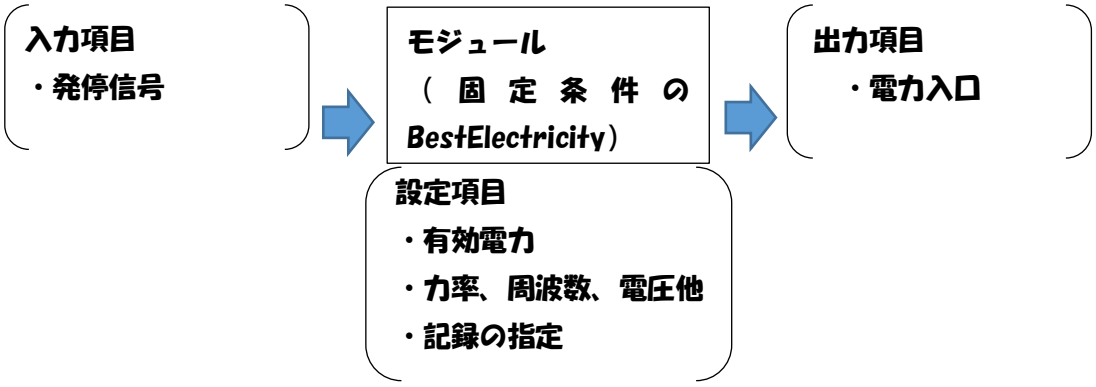


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	有効電力	double	c_activePower	0	[W]	ー	0		←この有効電力の BestElectricity を作成します
2	力率	double	c_powerFactor	0.95	[-]	ー	0		←この力率の BestElectricity を作成します
3	周波数	double	c_frequency	50	[Hz]	ー	0		←この周波数の BestElectricity を作成します
4	電圧	double	c_voltage	020	[A]	ー	0		←この電圧の BestElectricity を作成します
5	電流	double	c_current	0	[-]	ー	0		←この電流の BestElectricity を作成します
6	相数	double	c_phase	3	[-]	ー	0		←この相数の BestElectricity を作成します
7	有効電力に乱数を掛ける	boolean	r_activePower	FALSE	[-]	ー	ー		←乱数を適用するときはチェックする
8	周波数に乱数を掛ける	boolean	r_frequency	FALSE	[-]	ー	ー		←乱数を適用するときはチェックする
9	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	On/off 信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	Onoff 信号の入口
3	電力入口	L0_eleIn	eleIn	-	状態	電力	入口	電力の入口

（6）記録項目

No.	項目	出力内容	単位	分類
1	固定 eleMessage#-#-	メッセージ	ー	メッセージ
2	固定 ele 有効電力#W#-	L0_eleIn の有効電力	W	入口
3	固定 ele 無効電力#Var#-	L0_eleIn の無効電力	Var	入口
4	固定 ele 周波数#Hz#周波数	L0_eleIn の周波数	Hz	入口

（７）計算フロー・計算内容

・L0_eleIn の電力

接続ノード L1_swcIn の onoff 信号により、設定された条件の BestElectricity 媒体 eleIn を作成し、L0_eleIn 接続ノードから出力する。

L1_swcIn の発停信号が運転の時は、

eleIn の有効電力[W]　＝　有効電力［W］

＊有効電力に乱数を掛ける　が有効の時は、Java 関数の乱数を掛けた値とする。

eleIn の無効電力[Var]　＝　有効電力×Math.pow(1/力率/ 力率 - 1, 0.5)

eleIn の電流[A]　＝　電流［A］

eleIn の周波数[Hz]　＝　周波数［Hz］

＊周波数に乱数を掛ける　が有効の時は、Java 関数の乱数を掛けた値とする。

eleIn の電圧[V]　＝　電圧［V］

L1_swcIn の発停信号が停止の時は、

eleIn の有効電力[W]　＝　0［W］

eleIn の無効電力[Var]　＝　0［var］

eleIn の電流[A]　＝　0［A］

eleIn の周波数[Hz]　＝　0 [Hz]

eleIn の電圧[V]　＝　0 [V]

とする。

（８）データ範囲と範囲外の取扱い

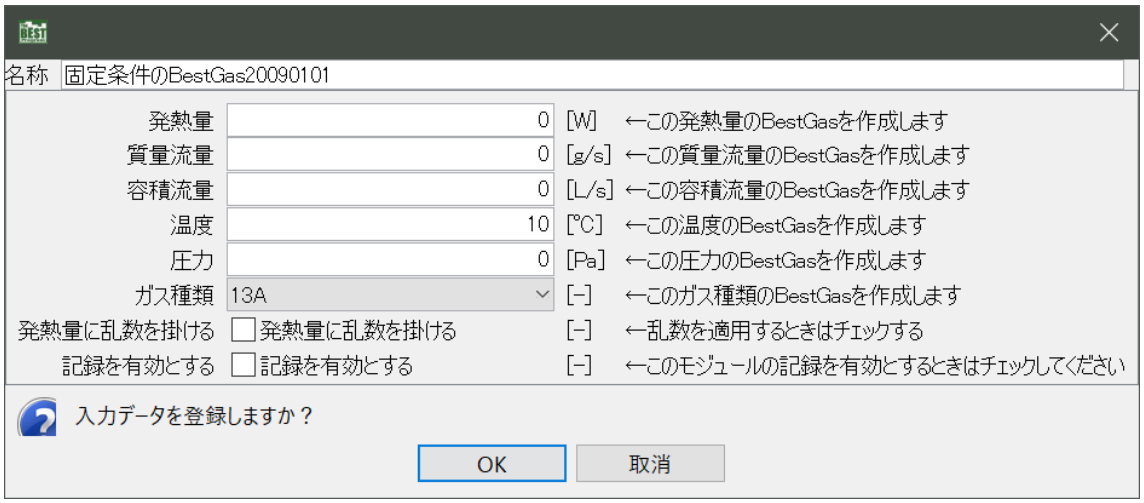
特になし。

「固定条件の BestGas」（場所：空調・換気設備／媒体 空気 水／）

モジュール名	固定条件の BestGas
クラス	ConstantGasInModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、ガス BestGas 媒体の境界条件を与えるモジュールである。

L1_swcln の onoff 信号により、設定された条件の BestGas 媒体 gasIn を作成し、L0_gasIn 接続ノードから出力する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

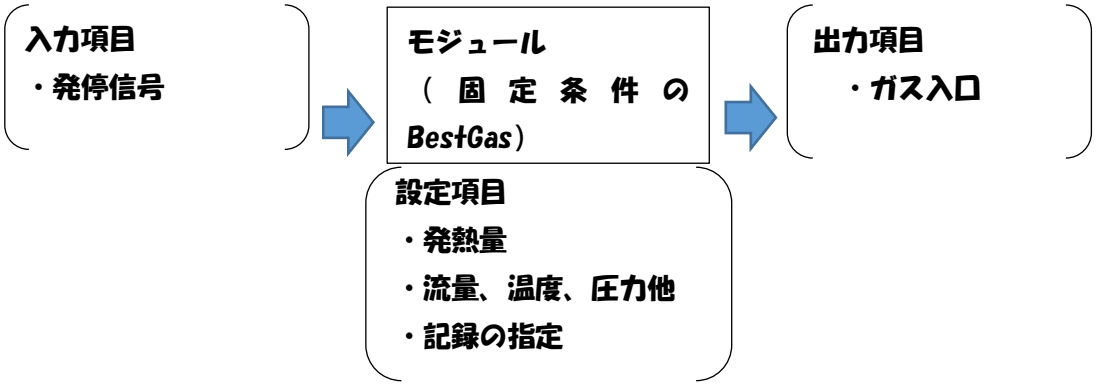


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	発熱量	Double	c_watt	0	[W]	ー	0		←この発熱量の BestGas を作成します
2	質量流量	Double	c_massFlowRate	0	[g/s]	ー	0		←この質量流量の BestGas を作成します
3	容積流量	double	c_volumeFlowRate	0	[L/s]	ー	0		←この容積流量の BestGas を作成します
4	温度	Double	c_temperature	10	[℃]	ー	0		←この温度の BestGas を作成します
5	圧力	double	c_pressure	0	[Pa]	ー	0		←この圧力の BestGas を作成します
6	ガス種類	String	c_gasType	13A	[-]	ー	0		←このガス種類の BestGas を作成します
7	発熱量に乱数を掛ける	boolean	r_wattRandom	FALSE	[-]	ー	ー		←乱数を適用するときはチェックする
8	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	On/off 信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	Onoff 信号の入口
3	ガス入口	L0_gasIn	gasIn	-	状態	ガス	入口	ガスの入口

（6）記録項目

No.	項目	出力内容	単位	分類
1	固定 gasMessage#-#-	メッセージ	ー	メッセージ
2	固定 gas 発熱量#W#熱量	L0_gasIn の発熱量	W	入口
3	固定 gas 質量流量#g/s#質量流量	L0_gasIn の質量流量	g/s	入口
4	固定 gas 容積流量#L/s#容積流量	L0_gawIn の容積流量	L/s	入口
5	固定 gas 温度#W#温度	L0_gasIn の温度	℃	入口
6	固定 gas 圧力#Pa#圧力	L0_gasIn の圧力	Pa	入口
7	固定 gas 種類#-#-	L0_gasIn の種類	-	入口

（7）計算フロー・計算内容

・L0_gasIn のガス

接続ノード L1_swcIn の onoff 信号により、設定された条件の BestGas 媒体 gasIn を作成し、L0_gasIn 接続ノードから出力する。

L1_swcIn の発停信号が運転の時は、

gasIn の発熱量[W]　＝　発熱量［W］

＊発熱量に乱数を掛けるが有効の時は、Java 関数の乱数をかけた値とする。

gasIn の質量流量[g/s]　＝　質量流量［g/s］

gasIn の容積流量[L/s]　＝　容積流量［L/s］

gasIn の温度[℃]　＝　温度［℃］

gasIn の圧力[Pa]　＝　圧力［Pa］

gasIn のガスタイプ[-]　＝　電圧［V］

L1_swcIn の発停信号が停止の時は、

gasIn の発熱量[W]　＝　0［W］

gasIn の質量流量[g/s]　＝　0［g/s］

gasIn の容積流量[L/s]　＝　0［L/s］

gasIn の温度[℃]　＝　温度［℃］

gasIn の圧力[Pa]　＝　圧力［Pa］

gasIn のガスタイプ[-]　＝　電圧［V］

とする。

（8）データ範囲と範囲外の取扱い

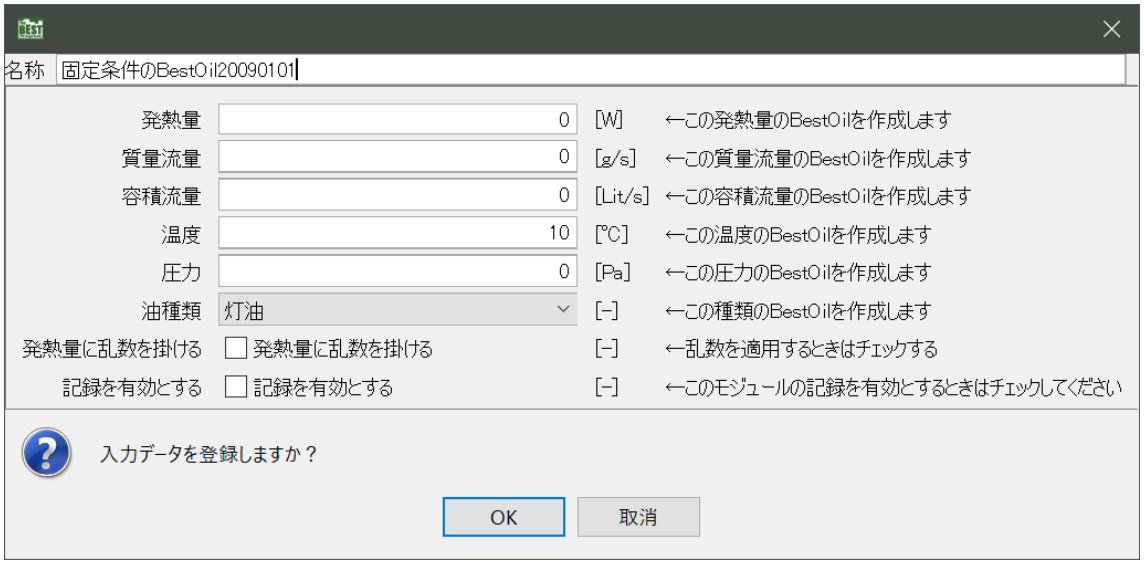
特になし。

「固定条件の BestOil」（場所：空調・換気設備／媒体 空気 水／）

モジュール名	固定条件の BestOil
クラス	ConstantOilInModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、ガス BestOil 媒体の境界条件を与えるモジュールである。

L1_swcln の onoff 信号により、設定された条件の BestOil 媒体 oilln を作成し、L0_oilln 接続ノードから出力する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

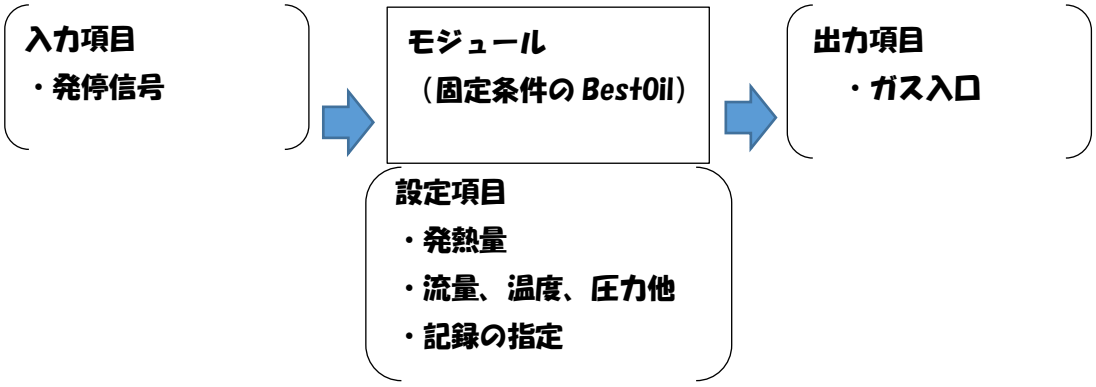


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	ー	[-]	ー	ー		
1	発熱量	Double	c_watt	0	[W]	ー	0		←この発熱量の BestOil を作成します
2	質量流量	Double	c_massFlowRate	0	[g/s]	ー	0		←この質量流量の BestOil を作成します
3	容積流量	double	c_volumeFlowRate	0	[L/s]	ー	0		←この容積流量の BestOil を作成します
4	温度	Double	c_temperature	10	[℃]	ー	0		←この温度の BestOil を作成します
5	圧力	double	c_pressure	0	[Pa]	ー	0		←この圧力の BestOil を作成します
6	油種類	String	c_gasType	灯油	[-]	ー	ー		←この油の種類の BestOil を作成します
7	発熱量に乱数を掛ける	boolean	r_wattRandom	FALSE	[-]	ー	ー		←乱数を適用するときはチェックする
8	記録を有効とする	boolean	isRecood	FALSE	[-]	ー	ー		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	On/off 信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	OnOff 信号の入口
3	ガス入口	L0_oilIn	oilIn	-	状態	油	入口	油の入口

（6）記録項目

No.	項目	出力内容	単位	分類
1	固定 oilMessage#-#-	メッセージ	ー	メッセージ
2	固定 oil 発熱量#W#熱量	L0_oilIn の発熱量	W	入口
3	固定 oil 質量流量#g/s#質量流量	L0_oilIn の質量流量	g/s	入口
4	固定 oil 容積流量#L/s#容積流量	L0_oilIn の容積流量	L/s	入口
5	固定 oil 温度#W#温度	L0_oilIn の温度	℃	入口
6	固定 oil 圧力#Pa#圧力	L0_oilIn の圧力	Pa	入口
7	固定 oil 種類#-#-	L0_oilIn の種類	-	入口

