KARACRIX 入門実用ガイド

7章　温室換気システム

（章別取扱説明書 v3.00）

株式会社 エスアイ創房

■改定履歴

第 3.00 版　2009/12/01

■ おことわり

(1) 本書内容の一部又は全部を、無断で他に転載することは禁止されています。

(2) 本書内容は、将来予告無く変更する場合があります。

目次

7章　温室換気システムの作成 (マルチタスク実感応用編)

本章では、温度センサを使用して温室などの温度監視と制御を行なうシステムを構築してみます。
また、Linux(UNIX 系)OS の基本機能としての複数の処理を同時に並行して行なうマルチタスク処理を、アプリケーションのレベルで実感していただくことも目標にしています。

7.1　システム設計

システム構成は、温度をセンサで計測し、管理(目標)温度を定めこれに保てる様に換気扇の ON/OFF 制御を行ってみます。また、温度が換気扇でコントロールできる上限値を超えた場合には警報ブザーを鳴らすと共に、インターネットを使用して警報 E メールを発信させます。

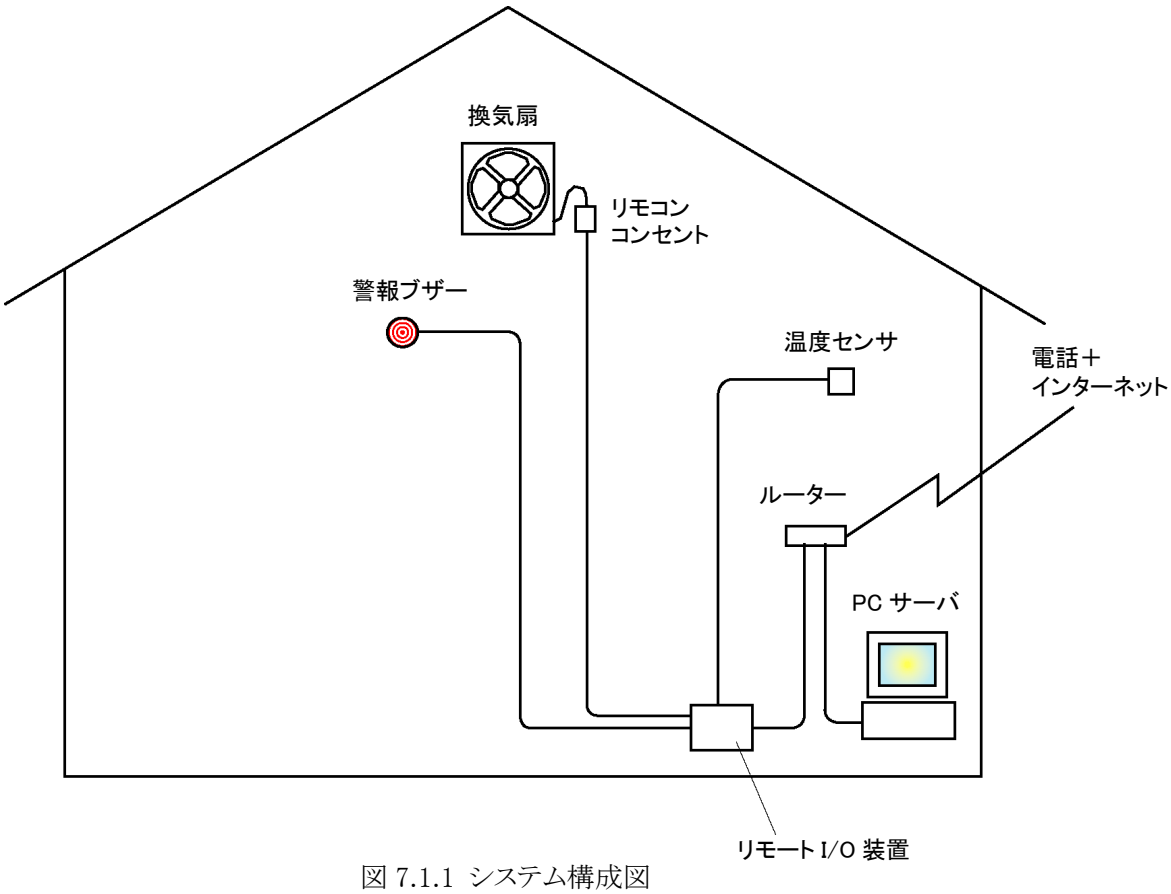


図 7.1.1 システム構成図

7.2　センサ、アクチュエータの設置

リモート I/O 装置には、前章に引き続き KaracriBoard-TK0040A（以下、TK0040A）を使用します。本章では、温度センサ入力としてアナログ入力 1 点、リモコンリレーコンセント制御及びブザー出力用としてデジタル出力 2 点を使用します。