（7）計算フロー・計算内容

・L0_oiilln のガス

接続ノード L1_swcln の onoff 信号により、設定された条件の BestOil 媒体 oilln を作成し、L0_oilln 接続ノードから出力（消費量、状態値等）する。

L1_swcln の発停信号が運転の時は、

oilln の発熱量[W]　＝　発熱量［W］

＊発熱量に乱数を掛けるが有効の時は、Java 関数の乱数をかけた値とする。

oilln の質量流量[g/s]　＝　質量流量［g/s］

oilln 容積流量[L/s]　＝　容積流量［L/s］

oilln 温度[℃]　＝　温度［℃］

oilln の圧力[Pa]　＝　圧力［Pa］

oilln の油タイプ[-]　＝　（タイプ）［-］

L1_swcln の発停信号が停止の時は、

oilln の発熱量[W]　＝　0［W］

oilln の質量流量[g/s]　＝　0［g/s］

oilln の容積流量[L/s]　＝　0［L/s］

oilln の温度[℃]　＝　温度［℃］

oilln の圧力[Pa]　＝　圧力［Pa］

oilln の油タイプ[-]　＝　（タイプ）［-］

とする。

（8）データ範囲と範囲外の取扱い

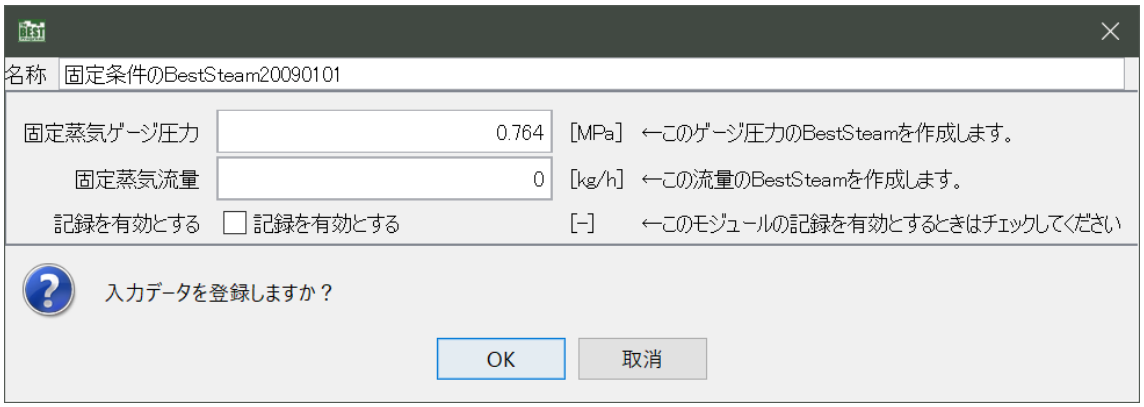
特になし。

「固定条件の BestSteam」（場所：空調・換気設備／媒体 空気 水／）

モジュール名	固定条件の BestSteam
クラス	ConstantSteamModule20090101

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、ガス BestSteam 媒体の境界条件を与えるモジュールである。

L1_swcln の onoff 信号により、設定された条件の BestSteam 媒体 steOut を作成し、L0_steOut 接続ノードから出力する。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

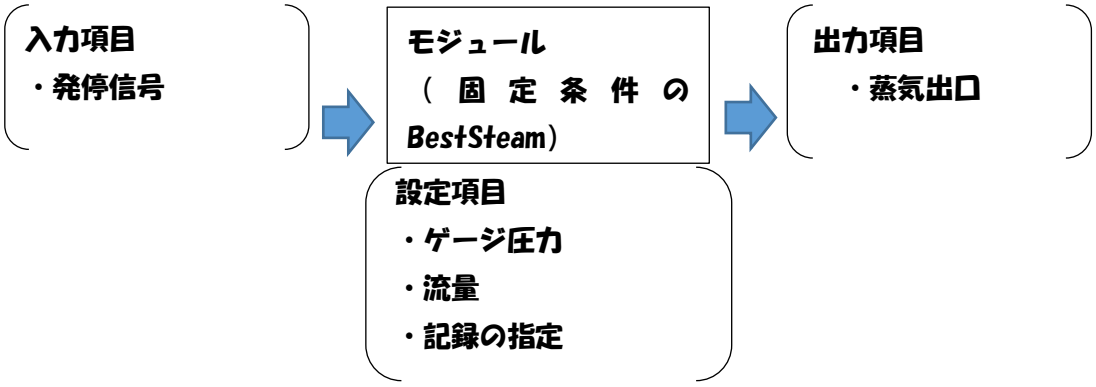


図1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図１における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	発熱量	Double	c_watt	0	[W]	―	0		←この発熱量の BestSteam を作成します
2	質量流量	Double	c_FlowRate	0	[g/s]	―	0		←この質量流量の BestSteam を作成します
5	圧力	double	c_pressure	0	[Pa]	―	0		←この圧力の BestSteam を作成します
8	記録を有効とする	boolean	isRecood	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	On/off 信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	OnOff 信号の入口
3	蒸気出口	L0_steOut	steOut	-	状態	蒸気	出口	蒸気の出口

（6）記録項目

No.	項目	出力内容	単位	分類
1	固定 SteamMessage#-#-	メッセージ	―	メッセージ
2	固定 Steam 温度#温度	L0_steOut の温度	℃	出口
3	固定 Steam 質量流量#g/s#質量流量	L0_steOut の質量流量	g/s	出口

（7）計算フロー・計算内容

・L0_oiilln のガス

接続ノード L1_swcln の onoff 信号により、設定された条件の BestSteam 媒体 steOut を作成し、L0_steOut 接続ノードから出力（流量等）する。

L1_swcln の発停信号が運転の時は、

steOut の質量流量[g/s]　＝　質量流量［g/s]

steOut の圧力[Pa]　＝　圧力［Pa]

L1_swcln の発停信号が停止の時は、

steOut 質量流量[g/s]　＝　0［g/s]

steOut の圧力[Pa]　＝　圧力［Pa]

とする。

（8）データ範囲と範囲外の取扱い

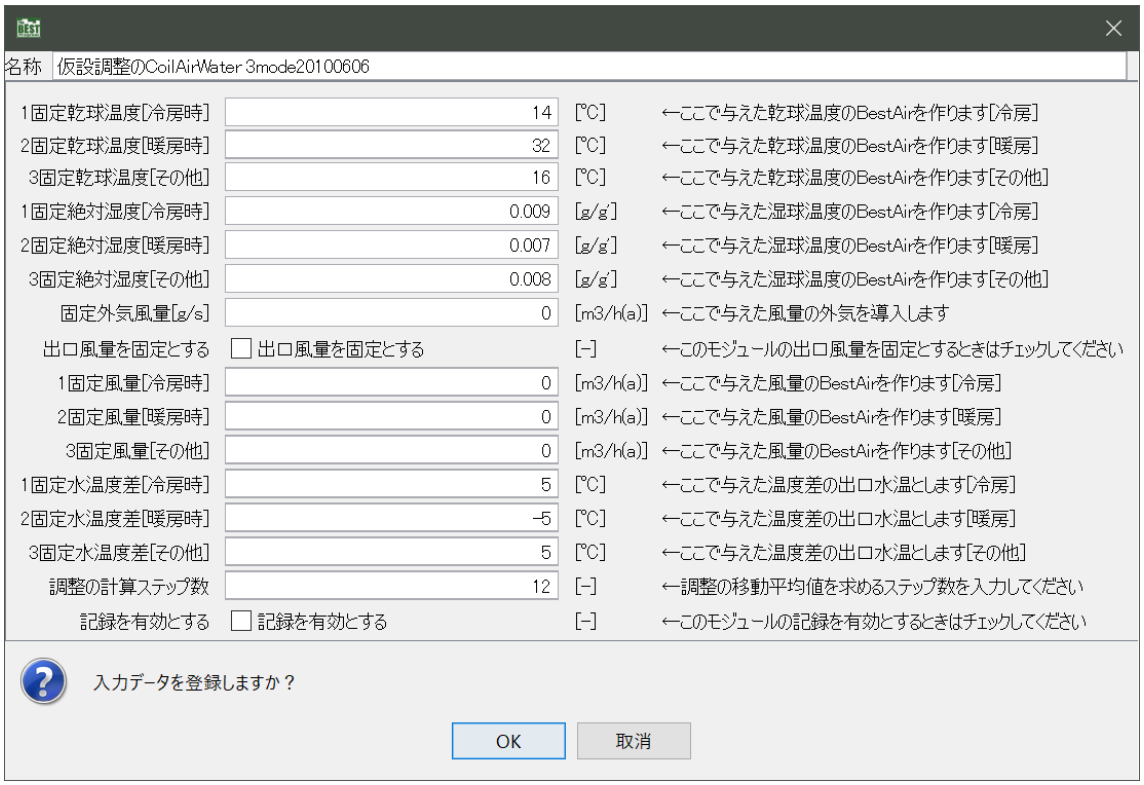
特になし。

「仮設調整の CoilAirWater 3mode」（場所：空調・換気設備／媒体 空気 水／）

モジュール名	仮設調整の CoilAirWater 3mode
クラス	AdjustCoilAirWater3modeModule20100606

（1）入力画面

・スペック



（2）モジュールの概要

本モジュールは、冷房、暖房、中間期の別に設定した一定の温湿度の BestAir を発生し送るとともに、仮設コイル（空調機）の入口側の空気流量と出入口空気のエンタルピ差から、送風状態にするために必要な熱量と水量を計算して処理熱量等を記録する。仮設コイルの処理熱量を簡易に計算することができるモジュールである。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

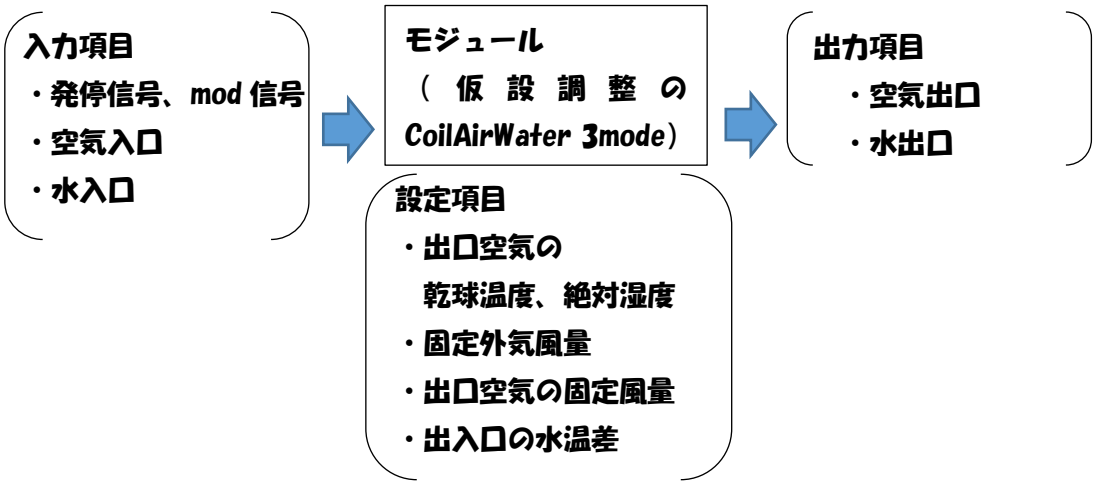
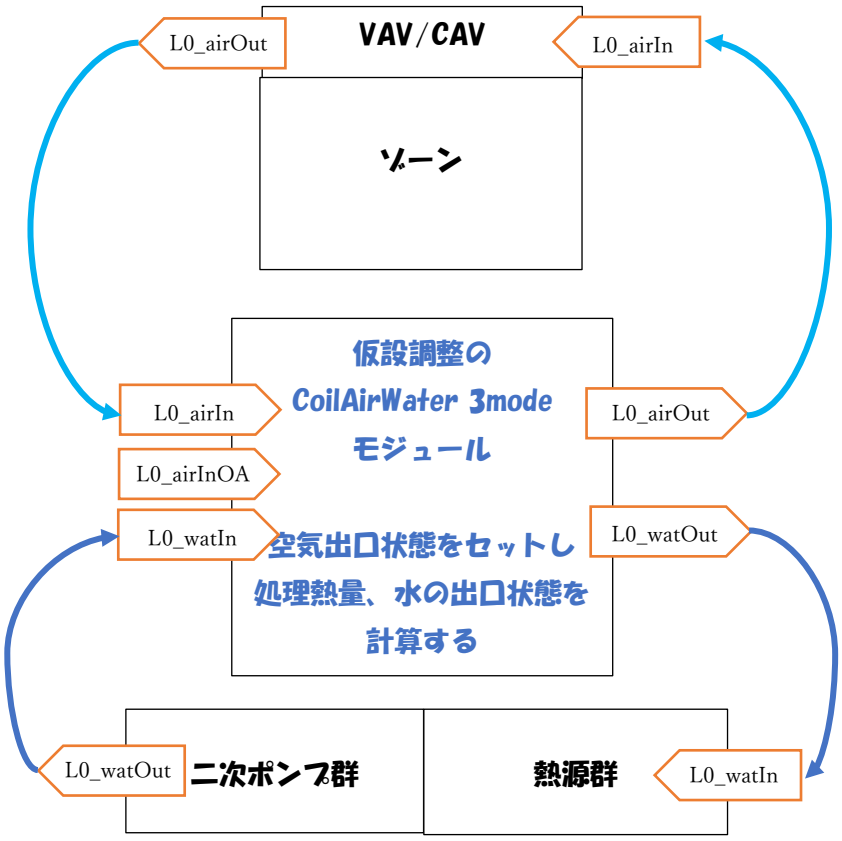


図１　モジュールの設定項目・入力項目・出力項目



（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name	―	[-]	―	―		
1	1 固定乾球温度[冷房時]	double	c_DB_airOut[0]	14	[℃]	―	―		←ここで与えた乾球温度の BestAir を作ります[冷房]
2	2 固定乾球温度[暖房時]	double	c_DB_airOut[1]	32	[℃]	―	―		←ここで与えた乾球温度の BestAir を作ります[暖房]
3	3 固定乾球温度[その他]	double	c_DB_airOut[2]	16	[℃]	―	―		←ここで与えた乾球温度の BestAir を作ります[その他]
4	1 固定絶対湿度[冷房時]	double	c_X_airOut[0]	0.009	[g/g']	―	0		←ここで与えた絶対湿度の BestAir を作ります[冷房]
5	2 固定絶対湿度[暖房時]	double	c_X_airOut[1]	0.007	[g/g']	―	0		←ここで与えた絶対湿度の BestAir を作ります[暖房]
6	3 固定絶対湿度[その他]	double	c_X_airOut[2]	0.008	[g/g']	―	0		←ここで与えた絶対湿度の BestAir を作ります[その他]
7	固定外気風量[g/s]	double	c_M_aiInOA	0	[m3/h(a)]	―	0		←ここで与えた風量の外気を導入します
8	出口風量を固定とする	boolean	isConstant	False	[-]	―	―		←出口風量を固定とするときはチェックしてください
9	1 固定風量[冷房時]	Double	c_M_airOut[0]	0	[m3/h(a)]	―	0		←ここで与えた風量の BestAir を作ります[冷房]
10	2 固定風量[暖房時]	Double	c_M_airOut[1]	0	[m3/h(a)]	―	0		←ここで与えた風量の BestAir を作ります[暖房]
11	3 固定風量[その他]	Double	c_M_airOut[2]	0	[m3/h(a)]	―	0		←ここで与えた風量の BestAir を作ります[その他]
12	1 固定水温度差[冷房時]	Double	c_DT_watInOut[0]	5	[℃]	―	0		←ここで与えた温度差の出口水温とします[冷房]
13	2 固定水温度差[暖房時]	Double	c_DT_watInOut[1]	-5	[℃]	―	0		←ここで与えた温度差の出口水温とします[暖房]
14	3 固定水温度差[その他]	Double	c_DT_watInOut[2]	5	[℃]	―	0		←ここで与えた温度差の出口水温とします[その他]
15	調整の計算ステップ数	int	numAdjustSteps	12	[-]	―	1		←調整の移動平均値を求めるステップ数を入力してください
16	記録を有効とする	boolean	isRecood	FALSE	[-]	―	―		←このモジュールの記録を有効とするときはチェックしてください

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	On/off 信号	L1_swcIn	swcIn	-	制御	OnOff 信号	入口	OnOff 信号の入口（現在未使用）
3	On/off 信号-外気	L1_swcInOA	swcInOA	-	制御	OnOff 信号	入口	外気導入の OnOff 信号の入口
4	On/off 信号-SAFan	L1_swcInFANSA	swcInFANSA	-	制御	OnOff 信号	入口	SAFan の OnOff 信号の入口
5	On/off 信号-RAFan	L1_swcInFANRA	swcInFANRA	-	制御	OnOff 信号	入口	RAFan の OnOff 信号の入口（現在未使用）
6	On/off 信号-EAFan	L1_swcInFANEA	swcInFANEA	-	制御	OnOff 信号	入口	EAFan の OnOff 信号の入口（現在未使用）
7	On/off 信号-OAFan	L1_swcInFANOA	swcInFANOA	-	制御	OnOff 信号	入口	OAFan の OnOff 信号の入口（現在未使用）
8	On/off 信号-Coil	L1_swcInCOIL	swcInCOIL	-	制御	OnOff 信号	入口	Coil の OnOff 信号の入口（現在未使用）
9	On/off 信号-SPRAY	L1_swcInSPRAY	swcInSPRAY	-	制御	OnOff 信号	入口	SPRAY の OnOff 信号の入口（現在未使用）
10	On/off 信号-THE	L1_swcInTHE	swcInTHE	-	制御	OnOff 信号	入口	THE の OnOff 信号の入口
11	運転モード	L1_modIn	modIn	-	制御	制御モード	入口	制御モードの入口
12	空気入口	L0_airIn	airIn	-	状態	空気	入口	RA 入口接続用
13	外気入口	L0_airInOA	airInOA	-	状態	空気	入口	OA 入口接続用
14	空気出口	L0_airOut	airOut	-	状態	空気	出口	SA 出口接続用
15	水入口	L0_watIn	watIn	-	状態	水	入口	冷温水入口接続用
16	水出口	L0_watOut	watOut	-	状態	水	出口	冷温水出口接続用

（6）記録項目

No.	項目	出力内容	単位	分類
1	AdjustCoilAir_Message#-#-	メッセージ	－	メッセージ
2	AdjustCoilAir_出口乾球温度#℃#温度	LO_airOut の乾球温度	℃	出口
3	AdjustCoilAir_出口絶対湿度#g/g'#絶対湿度	LO_airOut の絶対湿度	g/g ‘	出口
4	AdjustCoilAir_出口質量風量#g/s#質量流量	LO_airOut の質量風量	g/s	出口
5	AdjustCoilAir_入口乾球温度#℃#温度	LO_airIn の乾球温度	℃	入口
6	AdjustCoilAir_入口絶対湿度#g/g'#絶対湿度	LO_airIn の絶対湿度	g/g ‘	入口
7	AdjustCoilAir_入口質量風量#g/s#質量流量	LO_airIn の質量風量	g/s	入口
8	AdjustCoilWater_出口温度#℃#温度	LO_watOut の温度	℃	出口
9	AdjustCoilWater_出口質量流量#g/s#質量流量	LO_watOut の質量流量	g/s	出口
10	AdjustCoilWater_入口温度#℃#温度	LO_watIn の温度	℃	入口
11	AdjustCoilWater_入口質量流量#g/s#質量流量	LO_watIn の質量流量	g/s	入口
12	AdjustCoilWater_mode1 必要熱量#W#熱量	出口空気状態にするために必要な熱量[冷房]	W	出口
13	AdjustCoilWater_mode2 必要熱量#W#熱量	出口空気状態にするために必要な熱量[暖房]	W	出口
14	AdjustCoilWater_mode3 必要熱量#W#熱量	出口空気状態にするために必要な熱量[その他]	W	出口
15	AdjustCoilAir_調整最大質量風量#g/s#質量流量	調整結果の最大風量（移動平均値）	g/s	調整
16	AdjustCoilWater_調整最大質量流量#g/s#質量流量	調整結果の最大流量（移動平均値）	g/s	調整
17	AdjustCoilWater_調整最大必要熱量#W#熱量	調整結果の最大必要熱量（移動平均値）	W	調整
18	AdjustCoilWater_調整最小必要熱量#W#熱量	調整結果の最小必要熱量（移動平均値）	W	調整
19	AdjustCoilAir_調整ステップ平均質量風量#g/s#質量流量	調整ステップの平均風量	g/s	調整
20	AdjustCoilWater_調整ステップ平均質量流量#g/s#質量流量	調整ステップの平均流量	g/s	調整
21	AdjustCoilWater_調整ステップ平均必要熱量#W#熱量	調整ステップの平均必要熱量	W	調整

（7）計算フロー・計算内容

・L1_swcInFANSA の OnOff 信号による動作

L1_swcInFANSA の信号により、運転か停止かを判断する。

停止の場合は、出口空気の流量を 0 とする。

運転の場合、次の動作となる。

出口風量を固定とする＝TRUE の場合：

出口空気の流量を入力された固定風量とする。

出口風量を固定とする＝FALSE の場合：

出口空気の流量を airIn の風量とする。

・L1_swcInOA の OnOff 信号による動作

L1_swcInOA の信号により、外気導入を判断する。

停止の場合は、airInOA の流量を 0 とする。

運転の場合は、airInOA の流量を固定外気風量の入力値とする。

・L0_airOut の空気

入口接続ノード L1_modIn からの制御モード信号から冷暖房中間期を判定し、それぞれの期間に応じた入力値を BestAir 媒体 airOut に設定し接続ノード L0_airOut から渡す。

・熱量の計算

入口空気 airIn の風量が 0 または出口空気 airOut の風量が 0 の場合：

必要熱量＝ 0

入口外気 airInOA の風量が出口空気 airOut の風量より大きい場合：

必要熱量＝（ 出口空気比エンタルピ ー 外気比エンタルピ ）×出口空気質量流量

入口外気 airInOA の風量が出口空気 airOut の風量以下の場合：

必要熱量＝ 出口空気比エンタルピ × 出口空気質量流量

ー 外気比エンタルピ × 外気質量流量

ー 入口空気比エンタルピー ×（ 入口空気質量流量 － 外気質量流量 ）

上記計算結果で必要熱量は冷却が負の値、加熱が正の値となる。

冷房時に必要熱量＞ 0、暖房時に必要熱量＜ 0 となる場合は必要熱量を 0 とする

・出口水 watOut の状態値の計算

流量＝必要熱量÷固定水温度差÷水の比熱

水温＝入口水温＋固定水温度差

（8）データ範囲と範囲外の取扱い

特になし。

「媒体テストデータセット 2011」（場所：設備 2015／媒体境界条件 2015／）

モジュール名	媒体テストデータセット 2011
クラス	MediumTestSetsModule20110725

（1）入力画面

・スペック

（2）モジュールの概要

媒体テストデータセットモジュールは、熱源などの機器モジュールや複数のモジュールで構成した部分システムの動作テストを行うときに、その境界条件を与えるモジュールとして利用できるものである。

境界条件はこのモジュールの BEST 媒体の出口接続ノードを接続することで利用できる。

動作をテストする時の境界条件は、そのモジュールの入口側の接続ノードの情報である。

例えば HP チラーのモジュールでは、冷房時の機器特性は冷水の watIn（入口温度、流量）、airIn（外気乾球温度）、modIn（冷暖モード）、swcIn（発停）、valInCapCtrl（容量制御）、出口目標温度（valInSP_watOut）が境界条件として考えられる。

媒体テストデータセットモジュールは、これらの動作テストに必要な境界条件をまとめて設定するものであり、入力したパラメータのすべての組合わせの境界条件を作成し、それらの媒体出口接続ノードから発信することができる。

設定できる媒体は、冷温水（watOutCH）、温水（watOutH）、冷却水（watOutCD）、熱源水（watOutHS）、排熱水（watOutHE）、外気（airOutOA）、室空気（airOutRM）、val × 3（valOutA、valOutB、valOutC）、ブライン冷水（briOutC）、ブライン冷却水（briOutCD）、ブライン熱源水（briOutHS）、蒸気発生 × 3（steOutA、steOutB、steOutC）、蒸気消費 × 4（steInA、steInB、steInC、steInD）、運転モード × 3（合成モードを modOut）である。

設定できる媒体の管理変数は、wat と bri が水温・流量、air は乾球温度・相対湿度・流量、val は値、ste はゲージ圧力・蒸気流量・凝縮水流量である。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

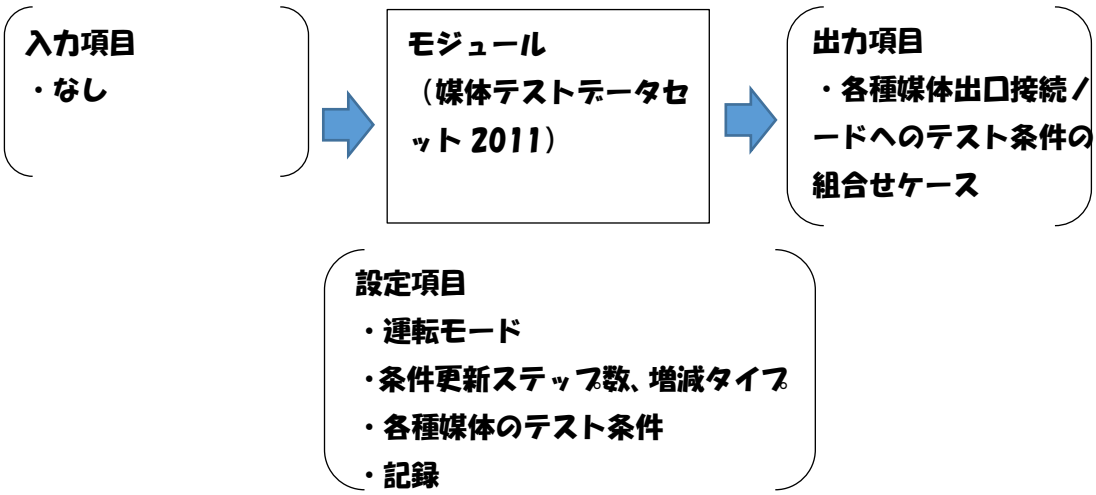


図 1　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図1における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	―	―		
1	運転モード1 運転モード2 運転モード3	備考欄から選択	de_mod1 de_mod2 de_mod3	COOL	[-]	―	―		COOL、HEAT、ICE、RECOVER、C_CHARGE、H_CHARGE、HUMIDIFY、DEHUMIDIFY、DEFROST、VAV、OACUT、FREECOOL、PEAKCUT、NIGHTPURGE、VENTILATE、E_CHARGE、PEAKSHIFT、WH_RECOVERY
2	条件更新ステップ数	String	de_keepSteps	1	[-]	―	1		←このステップ数同じ条件を発生します
3	増減タイプ	1,2,3 から選択	de_ud, typeUp	1	[-]	―	1		←増減タイプ　=1 増加のみ、=2:減少のみ、=3:増加後減少
4	冷温水　水温設定値	String	de_T_watOutCH	0	[℃ ℃ ・・・]	―	―		←watOutCH
5	冷温水　流量設定値	String	de_M_watOutCH	0	[g/s g/s ・・・]	―	―		←watOutCH
6	温水　水温設定値	String	de_T_watOutH	0	[℃ ℃ ・・・]	―	―		←watOutH
7	温水　流量設定値	String	de_M_watOutH	0	[g/s g/s ・・・]	―	―		←watOutH
8	冷却水　水温設定値	String	de_T_watOutCD	0	[℃ ℃ ・・・]	―	―		←watOutCD
9	冷却水　流量設定値	String	de_M_watOutCD	0	[g/s g/s ・・・]	―	―		←watOutCD
10	熱源水　水温設定値	String	de_T_watOutHS	0	[℃ ℃ ・・・]	―	―		←watOutHS
11	熱源水　流量設定値	String	de_M_watOutHS	0	[g/s g/s ・・・]	―	―		←watOutHS
12	排熱水　水温設定値	String	de_T_watOutHE	0	[℃ ℃ ・・・]	―	―		←watOutHE
13	排熱水　流量設定値	String	de_M_watOutHE	0	[g/s g/s ・・・]	―	―		←watOutHE
14	外気　乾球温度設定値	String	de_T_airOutOA	0	[℃ ℃ ・・・]	―	―		←airOutOA
15	外気　相対湿度設定値	String	de_RH_airOutOA	50	[% % ・・・]	―	―		←airOutOA 相対湿度=０%は計算エラーとなることがあります
16	外気　流量設定値	String	de_M_airOutOA	0	[g/s g/s ・・・]	―	―		←airOutOA
17	室空気　乾球温度設定値	String	de_T_airOutRM	0	[℃ ℃ ・・・]	―	―		←airOutRM
18	室空気　相対湿度設定値	String	de_RH_airOutRM	50	[% % ・・・]	―	―		←airOutRM 相対湿度=０%は計算エラーとなることがあります
19	室空気　流量設定値	String	de_M_airOutRM	0	[g/s g/s ・・・]	―	―		←airOutRM