◇使用部品の紹介

●温度センサ

温度センサには、4章で紹介している-25℃～+105℃計測可能な AD592 を使用します。
このセンサは電流出力型と言ってセンサの配線を長く引き回しても配線抵抗による誤差がありません。
但し、リモートI/O装置に入力する場合には電流を電圧に変換する必要がありますので下図に示すように高精度(0.5%)の抵抗を組み合わせて使用します。（ノイズも多く拾うのでフィルタ用コンデンサを併設します）

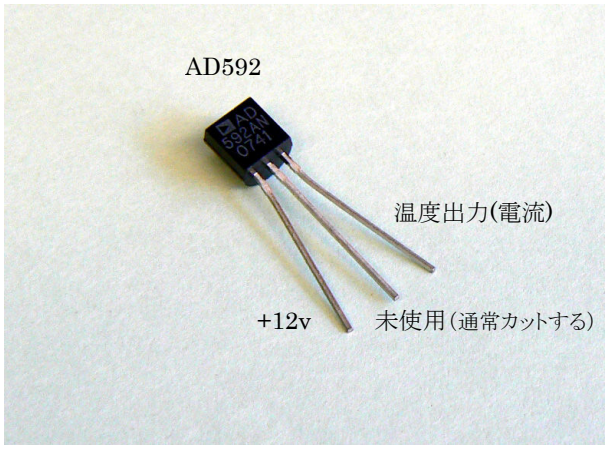


写真 7.2.1　温度センサ（AD592 / TO92 パッケージ）

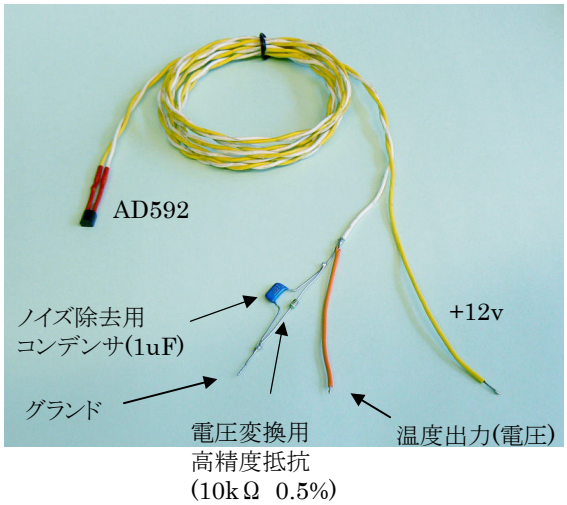


写真 7.2.2　温度センサ配線

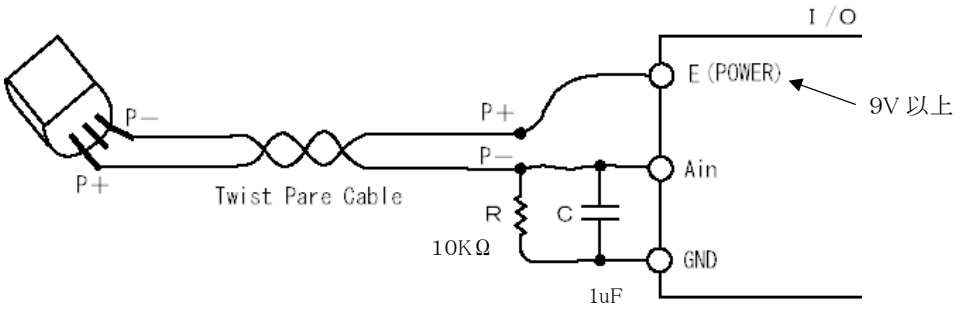


図 7.2.1　温度センサ実体配線図

●センサー保護管

センサー保護管として、防水可能な弊社製 KB-TO92-S730(ステンレス)を使用することもでき、土中の温度を測るとき等に使用します。

写真 7.2.3 温度センサ保護管

●換気扇とリモコンリレーコンセント

温度制御は、換気扇で行います。換気扇の電源をリモコンリレーコンセントから供給して、これをON/OFF 制御します。リモコンリレーコンセントへの制御端子をリモート I/O 装置のデジタル出力ポイント(DO)のリレー出力(オプション装着部品)につなぎます。