20	valOutA　値	String	de_V_valOutA	0	[- - ・・・]	—	—		←valOutA
21	valOutB　値	String	de_V_valOutB	0	[- - ・・・]	—	—		←valOutB
22	valOutC　値	String	de_V_valOutC	0	[- - ・・・]	—	—		←valOutC
23	ブライン冷水　水温設定値	String	de_T_briOutC	0	[℃ ℃ ・・・]	—	—		←briOutC
24	ブライン冷水　流量設定値	String	de_M_briOutC	0	[g/s g/s ・・・]	—	—		←briOutC
25	ブライン冷却水　水温設定値	String	de_T_briOutCD	0	[℃ ℃ ・・・]	—	—		←briOutCD
26	ブライン冷却水　流量設定値	String	de_M_briOutCD	0	[g/s g/s ・・・]	—	—		←briOutCD
27	ブライン熱源水　水温設定値	String	de_T_briOutHS	0	[℃ ℃ ・・・]	—	—		←briOutHS
28	ブライン熱源水　流量設定値	String	de_M_briOutHS	0	[g/s g/s ・・・]	—	—		←briOutHS
29	蒸気発生Ａ　ゲージ圧力設定値	String	de_Pa_steOutA	784000	[Pa Pa ・・・]	—	—		←steOutA
30	蒸気発生Ａ　蒸気流量設定値	String	de_Ms_steOutA	0	[g/s g/s ・・・]	—	—		←steOutA
31	蒸気発生Ａ　凝縮水流量設定値	String	de_Mc_steOutA	0	[g/s g/s ・・・]	—	—		←steOutA
32	蒸気発生Ｂ　ゲージ圧力設定値	String	de_Pa_steOutB	784000	[Pa Pa ・・・]	—	—		←steOutB
33	蒸気発生Ｂ　蒸気流量設定値	String	de_Ms_steOutB	0	[g/s g/s ・・・]	—	—		←steOutB
34	蒸気発生Ｂ　凝縮水流量設定値	String	de_Mc_steOutB	0	[g/s g/s ・・・]	—	—		←steOutB
35	蒸気発生Ｃ　ゲージ圧力設定値	String	de_Pa_steOutC	784000	[Pa Pa ・・・]	—	—		←steOutC
36	蒸気発生Ｃ　蒸気流量設定値	String	de_Ms_steOutC	0	[g/s g/s ・・・]	—	—		←steOutC
37	蒸気発生Ｃ　凝縮水流量設定値	String	de_Mc_steOutC	0	[g/s g/s ・・・]	—	—		←steOutC
38	蒸気消費Ａ　ゲージ圧力設定値	String	de_Pa_steInA	784000	[Pa Pa ・・・]	—	—		←steInA
39	蒸気消費Ａ　蒸気流量設定値	String	de_Ms_steInA	0	[g/s g/s ・・・]	—	—		←steInA
40	蒸気消費Ａ　凝縮水流量設定値	String	de_Mc_steInA	0	[g/s g/s ・・・]	—	—		←steInA
41	蒸気消費Ｂ　ゲージ圧力設定値	String	de_Pa_steInB	784000	[Pa Pa ・・・]	—	—		←steInB
42	蒸気消費Ｂ　蒸気流量設定値	String	de_Ms_steInB	0	[g/s g/s ・・・]	—	—		←steInB
43	蒸気消費Ｂ　凝縮水流量設定値	String	de_Mc_steInB	0	[g/s g/s ・・・]	—	—		←steInB
44	蒸気消費Ｃ　ゲージ圧力設定値	String	de_Pa_steInC	784000	[Pa Pa ・・・]	—	—		←steInC
45	蒸気消費Ｃ　蒸気流量設定値	String	de_Ms_steInC	0	[g/s g/s ・・・]	—	—		←steInC
46	蒸気消費Ｃ　凝縮水流量設定値	String	de_Mc_steInC	0	[g/s g/s ・・・]	—	—		←steInC
47	蒸気消費Ｄ　ゲージ圧力設定値	String	de_Pa_steInD	784000	[Pa Pa ・・・]	—	—		←steInD
48	蒸気消費Ｄ　蒸気流量設定値	String	de_Ms_steInD	0	[g/s g/s ・・・]	—	—		←steInD
49	蒸気消費Ｄ　凝縮水流量設定値	String	de_Mc_steInD	0	[g/s g/s ・・・]	—	—		←steInD
50	記録を有効とする	boolean	isRecord	false	[-]	—	—		

（5）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	On/Off 信号	L1_swcOut	swcOut		制御	制御	出口	常に on 信号を出力する
3	運転モード	L1_modOut	modOut		制御	制御モード	出口	入力画面の運転モード 1,2,3 を合成したモード信号を出力する
4	外気	LO_airOutOA	airOutOA		状態	空気	出口	外気の乾球温度、相対湿度および流量の入力値から組合せ条件を作成したものを出力する。
5	室空気	LO_airOutRM	airOutRM		状態	空気	出口	室空気の乾球温度、相対湿度および流量の入力値から組合せ条件を作成したものを出力する。
6	冷温水	LO_watOutCH	watOutCH		状態	水	出口	冷温水の水温と流量の入力値から組合せ条件を作成したものを出力する。
7	温水	LO_watOutH	watOutH		状態	水	出口	温水の水温と流量の入力値から組合せ条件を作成したものを出力する。
8	冷却水	LO_watOutCD	watOutCD		状態	水	出口	冷却水の水温と流量の入力値から組合せ条件を作成したものを出力する。
9	熱源水	LO_watOutHS	watOutHS		状態	水	出口	熱源水の水温と流量の入力値から組合せ条件を作成したものを出力する。
10	排熱水	LO_watOutHE	watOutHE	-	状態	水	出口	排熱水の水温と流量の入力値から組合せ条件を作成したものを出力する。
11	値 val	LO_valOutA LO_valOutB LO_valOutC	valOutA valOutB valOutC	-	状態	Double 値	出口	値の入力値から条件を作成したものを出力する。
12	ブライン冷水	LO_briOutC	briOutC	-	状態	ブライン	出口	ブライン冷水の水温と流量の入力値から組合せ条件を作成したものを出力する。
13	ブライン冷却水	LO_briOutCD	briOutCD	-	状態	ブライン	出口	ブライン冷却水の水温と流量の入力値から組合せ条件を作成したものを出力する。
14	ブライン熱源水	LO_briOutHS	briOutHS	-	状態	ブライン	出口	ブライン熱源水の水温と流量の入力値から組合せ条件を作成したものを出力する。
15	蒸気発生	LO_steOutA LO_steOutB LO_steOutC	steOutA steOutB steOutC	-	状態	蒸気	出口	蒸気のゲージ圧力、蒸気流量と凝縮水流量の入力値から組合せ条件を作成したものを出力する。

16	蒸気消費	LO_steInA LO_steInB LO_steInC LO_steInD	steInA steInB steInC steInD	-	状態	蒸気	出口	蒸気のゲージ圧力、蒸気流量と凝縮水流量の入力値から組合せ条件を作成したものを出力する。

（6）記録項目

No.	項目	出力内容	単位	分類
1	MTS_Message#-#-	メッセージ	-	メッセージ
2	MTS_caseNumber#-#-	境界条件の組合せ条件データのケース通し番号	-	出口
3	MTS_01 冷温水温度#℃#温度	LO_watOutCH の温度	℃	出口
4	MTS_02 冷温水流量#g/s#質量流量	LO_watOutCH の流量	g/s	出口
5	MTS_03 温水温度#℃#温度	LO_watOutH の温度	℃	出口
6	MTS_04 温水流量#g/s#質量流量	LO_watOutH の流量	g/s	出口
7	MTS_05 冷却水温度#℃#温度	LO_watOutCD の温度	℃	出口
8	MTS_06 冷却水流量#g/s#質量流量	LO_watOutCD の流量	g/s	出口
9	MTS_07 熱源水温度#℃#温度	LO_watOutHS の温度	℃	出口
10	MTS_08 熱源水流量#g/s#質量流量	LO_watOutHS の流量	g/s	出口
11	MTS_09 廃熱水温度#℃#温度	LO_watOutHE の温度	℃	出口
12	MTS_10 廃熱水流量#g/s#質量流量	LO_watOutHE の流量	g/s	出口
13	MTS_11 外気乾球温度#℃#温度	LO_airOutOA の乾球温度	℃	出口
14	MTS_12 外気絶対湿度#g/g#絶対湿度	LO_airOutOA の絶対湿度	g/g'	出口
15	MTS_12 外気相対湿度#%#相対湿度	LO_airOutOA の相対湿度	%	出口
16	MTS_13 外気流量#g/s#質量流量	LO_airOutOA の風量	g/s	出口
17	MTS_13 外気湿球温度#℃#温度	LO_airOutOA の湿球温度	℃	出口
18	MTS_13 外気露点温度#℃#温度	LO_airOutOA の露点温度	℃	出口
19	MTS_13 外気比エンタルピ#J/g'#比エンタルピ	LO_airOutOA の比エンタルピ	J/g'	出口
20	MTS_14 室空気乾球温度#℃#温度	LO_airOutRM の乾球温度	℃	出口
21	MTS_15 室空気絶対湿度#g/g#絶対湿度	LO_airOutRM の絶対湿度	g/g'	出口
22	MTS_15 室空気相対湿度#%#相対湿度	LO_airOutRM の相対湿度	%	出口
23	MTS_16 室空気流量#g/s#質量流量	LO_airOutRM の風量	g/s	出口
24	MTS_16 室空気湿球温度#℃#温度	LO_airOutRM の湿球温度	℃	出口

25	MTS_16 室空気露点温度#℃#温度	LO_airOutRM の露点温度	℃	出口
26	MTS_16 室空気比エンタルピ#J/g'#比エンタルピ	LO_airOutRM の比エンタルピ	J/g'	出口
27	MTS_17valA#-#-	LO_valOutA の値		出口
28	MTS_18valB#-#-	LO_valOutB の値		出口
29	MTS_19valC#-#-	LO_valOutC の値		出口
30	MTS_20 ブライン冷水温度#℃#温度	LO_briOutC の温度		出口
31	MTS_21 ブライン冷水流量#g/s#質量流量	LO_briOutC の流量		出口
32	MTS_22 ブライン冷却水温度#℃#温度	LO_briOutCD の温度		出口
33	MTS_23 ブライン冷却水流量#g/s#質量流量	LO_briOutCD の流量		出口
34	MTS_24 ブライン熱源水温度#℃#温度	LO_briOutHS の温度		出口
35	MTS_25 ブライン熱源水流量#g/s#質量流量	LO_briOutHS の流量		出口
36	MTS_26 蒸気発生 A 圧力#Pa#圧力	LO_steOutA の圧力		出口
37	MTS_27 蒸気発生 A 蒸気流量#g/s#質量流量	LO_steOutA の蒸気流量		出口
38	MTS_28 蒸気発生 A 凝縮水流量#g/s#質量流量	LO_steOutA の凝縮水流量		出口
39	MTS_29 蒸気発生 B 圧力#Pa#圧力	LO_steOutB の圧力		出口
40	MTS_30 蒸気発生 B 蒸気流量#g/s#質量流量	LO_steOutB の蒸気流量		出口
41	MTS_31 蒸気発生 B 凝縮水流量#g/s#質量流量	LO_steOutB の凝縮水流量		出口
42	MTS_32 蒸気発生 C 圧力#Pa#圧力	LO_steOutC の圧力		出口
43	MTS_33 蒸気発生 C 蒸気流量#g/s#質量流量	LO_steOutC の蒸気流量		出口
44	MTS_34 蒸気発生 C 凝縮水流量#g/s#質量流量	LO_steOutC の凝縮水流量		出口
45	MTS_35 蒸気消費 A 圧力#Pa#圧力	LO_steInA の圧力		出口
46	MTS_36 蒸気消費 A 蒸気流量#g/s#質量流量	LO_steInA の蒸気流量		出口
47	MTS_37 蒸気消費 A 凝縮水流量#g/s#質量流量	LO_steInA の凝縮水流量		出口
48	MTS_38 蒸気消費 B 圧力#Pa#圧力	LO_steInB の圧力		出口
49	MTS_39 蒸気消費 B 蒸気流量#g/s#質量流量	LO_steInB の蒸気流量		出口
50	MTS_40 蒸気消費 B 凝縮水流量#g/s#質量流量	LO_steInB の凝縮水流量		出口
51	MTS_41 蒸気消費 C 圧力#Pa#圧力	LO_steInC の圧力		出口
52	MTS_42 蒸気消費 C 蒸気流量#g/s#質量流量	LO_steInC の蒸気流量		出口
53	MTS_43 蒸気消費 C 凝縮水流量#g/s#質量流量	LO_steInC の凝縮水流量		出口
54	MTS_44 蒸気消費 D 圧力#Pa#圧力	LO_steInD の圧力		出口
55	MTS_45 蒸気消費 D 蒸気流量#g/s#質量流量	LO_steInD の蒸気流量		出口
56	MTS_46 蒸気消費 D 凝縮水流量#g/s#質量流量	LO_steInD の凝縮水流量		出口

（7）計算フロー・計算内容

・運転モード（L1_modOut)、発停信号(L1_swcOut)

運転モード mod は種類が多く下記の通りとなる。

Airmod クラスで定義されたモード

COOL = 1;//冷房運転　熱源、空調機
HEAT = 2;//暖房運転　熱源、空調機
ICE = 4;//製氷運転　熱源
RECOVER = 8;//熱回収　熱源、熱交換器、空調機、全熱交換器
C_CHARGE = 16;//蓄熱運転　熱源　冷房
H_CHARGE = 32;//蓄熱運転　熱源　暖房
HUMIDIFY = 64;//加湿　空調機、加湿器
DEHUMIDIFY = 128;//除湿　空調機、加湿器
DEFROST = 256;//除霜　熱源
VAV = 512;//VAV　空調機、VAV ユニット
OACUT = 1024;//外気カット
FREECOOL = 2048;//外気冷房
PEAKCUT = 4096;//ピークカット
NIGHTPURGE = 8192;//ナイトパージ
VENTILATE = 16384;//換気運転　空調機
E_CHARGE = 32768;//蓄電
PEAKSHIFT = 65536;//ピークシフト
WINTER = 131072;//
SPRING = 262144;//
SUMMER = 524288;//
AUTUMN = 1048576;//
OACO2 = 2097152;//
CONSTPOWER = 4194304;//発電電力平準化（一定値）制御（蓄電池）
SMOOTH = 8388608;//出力変動抑制制御（蓄電池）
PEAKCUTALL = 16777216;//全日ピークカット運転（充電時間時もピークカット目標値を適用）
WH_RECOVERY= 33554432;//排熱単独運転　（←2016.2.16 から追加対応）

運転モードは入力で指定された３種類のモードを合成して出力する。
運転モード１が COOL、運転モード２が HEAT の場合 COOL+HEAT=３が送信される。
冷房モードのテストをしたい場合は、運転モード１～３をすべて COOL とする。

＊swcOut からは起動信号 on が常に出力される
＊modOut からは運転モード１～３の合成値が常に出力される

・各媒体の条件の与え方
入力画面の冷温水の水温設定値からの入力項目には、境界条件の条件を設定する。
例えば、熱源への冷温水の戻り水温として 12℃、13℃、14℃のケースを境界条件としたい場合は、冷温水の水温設定値を「12 13 14」のように半角スペースで区切って条件を入力すればよい。流量も条件として与えたい場合は、例えば冷温水の流量設定値を「100 200」のように単位に注意して（この場合の単位は g/s）入力する。
水温設定値を「12 13 14」、流量設定値を「100 200」とした場合、このモジュールはこれらの入力値から組合せ条件のケースを自動的に作成し（この場合３×２＝６通りの組合せ条件を作成）、計算ステップごとに順次 L0_watOutCH ノードの媒体の値へセットする。
計算 step1　L0_watOutCH（12℃、100g/s）
計算 step2　L0_watOutCH（12℃、200g/s）
計算 step3　L0_watOutCH（13℃、100g/s）
計算 step4　L0_watOutCH（13℃、200g/s）
計算 step5　L0_watOutCH（14℃、100g/s）
計算 step6　L0_watOutCH（14℃、200g/s）
すべての組合せ条件のケースが実行された場合は、最初の組合せに戻り繰り返す。

テスト条件が多い場合は、次の入力方法が便利である。

・MMNS 関数と MMND 関数
４個の数値（境界条件の最小値、最大値、基準値と、増加値あるいは分割数）を与えることで境界条件を作成する機能を追加した。
この入力方法の条件として、隣接する２つの値の差（増加幅）は一定としている。
→ただし基準値は例外で、昇順で並べた適切な場所にセットされる。
また、値は小さいものから大きいものへと境界条件値を発生し変化させることとする。
４個の数値の与え方は MMNS 型（MinMaxNvalStep）と MMND 型（MinMaxNvalDiv）の２種類ある。
MMNS 型（MinMaxNvalStep）は、境界条件の最小値、最大値、基準値、増加値
MMND 型（MinMaxNvalDiv）は、境界条件の最小値、最大値、基準値、分割数
の４個の数値を入力する。
基準値には、例えば定格条件時の値を入力することを想定している。

これにより、定格条件時の能力やエネルギー消費量の算定を計算ケースに確実に含めることができる。

具体的な入力は、入力型のキーワード“MMNS”or“MinMaxNvalStep”あるいは“MMND”or“MinMaxNvalDiv”のあとに半角スペース“ ”あるいは半角“_”文字で 4 個の数値を区切って入力する。

例えば、熱源への還り冷水温度のテスト境界条件を 5°Cから 21°Cの範囲で 2 °C刻みとし、基準の還り冷水温度が 12°Cの場合　次のようになる。

→　MMNS 型で入力すると　　⇔　「MMNS_5_21_12_2」or「MMNS 5 21 12 2」

＊下限値 5 °Cから 2 °C刻みで加算 21°Cまで加算する。

→　これを MMND 型入力の場合 ⇔　「MMND_5_21_12_8」or「MMND 5 21 12 8」

＊上下限の差 21－5=16°Cを 8 分割（16÷8＝2°C）する。

→　従来方式の入力の場合　　⇔　「5 7 9 11 12 13 15 17 19 21」

ここに示した 3 通りの入力データの例では、同じ境界条件が作成されることになる。

MMNS 型と MMND 型の 3 番目の数値である基準値は、従来方式の入力の場合の例と同じで、昇順の適切な場所に設定される。

☞4 個の数値の入力は、「最小値、最大値、基準値および値の増加幅あるいは分割数」の順番を守ること。

☞基準値が最小値と最大値の間にない場合でも、昇順の適切な場所に設定される。

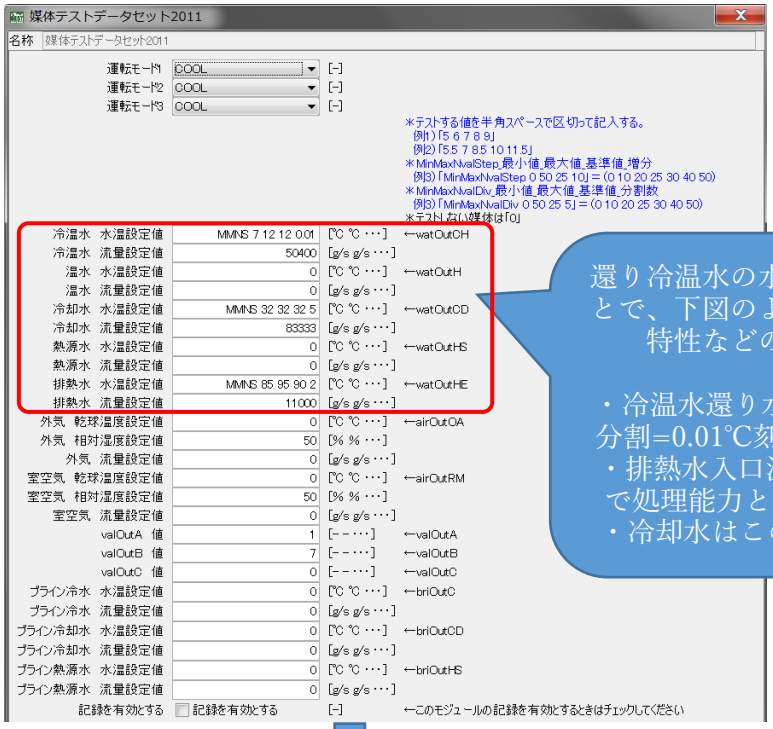
☞最小値、最大値、基準値がすべて同じ値の場合は、増加幅および分割数は無視する。

MMNS 関数と MMND 関数の参考例）

MMNS 型で増加幅を小さくする、あるいは MMND 型で分割数を大きくすることで、境界条件が連続変化しているような計算結果が得られる。

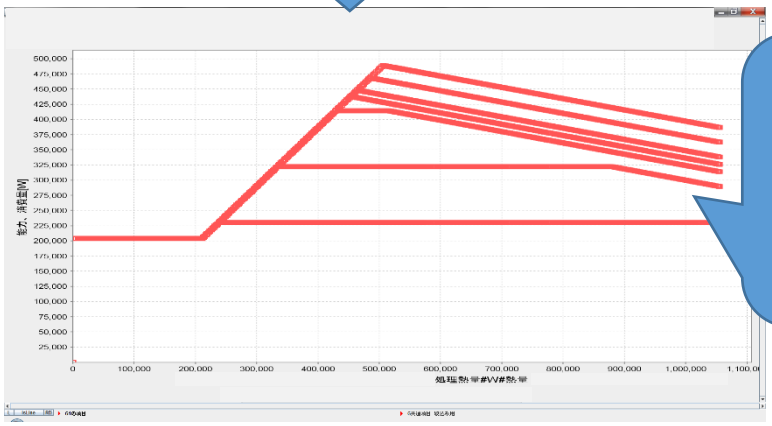
従来の境界条件の設定方法では、機器特性で変曲点があるような場合のチェックが困難であったが、新たな入力方法を使用することで変曲点近傍の計算ケースも簡単に設定でき結果をチェックすることが可能となる。

次の媒体テストデータセットの入力画面は、機器モジュールに対しての境界条件の入力例である。



還り冷温水の水温の増加分を小さくとることで、下図のような変曲点のある部分負荷特性などの確認が簡単にできる！

・冷温水還り水温を 7～12℃の範囲を 500 分割=0.01℃刻みし部分負荷を発生させる
・排熱水入口温度 85,87,89,90,91,93,95℃で処理能力と排熱水消費量を散布図表示
・冷却水はこの場合 32℃で固定値を出力



境界条件の 2 変数を、一方を細かく分割（冷温水還り水温）、他方を粗く分割（排熱水温）しておくと、左図のような表現が結果データの編集をしなくても可能である。

☞機器モジュールの処理熱量と消費量の散布図は、計算結果から別途作図したものである。プロットが近接しているため線図のような表現となっている。

・境界条件の組合わせ数と計算期間に関して

テストする機器モジュールに対して必要な境界条件の媒体ノードを接続し、入力画面でそこから送信する媒体の状態値（水温や流量など）を入力する。

複数の条件がある場合は半角のスペースで区切って（←従来の入力方法）記入する。

媒体テストデータセットモジュールは、計算ステップごとに条件の組合わせをかえて、入力された媒体の条件のすべての組合わせを作りだす。
計算期間内はこれを繰り返す。逆に言えば、すべての組合わせケースを計算できる計算期間を設定する必要がある。

例えば、設定条件が、冷温水入口（温度が 10 個、流量が 5 個)、外気（乾球温度 10 個、相対湿度 1 個、流量 1 個)、負荷率 10 個、出口設定温度 10 個、の設定がされた場合、すべての組合わせは 50000 通り（＝10×5×10×1×1×10×10）で、50000 計算ステップが必要となり、５分間隔の計算では 250000 分≒173.61 日となりこれ以上の計算期間を設定することになる。

（８）データ範囲と範囲外の取扱い

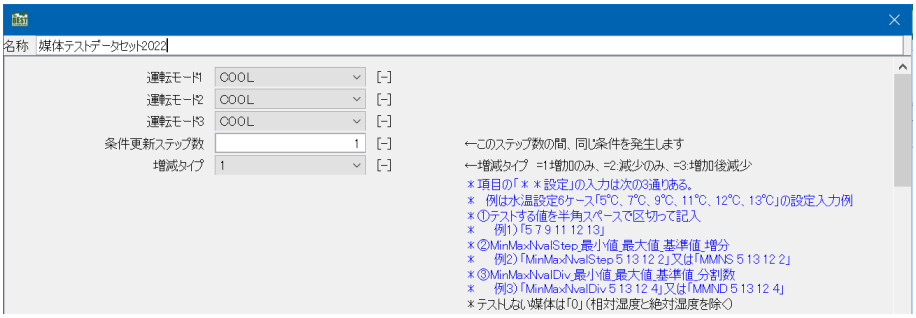
特になし。

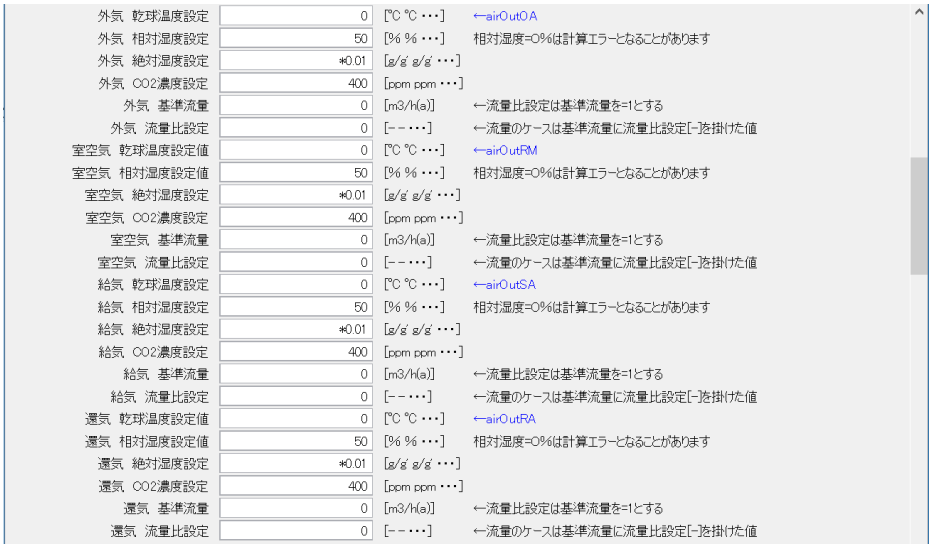
「媒体テストデータセット 2022」（場所：設備 2015／媒体境界条件 2015／）

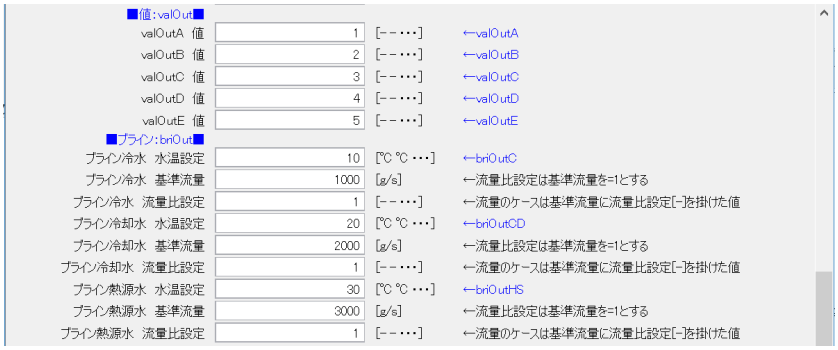
モジュール名	媒体テストデータセット 2022
クラス	MediumTestSetsModule2022

（1）入力画面

・スペック







■値:valOut■

valOutA 値	1	[－－・・・]	←valOutA
valOutB 値	2	[－－・・・]	←valOutB
valOutC 値	3	[－－・・・]	←valOutC
valOutD 値	4	[－－・・・]	←valOutD
valOutE 値	5	[－－・・・]	←valOutE

■ブライン:briOut■

ブライン冷水 水温設定	10	[℃ ℃・・・]	←briOutC
ブライン冷水 基準流量	1000	[g/s]	←流量比設定は基準流量を=1とする
ブライン冷水 流量比設定	1	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
ブライン冷却水 水温設定	20	[℃ ℃・・・]	←briOutCD
ブライン冷却水 基準流量	2000	[g/s]	←流量比設定は基準流量を=1とする
ブライン冷却水 流量比設定	1	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
ブライン熱源水 水温設定	30	[℃ ℃・・・]	←briOutHS
ブライン熱源水 基準流量	3000	[g/s]	←流量比設定は基準流量を=1とする
ブライン熱源水 流量比設定	1	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値

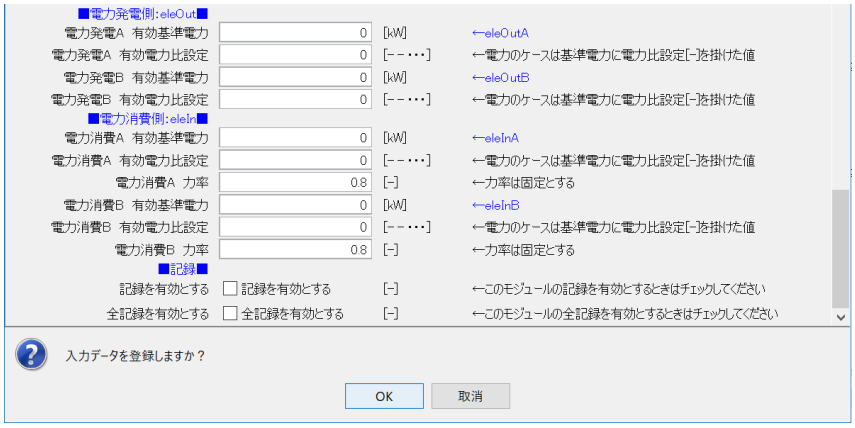
＊各蒸気媒体はゲージ圧力、蒸気流量、凝縮水流量を入力

■蒸気発生側:steOut■

蒸気発生A ゲージ圧力設定	784000	[Pa Pa・・・]	←steOutA
蒸気発生A 蒸気基準流量	0	[g/s]	←流量比設定は基準値を=1とする
蒸気発生A 蒸気流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気発生A 凝縮水基準流量	0	[g/s]	←流量比設定は基準値を=1とする
蒸気発生A 凝縮水流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気発生B ゲージ圧力設定	784000	[Pa Pa・・・]	←steOutB
蒸気発生B 蒸気基準流量	0	[g/s]	←流量比設定は基準値を=1とする
蒸気発生B 蒸気流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気発生B 凝縮水基準流量	0	[g/s]	←流量比設定は基準値を=1とする
蒸気発生B 凝縮水流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気発生C ゲージ圧力設定	784000	[Pa Pa・・・]	←steOutC
蒸気発生C 蒸気基準流量	0	[g/s]	←流量比設定は基準値を=1とする
蒸気発生C 蒸気流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気発生C 凝縮水基準流量	0	[g/s]	←流量比設定は基準値を=1とする
蒸気発生C 凝縮水流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値

■蒸気消費側:steIn■

蒸気消費A ゲージ圧力設定	784000	[Pa Pa・・・]	←steInA
蒸気消費A 蒸気基準流量	0	[g/s]	←流量比設定は基準流量を=1とする
蒸気消費A 蒸気流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気消費A 凝縮水基準流量	0	[g/s]	←流量比設定は基準流量を=1とする
蒸気消費A 凝縮水流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気消費B ゲージ圧力設定	784000	[Pa Pa・・・]	←steInB
蒸気消費B 蒸気基準流量	0	[g/s]	←流量比設定は基準流量を=1とする
蒸気消費B 蒸気流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気消費B 凝縮水基準流量	0	[g/s]	←流量比設定は基準流量を=1とする
蒸気消費B 凝縮水流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気消費C ゲージ圧力設定	784000	[Pa Pa・・・]	←steInC
蒸気消費C 蒸気基準流量	0	[g/s]	←流量比設定は基準流量を=1とする
蒸気消費C 蒸気流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気消費C 凝縮水基準流量	0	[g/s]	←流量比設定は基準流量を=1とする
蒸気消費C 凝縮水流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気消費D ゲージ圧力設定	784000	[Pa Pa・・・]	←steInD
蒸気消費D 蒸気基準流量	0	[g/s]	←流量比設定は基準流量を=1とする
蒸気消費D 蒸気流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値
蒸気消費D 凝縮水基準流量	0	[g/s]	←流量比設定は基準流量を=1とする
蒸気消費D 凝縮水流量比設定	0	[－－・・・]	←流量のケースは基準流量に流量比設定[－]を掛けた値



■電力発電側:eleOut■

電力発電A 有効基準電力	0	[kW]	←eleOutA
電力発電A 有効電力比設定	0	[－－・・・]	←電力のケースは基準電力に電力比設定[－]を掛けた値
電力発電B 有効基準電力	0	[kW]	←eleOutB
電力発電B 有効電力比設定	0	[－－・・・]	←電力のケースは基準電力に電力比設定[－]を掛けた値

■電力消費側:eleIn■

電力消費A 有効基準電力	0	[kW]	←eleInA
電力消費A 有効電力比設定	0	[－－・・・]	←電力のケースは基準電力に電力比設定[－]を掛けた値
電力消費A 力率	0.8	[－]	←力率は固定とする
電力消費B 有効基準電力	0	[kW]	←eleInB
電力消費B 有効電力比設定	0	[－－・・・]	←電力のケースは基準電力に電力比設定[－]を掛けた値
電力消費B 力率	0.8	[－]	←力率は固定とする