写真 7.2.4 換気扇とリモコンリレーコンセント

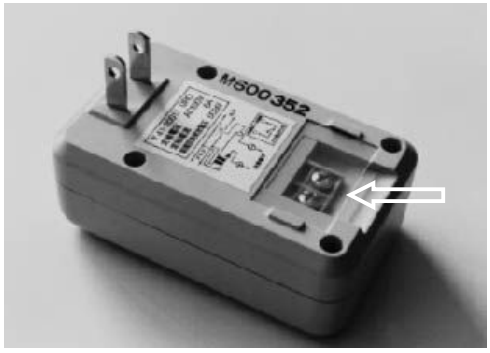


写真 7.2.5 リモコンリレーコンセントの制御端子
(左の2極が 100V へ、右の2ネジがコントロール端子)

●ブザー

ブザーは、DC12V で動作するものを使用しました。

写真 7.2.6 ブザー

◇システム配線

システムの全体配線図を以下に示します。

(Web サーバとインターネット接続に関しては本実用ガイド6章を参照して下さい)

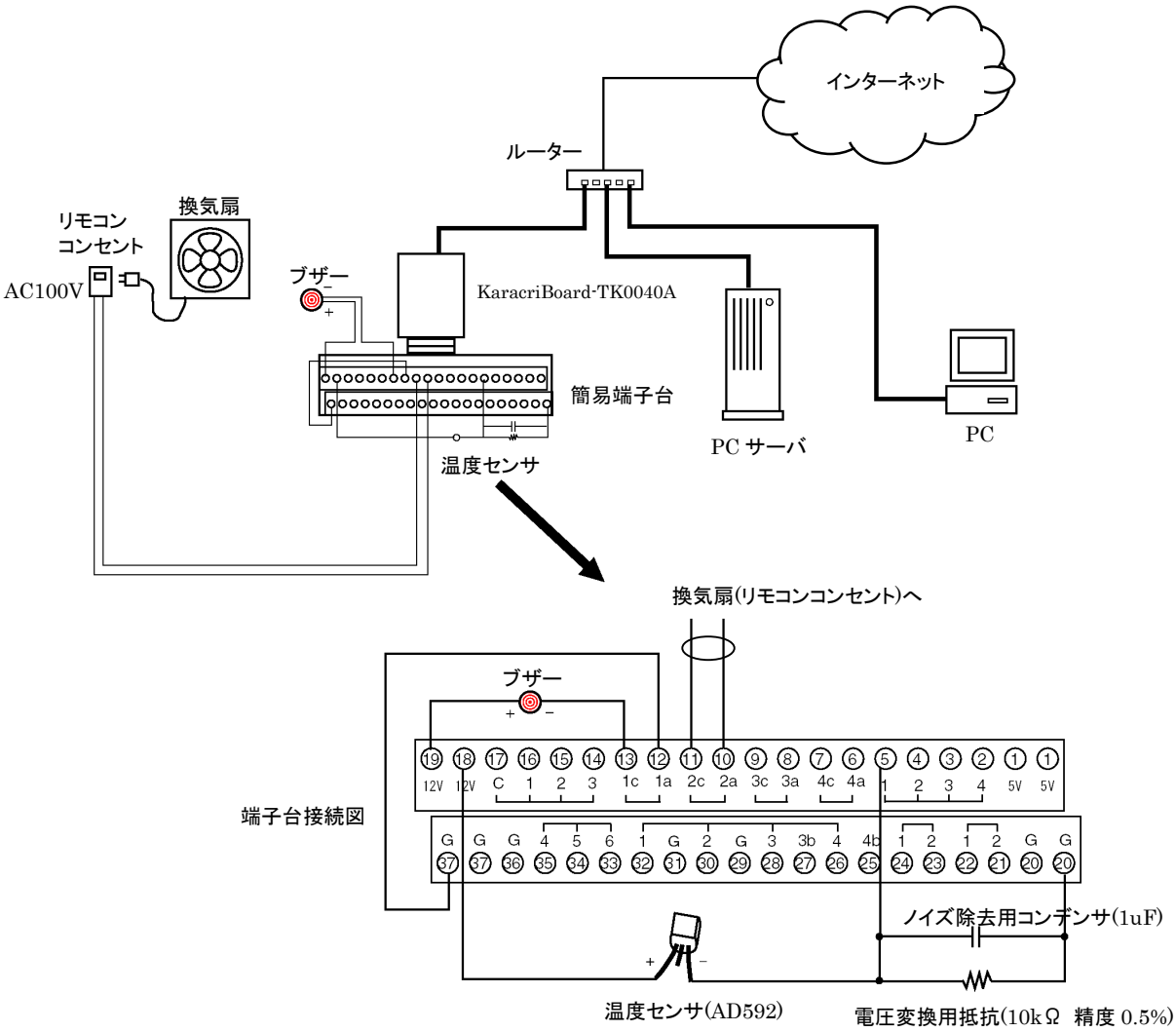


図 7.2.2 システム配線図

写真 7.2.7 横型ルータと HUB

センサ、アクチュエータからの配線は、TK0040A の簡易端子台に接続しています。

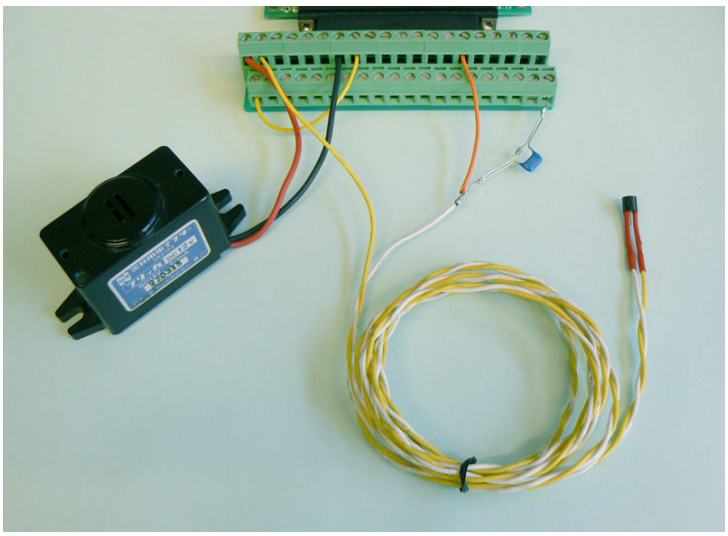