■記録■

記録を有効とする	☐ 記録を有効とする	[－]	←このモジュールの記録を有効とするときはチェックしてください
全記録を有効とする	☐ 全記録を有効とする	[－]	←このモジュールの全記録を有効とするときはチェックしてください

入力データを登録しますか？

OK　取消

（2）モジュールの概要

媒体テストデータセットモジュールは、熱源などの機器モジュールや複数のモジュールで構成した部分システムの動作テストを行うときに、その境界条件を与えるモジュールとして利用できるものである。

境界条件はこのモジュールの BEST 媒体の出口接続ノードを接続することで利用できる。

動作をテストする時の境界条件は、そのモジュールの入口側の接続ノードの情報である。

例えば HP チラーのモジュールでは、冷房時の機器特性は冷水の watIn（入口温度、流量）、airIn（外気乾球温度）、modIn（冷暖モード）、swcIn（発停）、valInCapCtrl（容量制御）、出口目標温度（valInSP_watOut）などが境界条件として考えられる。

媒体テストデータセットモジュールは、これらの動作テストに必要な境界条件をまとめて設定するものであり、入力したパラメータのすべての組合わせの境界条件を作成し、それらの媒体出口接続ノードから発信することができる。

設定できる媒体は、冷温水（watOutCH）、温水（watOutH）、冷却水（watOutCD）、熱源水（watOutHS）、排熱水（watOutHE）、外気（airOutOA）、室空気（airOutRM）、給気（airOutSA）、還気（airOutRA）、val × 5（valOutA、valOutB、valOutC、valOutD、valOutE）、ブライン冷水（briOutC）、ブライン冷却水（briOutCD）、ブライン熱源水（briOutHS）、蒸気発生 × 3（steOutA、steOutB、steOutC）、蒸気消費 × 4（steInA、steInB、steInC、steInD）、電力発電 × 2（eleOutA、eleOutB）、電力消費 × 2（eleInA、eleInB）、運転モード × 3（合成モードを modOut）である。

設定できる媒体の管理変数は、wat と bri が水温・流量、air は乾球温度・相対湿度・絶対湿度・流量、val は値、ste はゲージ圧力・蒸気流量・凝縮水流量、電力は有効電力である。

既存の「媒体テストデータセット 2011」モジュールとは次の入力方法を変更している。

・流量のケース設定は流量の値で設定していたものを、基準流量に対する流量比で設定する方法に変更した。

例えば、定格流量 132L/min を 25%、50%、75%、100%の部分負荷流量でテストするために「媒体テストデータセット 2011」では、まず流量を[L/min]から[g/s]に換算して部分負荷流量を算定し[550_1100_1650_2200][g/s,g/s,・・]と与えていた。

「媒体テストデータセット 2022」では基準流量＝132[L/min]（=2200[g/s]）と入力し流量比設定に[0.25_0.5_0.75_1][-,-,・・]とすればよく、負荷流量の算定の手間を不要とした。

・空気 airOut**に給気 aiOutSA と還気 airOutRA を追加し、計 4 種類の設定を可能とした。

・空気 airOut**の管理変数に CO2 濃度を追加し、CO2 濃度の設定を可能とした。

・空気 airOut**の設定項目に絶対湿度[g/g']を追加し、従来の乾球温度と相対湿度による空気状態の設定に加えて、乾球温度と絶対湿度による空気状態の設定、相対湿度と絶対湿度による空気状態の設定を可能とした。

・値 valOut*に valOutD と valOutE を追加し、計 5 種類の設定を可能とした。

・電力発電 eleOutA、eleOutB、電力消費 eleInA、eleInB を新たに追加した。

（3）モジュールの入出力

「ユーザーがスペック画面で入力する設定項目」、「プログラム上での入力項目」、「モジュールからの出力項目」を下記に示す。

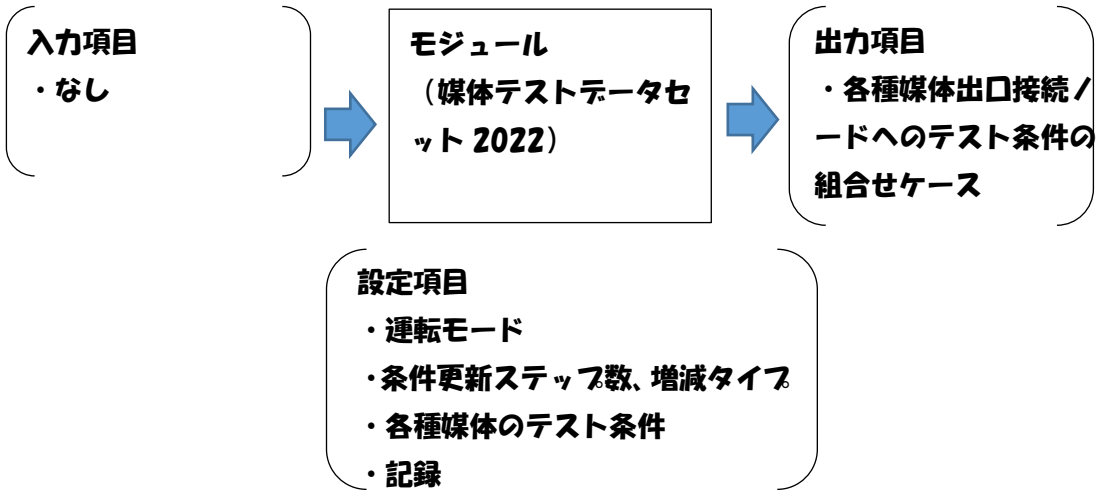


図１　モジュールの設定項目・入力項目・出力項目

（4）スペックの入力項目（図 1 における設定項目）

No	項目	入力内容	記号	デフォルト値	単位	上限値	下限値	変更の重要度	備考
0	名称	String	name		[-]	―	―		
1	運転モード 1 運転モード 2 運転モード 3	備考欄から選択	de_mod1 de_mod2 de_mod3	COOL	[-]	―	―		COOL、HEAT、ICE、RECOVER、C_CHARGE、H_CHARGE、HUMIDIFY、DEHUMIDIFY、DEFROST、VAV、OACUT、FREECOOL、PEAKCUT、NIGHTPURGE、VENTILATE、E_CHARGE、PEAKSHIFT、WH_RECOVERY
2	条件更新ステップ数	String	de_keepSteps	1	[-]	―	1		←このステップ数同じ条件を発生します
3	増減タイプ	1,2,3 から選択	de_ud, typeUp	1	[-]	―	1		←増減タイプ　=1 増加のみ、=2:減少のみ、=3:増加後減少
4	冷温水　水温設定	String	de_T_watOutCH	0	[℃ ℃ ・・・]	―	―		←watOutCH
5	冷温水　基準流量	double	de_M_watOutCH	0	[L/min(w)]	―	―		←流量比設定は基準流量を=1 とする
6	冷温水　流量比設定	String	de_R_watOutCH	0	[－ － ・・・]	―	―		←流量のケースは基準流量に流量比設定[-]を掛けた値
7	温水　水温設定	String	de_T_watOutH	0	[℃ ℃ ・・・]	―	―		←watOutH
8	温水　基準流量	double	de_M_watOutH	0	[L/min(w)]	―	―		←流量比設定は基準流量を=1 とする
9	温水　流量比設定	String	de_R_watOutH	0	[－ － ・・・]	―	―		←流量のケースは基準流量に流量比設定[-]を掛けた値
10	冷却水　水温設定	String	de_T_watOutCD	0	[℃ ℃ ・・・]	―	―		←watOutCD
11	冷却水　基準流量	double	de_M_watOutCD	0	[L/min(w)]	―	―		←流量比設定は基準流量を=1 とする
12	冷却水　流量比設定	String	de_R_watOutCD	0	[－ － ・・・]	―	―		←流量のケースは基準流量に流量比設定[-]を掛けた値
13	熱源水　水温設定	String	de_T_watOutHS	0	[℃ ℃ ・・・]	―	―		←watOutHS
14	熱源水　基準流量	double	de_M_watOutHS	0	[L/min(w)]	―	―		←流量比設定は基準流量を=1 とする
15	熱源水　流量比設定	String	de_R_watOutHS	0	[－ － ・・・]	―	―		←流量のケースは基準流量に流量比設定[-]を掛けた値
16	排熱水　水温設定	String	de_T_watOutHE	0	[℃ ℃ ・・・]	―	―		←watOutHE
17	排熱水　基準流量	double	de_M_watOutHE	0	[L/min(w)]	―	―		←流量比設定は基準流量を=1 とする
18	排熱水　流量比設定	String	de_R_watOutHE	0	[－ － ・・・]	―	―		←流量のケースは基準流量に流量比設定[-]を掛けた値

19	外気　乾球温度設定	String	de_T_airOutOA	0	[℃ ℃ ・・・]	―	―		←airOutOA
20	外気　相対湿度設定	String	de_RH_airOutOA	50	[% % ・・・]	―	―		←airOutOA 相対湿度=０%は計算エラーとなることがあります
21	外気　絶対湿度設定	String	de_X_airOutOA	0.01	[g/g' g/g' ・]	―	―		←airOutOA
22	外気　CO2 濃度設定	String	de_CO2_airOutOA	400	[ppm ppm ・・・]	―	―		←airOutOA
23	外気　基準流量	double	de_M_airOutOA	0	[m3/h(a)]	―	―		←流量比設定は基準流量を=1 とする
24	外気　流量比設定	String	de_R_airOutOA	0	[- - ・・・]	―	―		←流量のケースは基準流量に流量比設定[-]を掛けた値
25	室空気　乾球温度設定	String	de_T_airOutRM	0	[℃ ℃ ・・・]	―	―		←airOutRM
26	室空気　相対湿度設定	String	de_RH_airOutRM	50	[% % ・・・]	―	―		←airOutRM 相対湿度=０%は計算エラーとなることがあります
27	室空気　絶対湿度設定	String	de_X_airOutRM	0.01	[g/g' g/g' ・]	―	―		←airOutRM
28	室空気　CO2 濃度設定	String	de_CO2_airOutRM	400	[ppm ppm ・・・]	―	―		←airOutRM
29	室空気　基準流量	double	de_M_airOutRM	0	[m3/h(a)]	―	―		←流量比設定は基準流量を=1 とする
30	室空気　流量比設定	String	de_R_airOutRM	0	[- - ・・・]	―	―		←流量のケースは基準流量に流量比設定[-]を掛けた値
31	給気　乾球温度設定	String	de_T_airOutSA	0	[℃ ℃ ・・・]	―	―		←airOutSA
32	給気　相対湿度設定	String	de_RH_airOutSA	50	[% % ・・・]	―	―		←airOutSA 相対湿度=０%は計算エラーとなることがあります
33	給気　絶対湿度設定	String	de_X_airOutSA	0.01	[g/g' g/g' ・]	―	―		←airOutSA
34	給気　CO2 濃度設定	String	de_CO2_airOutSA	400	[ppm ppm ・・・]	―	―		←airOutSA
35	給気　基準流量	double	de_M_airOutSA	0	[m3/h(a)]	―	―		←流量比設定は基準流量を=1 とする
36	給気　流量比設定	String	de_R_airOutSA	0	[- - ・・・]	―	―		←流量のケースは基準流量に流量比設定[-]を掛けた値
37	還気　乾球温度設定	String	de_T_airOutRA	0	[℃ ℃ ・・・]	―	―		←airOutRA
38	還気　相対湿度設定	String	de_RH_airOutRA	50	[% % ・・・]	―	―		←airOutRA 相対湿度=０%は計算エラーとなることがあります
39	還気　絶対湿度設定	String	de_X_airOutRA	0.01	[g/g' g/g' ・]	―	―		←airOutRA
40	還気　CO2 濃度設定	String	de_CO2_airOutRA	400	[ppm ppm ・・・]	―	―		←airOutRA
41	還気　基準流量	double	de_M_airOutRA	0	[m3/h(a)]	―	―		←流量比設定は基準流量を=1 とする

42	還気　流量比設定	String	de_R_airOutRA	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
43	valOutA　値	String	de_V_valOutA	0	[- - ・・・]	—	—		←valOutA
44	valOutB　値	String	de_V_valOutB	0	[- - ・・・]	—	—		←valOutB
45	valOutC　値	String	de_V_valOutC	0	[- - ・・・]	—	—		←valOutC
46	valOutD　値	String	de_V_valOutD	0	[- - ・・・]	—	—		←valOutD
47	valOutE　値	String	de_V_valOutE	0	[- - ・・・]	—	—		←valOutE
48	ブライン冷水　水温設定	String	de_T_briOutC	0	[℃ ℃ ・・・]	—	—		←briOutC
49	ブライン冷水　基準流量	double	de_M_briOutC	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
50	ブライン冷水　流量比設定	String	de_M_briOutC	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
51	ブライン冷却水　水温設定	String	de_T_briOutCD	0	[℃ ℃ ・・・]	—	—		←briOutCD
52	ブライン冷却水　基準流量	double	de_M_briOutCD	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
53	ブライン冷却水　流量比設定	String	de_M_briOutCD	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
54	ブライン熱源水　水温設定	String	de_T_briOutHS	0	[℃ ℃ ・・・]	—	—		←briOutHS
55	ブライン熱源水　基準流量	double	de_M_briOutHS	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
56	ブライン熱源水　流量比設定	String	de_M_briOutHS	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
57	蒸気発生 A　ゲージ圧力設定	String	de_Pa_steOutA	784000	[Pa Pa ・・・]	—	—		←steOutA
58	蒸気発生 A　蒸気基準流量	double	de_Ms_steOutA	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
59	蒸気発生 A　蒸気流量比設定	String	de_Ms_steOutA	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
60	蒸気発生 A　凝縮水基準流量	double	de_Mc_steOutA	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
61	蒸気発生 A　凝縮水流量比設定	String	de_Mc_steOutA	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
62	蒸気発生 B　ゲージ圧力設定	String	de_Pa_steOutB	784000	[Pa Pa ・・・]	—	—		←steOutB
63	蒸気発生 B　蒸気基準流量	double	de_Ms_steOutB	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
64	蒸気発生 B　蒸気流量比設定	String	de_Ms_steOutB	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
65	蒸気発生 B　凝縮水基準流量	double	de_Mc_steOutB	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
66	蒸気発生 B　凝縮水流量比設定	String	de_Mc_steOutB	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
67	蒸気発生 C　ゲージ圧力設定	String	de_Pa_steOutC	784000	[Pa Pa ・・・]	—	—		←steOutC

68	蒸気発生 C　蒸気基準流量	double	de_Ms_steOutC	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
69	蒸気発生 C　蒸気流量比設定	String	de_Ms_steOutC	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
70	蒸気発生 C　凝縮水基準流量	double	de_Mc_steOutC	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
71	蒸気発生 C　凝縮水流量比設定	String	de_Mc_steOutC	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
72	蒸気消費 A　ゲージ圧力設定	String	de_Pa_steInA	784000	[Pa Pa ・・・]	—	—		←steInA
73	蒸気消費 A　蒸気基準流量	double	de_Ms_steInA	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
74	蒸気消費 A　蒸気流量比設定	String	de_Ms_steInA	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
75	蒸気消費 A　凝縮水基準流量	double	de_Mc_steInA	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
76	蒸気消費 A　凝縮水流量比設定	String	de_Mc_steInA	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
77	蒸気消費 B　ゲージ圧力設定	String	de_Pa_steInB	784000	[Pa Pa ・・・]	—	—		←steInB
78	蒸気消費 B　蒸気基準流量	double	de_Ms_steInB	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
79	蒸気消費 B　蒸気流量比設定	String	de_Ms_steInB	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
80	蒸気消費 B　凝縮水基準流量	double	de_Mc_steInB	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
81	蒸気消費 B　凝縮水流量比設定	String	de_Mc_steInB	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
82	蒸気消費 C　ゲージ圧力設定	String	de_Pa_steInC	784000	[Pa Pa ・・・]	—	—		←steInC
83	蒸気消費 C　蒸気基準流量	double	de_Ms_steInC	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
84	蒸気消費 C　蒸気流量比設定	String	de_Ms_steInC	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
85	蒸気消費 C　凝縮水基準流量	double	de_Mc_steInC	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
86	蒸気消費 C　凝縮水流量比設定	String	de_Mc_steInC	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
87	蒸気消費 D　ゲージ圧力設定	String	de_Pa_steInD	784000	[Pa Pa ・・・]	—	—		←steInD
88	蒸気消費 D　蒸気基準流量	double	de_Ms_steInD	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
89	蒸気消費 D　蒸気流量比設定	String	de_Ms_steInD	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値
90	蒸気消費 D　凝縮水基準流量	double	de_Mc_steInD	0	[g/s]	—	—		←流量比設定は基準流量を=1 とする
91	蒸気消費 D　凝縮水流量比設定	String	de_Mc_steInD	0	[- - ・・・]	—	—		←流量のケースは基準流量に流量比設定[-]を掛けた値