写真 7.2.8 簡易端子台とセンサ、アクチュエータの配線

リモート I/O 装置とPCを HUB を使用して接続している例を写真 7.2.9 に示します。

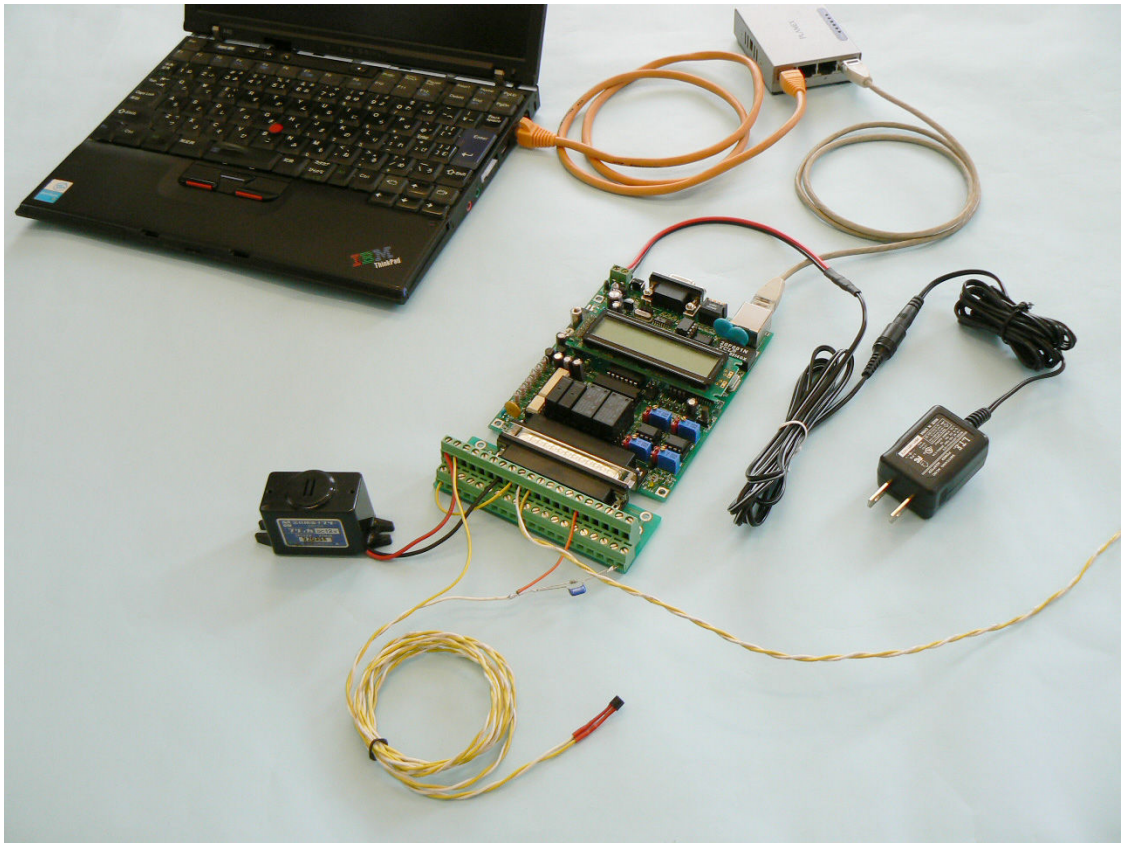


写真 7.2.9 PC との LAN 接続とリモート I/O 装置への配線

7.3　ポイント登録

使用するポイントオブジェクトに、「温度センサ」「警報ブザー」「換気扇」を登録します。
各ポイントに名前をつけておきます。 また、これらのポイントの種別も分類しておきます。

表 7.3.1 ポイント登録一覧

機器	OBJID	ポイント名	ポイント種別
温度センサ	ai001	室内温度	AI (アナログ入力)
警報ブザー	do001	警報ブザー	DO (デジタル出力)
換気扇	do002	換気扇	DO (デジタル出力)