92	電力発電 A　有効基準電力	double	de_Wa_eleOutA	0	[W]	—	—		←eleOutA 電力比設定は基準電力を=1 とする
93	電力発電 A　有効電力比設定	String	de_R_eleOutA	0	[－－・・・]	—	—		←電力のケースは基準電力に電力比設定[-]を掛けた値
94	電力発電 B　有効基準電力	double	de_Wa_eleOutB	0	[W]	—	—		←eleOutB 電力比設定は基準電力を=1 とする
95	電力発電 B　有効電力比設定	String	de_R_eleOutB	0	[－－・・・]	—	—		←電力のケースは基準電力に電力比設定[-]を掛けた値
96	電力消費 A　有効基準電力	double	de_Wa_eleInA	0	[W]	—	—		←eleInA 電力比設定は基準電力を=1 とする
97	電力消費 A　有効電力比設定	String	de_R_eleInA	0	[－－・・・]	—	—		←電力のケースは基準電力に電力比設定[-]を掛けた値
98	電力消費 A　力率	double	de_PF_eleInA	0.8	[-]	—	—		
99	電力消費 B　有効基準電力	double	de_Wa_eleInB	0	[W]	—	—		←eleInB 電力比設定は基準電力を=1 とする
100	電力消費 B　有効電力比設定	String	de_R_eleInB	0	[－－・・・]	—	—		←電力のケースは基準電力に電力比設定[-]を掛けた値
101	電力消費 B　力率	double	de_PF_eleInB	0.8	[-]	—	—		
102	記録を有効とする	boolean	isRecord	false	[-]	—	—		
103	全記録を有効とする	boolean	isRecordALL	false	[-]	—	—		

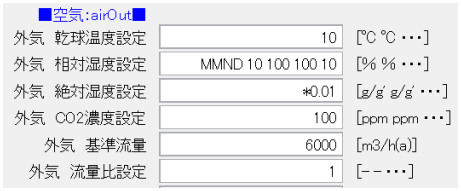
☞　空気媒体の外気、室空気、給気、還気の状態は、各媒体の３種類の設定項目（乾球温度、相対湿度、絶対湿度）の中の２種類の組合せで決める。従来は乾球温度と湿球温度（下図例１）で決めていたが、乾球温度と絶対湿度（例２）、相対湿度と絶対湿度（例３）という決め方が可能となった。

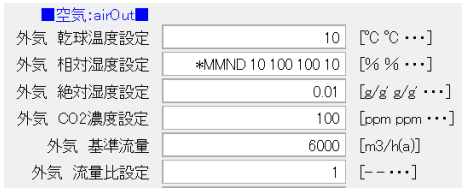
入力方法は、乾球温度、相対湿度、絶対湿度の中で１種類だけ無効とする入力項目の先頭文字を「＊」（半角）とする。

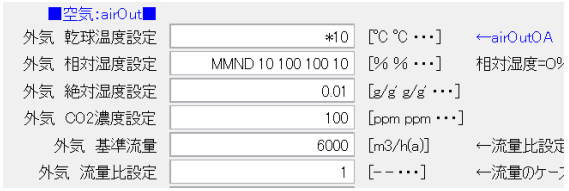
乾球温度、相対湿度、絶対湿度の入力文字の先頭が「＊」であるものが２種類以上にある場合は計算実行時にエラーとなる。

乾球温度、相対湿度、絶対湿度の入力文字の先頭が「＊」であるものが無い場合は従来の設定方法の乾球温度と相対湿度で状態を決める。

例１）乾球温度と相対湿度で状態を決定　　例２）乾球温度と絶対湿度で状態を決定　　例３）相対湿度と絶対湿度で状態を決定







（５）シーケンス接続（図１における入力項目、出力項目）

No	項目	接続端子名	記号	単位	ノード区分	媒体区分	InOut 区分	備考
1	記録出口	L2_recOut	R_NODE	-	記録	メモリ	出口	記録の出口
2	On/Off 信号	L1_swcOut	swcOut		制御	制御	出口	常に on 信号を出力する
3	運転モード	L1_modOut	modOut		制御	制御モード	出口	入力画面の運転モード 1,2,3 を合成したモード信号を出力する
4	外気	LO_airOutOA	airOutOA		状態	空気	出口	外気の乾球温度、相対湿度および流量の入力値から組合せ条件を作成したものを出力する。
5	室空気	LO_airOutRM	airOutRM		状態	空気	出口	室空気の乾球温度、相対湿度および流量の入力値から組合せ条件を作成したものを出力する。
6	給気	LO_airOutSA	airOutSA		状態	空気	出口	給気の乾球温度、相対湿度、CO2 濃度および流量の入力値から組合せ条件を作成したものを出力する。
7	還気	LO_airOutRA	airOutRA		状態	空気	出口	還気の乾球温度、相対湿度、CO2 濃度および流量の入力値から組合せ条件を作成したものを出力する。
8	冷温水	LO_watOutCH	watOutCH		状態	水	出口	冷温水の水温と流量の入力値から組合せ条件を作成したものを出力する。
9	温水	LO_watOutH	watOutH		状態	水	出口	温水の水温と流量の入力値から組合せ条件を作成したものを出力する。
10	冷却水	LO_watOutCD	watOutCD		状態	水	出口	冷却水の水温と流量の入力値から組合せ条件を作成したものを出力する。
11	熱源水	LO_watOutHS	watOutHS		状態	水	出口	熱源水の水温と流量の入力値から組合せ条件を作成したものを出力する。
12	排熱水	LO_watOutHE	watOutHE	-	状態	水	出口	排熱水の水温と流量の入力値から組合せ条件を作成したものを出力する。

13	値 val	LO_valOutA LO_valOutB LO_valOutC LO_valOutD LO_valOutE	valOutA valOutB valOutC valOutD valOutE	-	状態	Double 値	出口	値の入力値から条件を作成したものを出力する。
14	ブライン冷水	LO_briOutC	briOutC	-	状態	ブライン	出口	ブライン冷水の水温と流量の入力値から組合せ条件を作成したものを出力する。
15	ブライン冷却水	LO_briOutCD	briOutCD	-	状態	ブライン	出口	ブライン冷却水の水温と流量の入力値から組合せ条件を作成したものを出力する。
16	ブライン熱源水	LO_briOutHS	briOutHS	-	状態	ブライン	出口	ブライン熱源水の水温と流量の入力値から組合せ条件を作成したものを出力する。
17	蒸気発生	LO_steOutA LO_steOutB LO_steOutC	steOutA steOutB steOutC	-	状態	蒸気	出口	蒸気のゲージ圧力、蒸気流量と凝縮水流量の入力値から組合せ条件を作成したものを出力する。
18	蒸気消費	LO_steInA LO_steInB LO_steInC LO_steInD	steInA steInB steInC steInD	-	状態	蒸気	出口	蒸気のゲージ圧力、蒸気流量と凝縮水流量の入力値から組合せ条件を作成したものを出力する。
19	電力発電	LO_eleOutA LO_eleOutB	eleOutA eleOutB	-	状態	電力	出口	電力の有効電力の入力値から条件を作成したものを出力する。（力率＝１）
20	電力消費	LO_eleInA LO_eleInB	eleInA eleInB	-	状態	電力	出口	電力消費の有効電力の入力値から条件を作成したものを出力する。（力率＝入力値で固定）

（6）記録項目

No.	項目	出力内容	単位	分類
1	MTS_Message#-#-	メッセージ	-	メッセージ
2	MTS_caseNumber#-#-	境界条件の組合せ条件データのケース通し番号	-	出口
3	MTS_01 冷温水温度#℃#温度	LO_watOutCH の温度	℃	出口
4	MTS_01 冷温水流量#g/s#質量流量	LO_watOutCH の流量	g/s	出口
5	MTS_01 冷温水流量比#-#-	LO_watOutCH の基準流量比	-	出口
6	MTS_02 温水温度#℃#温度	LO_watOutH の温度	℃	出口
7	MTS_02 温水流量#g/s#質量流量	LO_watOutH の流量	g/s	出口
8	MTS_02 温水流量比#-#-	LO_watOutH の基準流量比	-	出口
9	MTS_03 冷却水温度#℃#温度	LO_watOutCD の温度	℃	出口
10	MTS_03 冷却水流量#g/s#質量流量	LO_watOutCD の流量	g/s	出口
11	MTS_03 冷却水流量比#-#-	LO_watOutCD の基準流量比	-	出口
12	MTS_04 熱源水温度#℃#温度	LO_watOutHS の温度	℃	出口
13	MTS_04 熱源水流量#g/s#質量流量	LO_watOutHS の流量	g/s	出口
14	MTS_04 熱源水流量比#-#-	LO_watOutHS の基準流量比	-	出口
15	MTS_05 廃熱水温度#℃#温度	LO_watOutHE の温度	℃	出口
16	MTS_05 廃熱水流量#g/s#質量流量	LO_watOutHE の流量	g/s	出口
17	MTS_05 廃熱水流量比#-#-	LO_watOutHE の基準流量比	-	出口
18	MTS_11 外気乾球温度#℃#温度	LO_airOutOA の乾球温度	℃	出口
19	MTS_11 外気絶対湿度#g/g#絶対湿度	LO_airOutOA の絶対湿度	g/g'	出口
20	MTS_11 外気相対湿度#%#相対湿度	LO_airOutOA の相対湿度	%	出口
21	MTS_11 外気流量#g/s#質量流量	LO_airOutOA の風量	g/s	出口
22	MTS_11 外気流量比#-#-	LO_airOutOA の基準風量比	-	出口
23	MTS_11 外気湿球温度#℃#温度	LO_airOutOA の湿球温度	℃	出口
24	MTS_11 外気露点温度#℃#温度	LO_airOutOA の露点温度	℃	出口
25	MTS_11 外気比エンタルピ#J/g'#比エンタルピ	LO_airOutOA の比エンタルピ	J/g'	出口
26	MTS_11 外気 CO2 濃度#ppm#CO2 濃度	LO_airOutOA の CO2 濃度	ppm	出口
27	MTS_12 室空気乾球温度#℃#温度	LO_airOutRM の乾球温度	℃	出口
28	MTS_12 室空気絶対湿度#g/g#絶対湿度	LO_airOutRM の絶対湿度	g/g'	出口
29	MTS_12 室空気相対湿度#%#相対湿度	LO_airOutRM の相対湿度	%	出口
30	MTS_12 室空気流量#g/s#質量流量	LO_airOutRM の風量	g/s	出口
31	MTS_12 室空気流量比#-#-	LO_airOutRM の基準風量比	-	出口

32	MTS_12 室空気湿球温度#℃#温度	LO_airOutRM の湿球温度	℃	出口
33	MTS_12 室空気露点温度#℃#温度	LO_airOutRM の露点温度	℃	出口
34	MTS_12 室空気比エンタルピ#J/g'#比エンタルピ	LO_airOutRM の比エンタルピ	J/g'	出口
35	MTS_12 室空気 CO2 濃度#ppm#CO2 濃度	LO_airOutRM の CO2 濃度	ppm	出口
36	MTS_13 給気乾球温度#℃#温度	LO_airOutSA の乾球温度	℃	出口
37	MTS_13 給気絶対湿度#g/g#絶対湿度	LO_airOutSA の絶対湿度	g/g'	出口
38	MTS_13 給気相対湿度#%#相対湿度	LO_airOutSA の相対湿度	%	出口
39	MTS_13 給気流量#g/s#質量流量	LO_airOutSA の風量	g/s	出口
40	MTS_13 給気流量比#-#-	LO_airOutSA の基準風量比	-	出口
41	MTS_13 給気湿球温度#℃#温度	LO_airOutSA の湿球温度	℃	出口
42	MTS_13 給気露点温度#℃#温度	LO_airOutSA の露点温度	℃	出口
43	MTS_13 給気比エンタルピ#J/g'#比エンタルピ	LO_airOutSA の比エンタルピ	J/g'	出口
44	MTS_13 給気 CO2 濃度#ppm#CO2 濃度	LO_airOutSA の CO2 濃度	ppm	出口
45	MTS_14 還気乾球温度#℃#温度	LO_airOutRA の乾球温度	℃	出口
46	MTS_14 還気絶対湿度#g/g#絶対湿度	LO_airOutRA の絶対湿度	g/g'	出口
47	MTS_14 還気相対湿度#%#相対湿度	LO_airOutRA の相対湿度	%	出口
48	MTS_14 還気流量#g/s#質量流量	LO_airOutRA の風量	g/s	出口
49	MTS_14 還気流量比#-#-	LO_airOutRA の基準風量比	-	出口
50	MTS_14 還気湿球温度#℃#温度	LO_airOutRA の湿球温度	℃	出口
51	MTS_14 還気露点温度#℃#温度	LO_airOutRA の露点温度	℃	出口
52	MTS_14 還気比エンタルピ#J/g'#比エンタルピ	LO_airOutRA の比エンタルピ	J/g'	出口
53	MTS_14 還気 CO2 濃度#ppm#CO2 濃度	LO_airOutRA の CO2 濃度	ppm	出口
54	MTS_21valA#-#-	LO_valOutA の値	-	出口
55	MTS_22valB#-#-	LO_valOutB の値	-	出口
56	MTS_23valC#-#-	LO_valOutC の値	-	出口
57	MTS_24valD#-#-	LO_valOutD の値	-	出口
58	MTS_25valE#-#-	LO_valOutE の値	-	出口
59	MTS_31 ブライン冷水温度#℃#温度	LO_briOutC の温度	℃	出口
60	MTS_31 ブライン冷水流量#g/s#質量流量	LO_briOutC の流量	g/s	出口
61	MTS_31 ブライン冷水流量比#-#-	LO_briOutC の基準流量比	-	出口
62	MTS_32 ブライン冷却水温度#℃#温度	LO_briOutCD の温度	℃	出口
63	MTS_32 ブライン冷却水流量#g/s#質量流量	LO_briOutCD の流量	g/s	出口
64	MTS_32 ブライン冷却水流量比#-#-	LO_briOutCD の基準流量比	-	出口

65	MTS_33 ブライン熱源水温度#℃#温度	LO_briOutHS の温度	℃	出口
66	MTS_33 ブライン熱源水流量#g/s#質量流量	LO_briOutHS の流量	g/s	出口
67	MTS_33 ブライン熱源水流量比#-#-	LO_briOutHS の基準流量比	-	出口
68	MTS_41 蒸気発生 A 圧力#Pa#圧力	LO_steOutA の圧力	Pa	出口
69	MTS_41 蒸気発生 A 蒸気流量#g/s#質量流量	LO_steOutA の蒸気流量	g/s	出口
70	MTS_41 蒸気発生 A 蒸気流量比#-#-	LO_steOutA の蒸気基準流量比	-	出口
71	MTS_41 蒸気発生 A 凝縮水流量#g/s#質量流量	LO_steOutA の凝縮水流量	g/s	出口
72	MTS_41 蒸気発生 A 凝縮水流量比#-#-	LO_steOutA の凝縮水基準流量比	-	出口
73	MTS_42 蒸気発生 B 圧力#Pa#圧力	LO_steOutB の圧力	Pa	出口
74	MTS_42 蒸気発生 B 蒸気流量#g/s#質量流量	LO_steOutB の蒸気流量	g/s	出口
75	MTS_42 蒸気発生 B 蒸気流量比#-#-	LO_steOutB の蒸気基準流量比	-	出口
76	MTS_42 蒸気発生 B 凝縮水流量#g/s#質量流量	LO_steOutB の凝縮水流量	g/s	出口
77	MTS_42 蒸気発生 B 凝縮水流量比#-#-	LO_steOutB の凝縮水基準流量比	-	出口
78	MTS_43 蒸気発生 C 圧力#Pa#圧力	LO_steOutC の圧力	Pa	出口
79	MTS_43 蒸気発生 C 蒸気流量#g/s#質量流量	LO_steOutC の蒸気流量	g/s	出口
80	MTS_43 蒸気発生 C 蒸気流量比#-#-	LO_steOutC の蒸気基準流量比	-	出口
81	MTS_43 蒸気発生 C 凝縮水流量#g/s#質量流量	LO_steOutC の凝縮水流量	g/s	出口
82	MTS_43 蒸気発生 C 凝縮水流量比#-#-	LO_steOutC の凝縮水基準流量比	-	出口
83	MTS_51 蒸気消費 A 圧力#Pa#圧力	LO_steInA の圧力	Pa	入口（値の出口）
84	MTS_51 蒸気消費 A 蒸気流量#g/s#質量流量	LO_steInA の蒸気流量	g/s	入口（値の出口）
85	MTS_51 蒸気消費 A 蒸気流量比#-#-	LO_steInA の蒸気基準流量比	-	入口（値の出口）
56	MTS_51 蒸気消費 A 凝縮水流量#g/s#質量流量	LO_steInA の凝縮水流量	g/s	入口（値の出口）
87	MTS_51 蒸気消費 A 凝縮水流量比#-#-	LO_steInA の凝縮水基準流量比	-	入口（値の出口）
88	MTS_52 蒸気消費 B 圧力#Pa#圧力	LO_steInB の圧力	Pa	入口（値の出口）
89	MTS_52 蒸気消費 B 蒸気流量#g/s#質量流量	LO_steInB の蒸気流量	g/s	入口（値の出口）
90	MTS_52 蒸気消費 B 蒸気流量比#-#-	LO_steInB の蒸気基準流量比	-	入口（値の出口）
91	MTS_52 蒸気消費 B 凝縮水流量#g/s#質量流量	LO_steInB の凝縮水流量	g/s	入口（値の出口）
92	MTS_52 蒸気消費 B 凝縮水流量比#-#-	LO_steInB の凝縮水基準流量比	-	入口（値の出口）
93	MTS_53 蒸気消費 C 圧力#Pa#圧力	LO_steInC の圧力	Pa	入口（値の出口）
94	MTS_53 蒸気消費 C 蒸気流量#g/s#質量流量	LO_steInC の蒸気基準流量	g/s	入口（値の出口）
95	MTS_53 蒸気消費 C 蒸気流量比#-#-	LO_steInC の蒸気流量比	-	入口（値の出口）
96	MTS_53 蒸気消費 C 凝縮水流量#g/s#質量流量	LO_steInC の凝縮水流量	g/s	入口（値の出口）
97	MTS_53 蒸気消費 C 凝縮水流量比#-#-	LO_steInC の凝縮水基準流量比	-	入口（値の出口）

98	MTS_54 蒸気消費 D 圧力#Pa#圧力	LO_steInD の圧力	Pa	入口（値の出口）
99	MTS_54 蒸気消費 D 蒸気流量#g/s#質量流量	LO_steInD の蒸気流量	g/s	入口（値の出口）
100	MTS_54 蒸気消費 D 蒸気流量比#-#-	LO_steInD の蒸気基準流量比	-	入口（値の出口）
101	MTS_54 蒸気消費 D 凝縮水流量#g/s#質量流量	LO_steInD の凝縮水流量	g/s	入口（値の出口）
102	MTS_54 蒸気消費 D 凝縮水流量比#-#-	LO_steInD の凝縮水基準流量比	-	入口（値の出口）
103	MTS_61 電力発電 A 有効電力#W#発電電力-	LO_eleOutA の有効電力	W	出口
104	MTS_61 電力発電 B 有効電力#W#発電電力-	LO_eleOutB の有効電力	W	出口
105	MTS_61 電力消費 A 有効電力#W#消費電力-	LO_eleInA の有効電力	W	入口
106	MTS_61 電力消費 B 有効電力#W#消費電力-	LO_eleInB の有効電力	W	入口

（7）計算フロー・計算内容

・運転モード（L1_modOut)、発停信号(L1_swcOut)

運転モード mod は種類が多く下記の通りとなる。

Airmod クラスで定義されたモード

COOL = 1;//冷房運転　熱源、空調機
HEAT = 2;//暖房運転　熱源、空調機
ICE = 4;//製氷運転　熱源
RECOVER = 8;//熱回収　熱源、熱交換器、空調機、全熱交換器
C_CHARGE = 16;//蓄熱運転　熱源　冷房
H_CHARGE = 32;//蓄熱運転　熱源　暖房
HUMIDIFY = 64;//加湿　空調機、加湿器
DEHUMIDIFY = 128;//除湿　空調機、加湿器
DEFROST = 256;//除霜　熱源
VAV = 512;//VAV　空調機、VAV ユニット
OACUT = 1024;//外気カット
FREECOOL = 2048;//外気冷房
PEAKCUT = 4096;//ピークカット
NIGHTPURGE = 8192;//ナイトパージ
VENTILATE = 16384;//換気運転　空調機
E_CHARGE = 32768;//蓄電
PEAKSHIFT = 65536;//ピークシフト
WINTER = 131072;//
SPRING = 262144;//
SUMMER = 524288;//
AUTUMN = 1048576;//
OACO2 = 2097152;//
CONSTPOWER = 4194304;//発電電力平準化（一定値）制御（蓄電池）
SMOOTH = 8388608;//出力変動抑制制御（蓄電池）
PEAKCUTALL = 16777216;//全日ピークカット運転（充電時間時もピークカット目標値を適用）
WH_RECOVERY= 33554432;//排熱単独運転　（←2016.2.16 から追加対応）

運転モードは入力で指定された３種類のモードを合成して出力する。
運転モード１が COOL、運転モード２が HEAT の場合 COOL+HEAT=３が送信される。
冷房モードのテストをしたい場合は、運転モード１～３をすべて COOL とする。

＊swcOut からは起動信号 on が常に出力される
＊modOut からは運転モード１～３の合成値が常に出力される

・各媒体の条件の与え方
入力画面の冷温水の水温設定値からの入力項目には、境界条件の条件を設定する。
例えば、熱源への冷温水の戻り水温として 12℃、13℃、14℃のケースを境界条件としたい場合は、冷温水の水温設定値を「12 13 14」のように半角スペースで区切って条件を入力すればよい。流量も条件として与えたい場合は、例えば冷温水の流量設定値を「100 200」のように単位に注意して（この場合の単位は g/s）入力する。
水温設定値を「12 13 14」、流量設定値を「100 200」とした場合、このモジュールはこれらの入力値から組合せ条件のケースを自動的に作成し（この場合３×２＝６通りの組合せ条件を作成）、計算ステップごとに順次 L0_watOutCH ノードの媒体の値へセットする。
計算 step1　L0_watOutCH（12℃、100g/s）
計算 step2　L0_watOutCH（12℃、200g/s）
計算 step3　L0_watOutCH（13℃、100g/s）
計算 step4　L0_watOutCH（13℃、200g/s）
計算 step5　L0_watOutCH（14℃、100g/s）
計算 step6　L0_watOutCH（14℃、200g/s）
すべての組合せ条件のケースが実行された場合は、最初の組合せに戻り繰り返す。

テスト条件が多い場合は、次の入力方法が便利である。

・MMNS 関数と MMND 関数
４個の数値（境界条件の最小値、最大値、基準値と、増加値あるいは分割数）を与えることで境界条件を作成する機能を追加した。
この入力方法の条件として、隣接する２つの値の差（増加幅）は一定としている。
→ただし基準値は例外で、昇順で並べた適切な場所にセットされる。
また、値は小さいものから大きいものへと境界条件値を発生し変化させることとする。
４個の数値の与え方は MMNS 型（MinMaxNvalStep）と MMND 型（MinMaxNvalDiv）の２種類ある。
MMNS 型（MinMaxNvalStep）は、境界条件の最小値、最大値、基準値、増加値
MMND 型（MinMaxNvalDiv）は、境界条件の最小値、最大値、基準値、分割数
の４個の数値を入力する。
基準値には、例えば定格条件時の値を入力することを想定している。

これにより、定格条件時の能力やエネルギー消費量の算定を計算ケースに確実に含めることができる。

具体的な入力は、入力型のキーワード“MMNS”or“MinMaxNvalStep”あるいは“MMND”or“MinMaxNvalDiv”のあとに半角スペース“ ”あるいは半角“_”文字で 4 個の数値を区切って入力する。

例えば、熱源への還り冷水温度のテスト境界条件を 5°Cから 21°Cの範囲で 2 °C刻みとし、基準の還り冷水温度が 12°Cの場合　次のようになる。

→　MMNS 型で入力すると　　⇔　「MMNS_5_21_12_2」or「MMNS 5 21 12 2」

＊下限値 5 °Cから 2 °C刻みで加算 21°Cまで加算する。

→　これを MMND 型入力の場合 ⇔　「MMND_5_21_12_8」or「MMND 5 21 12 8」

＊上下限の差 21－5=16°Cを 8 分割（16÷8＝2°C）する。

→　従来方式の入力の場合　　⇔　「5 7 9 11 12 13 15 17 19 21」

ここに示した 3 通りの入力データの例では、同じ境界条件が作成されることになる。

MMNS 型と MMND 型の 3 番目の数値である基準値は、従来方式の入力の場合の例と同じで、昇順の適切な場所に設定される。

☞4 個の数値の入力は、「最小値、最大値、基準値および値の増加幅あるいは分割数」の順番を守ること。

☞基準値が最小値と最大値の間にない場合でも、昇順の適切な場所に設定される。

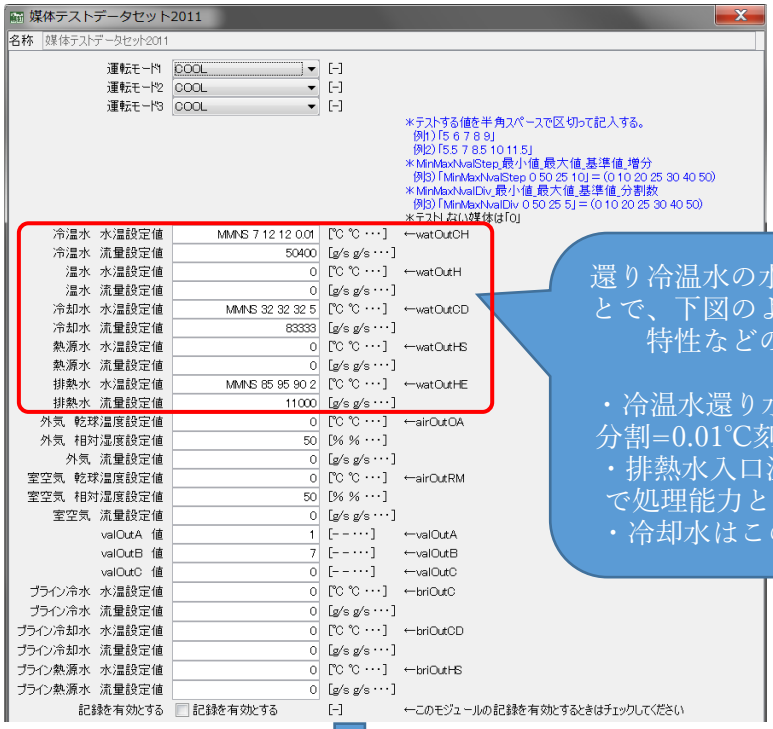
☞最小値、最大値、基準値がすべて同じ値の場合は、増加幅および分割数は無視する。

MMNS 関数と MMND 関数の参考例）

MMNS 型で増加幅を小さくする、あるいは MMND 型で分割数を大きくすることで、境界条件が連続変化しているような計算結果が得られる。

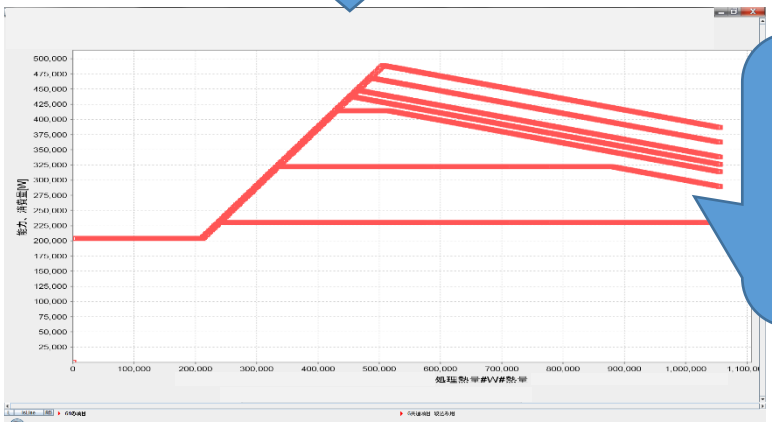
従来の境界条件の設定方法では、機器特性で変曲点があるような場合のチェックが困難であったが、新たな入力方法を使用することで変曲点近傍の計算ケースも簡単に設定でき結果をチェックすることが可能となる。

次の媒体テストデータセットの入力画面は、機器モジュールに対しての境界条件の入力例である。（「媒体テストデータセット 2011」モジュールの計算例）



還り冷温水の水温の増加分を小さくとることで、下図のような変曲点のある部分負荷特性などの確認が簡単にできる！

・冷温水還り水温を 7～12℃の範囲を 500 分割=0.01℃刻みし部分負荷を発生させる
・排熱水入口温度 85,87,89,90,91,93,95℃で処理能力と排熱水消費量を散布図表示
・冷却水はこの場合 32℃で固定値を出力



境界条件の 2 変数を、一方を細かく分割（冷温水還り水温）、他方を粗く分割（排熱水温）しておくと、左図のような表現が結果データの編集をしなくても可能である。

☞機器モジュールの処理熱量と消費量の散布図は、計算結果から別途作図したものである。プロットが近接しているため線図のような表現となっている。

・境界条件の組合わせ数と計算期間に関して

テストする機器モジュールに対して必要な境界条件の媒体ノードを接続し、入力画面でそこから送信する媒体の状態値（水温や流量など）を入力する。

複数の条件がある場合は半角のスペースで区切って（←従来の入力方法）記入する。

媒体テストデータセットモジュールは、計算ステップごとに条件の組合わせをかえて、入力された媒体の条件のすべての組合わせを作りだす。
計算期間内はこれを繰り返す。逆に言えば、すべての組合わせケースを計算できる計算期間を設定する必要がある。

例えば、設定条件が、冷温水入口（温度が 10 個、流量が 5 個)、外気（乾球温度 10 個、相対湿度 1 個、流量 1 個)、負荷率 10 個、出口設定温度 10 個、の設定がされた場合、すべての組合わせは 50000 通り（＝10×5×10×1×1×10×10）で、50000 計算ステップが必要となり、５分間隔の計算では 250000 分≒173.61 日となりこれ以上の計算期間を設定することになる。

（８）データ範囲と範囲外の取扱い

特になし。