(1)ポイント登録

アナログセンサの「温度センサ」を ai001 に割り当てます。
アクチュエータの「警報ブザー」「換気扇」を do001、do002 に割り当てます。
ポイント名を以下のように変更します。

(AI)

No.	種別	OBJID	タグ名	ポイント名	属性設定	W3	MB
1	AI	ai001	T-ai001	室内温度	(100.00/0.00) (-) (%8.2f)	*	*

(DO)

No.	種別	OBJID	タグ名	ポイント名	属性設定	W3	MB
1	DO	do001	T-do001	警報ブザー	(ON/OFF)	*	*
2	DO	do002	T-do002	換気扇	(ON/OFF)	*	*

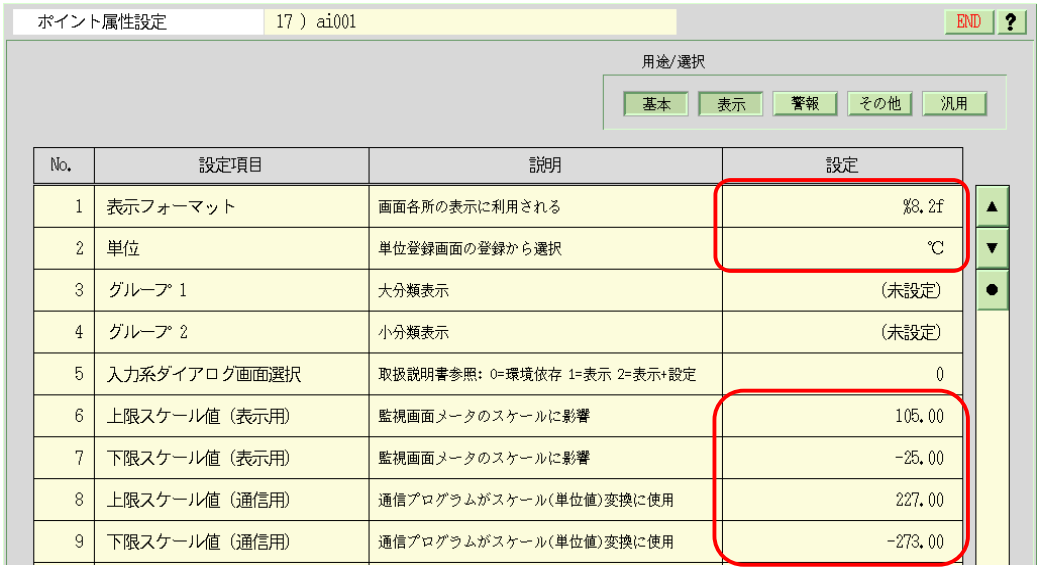
図 7.3.1 ポイント名の変更

(2)ポイント属性設定

●AI ポイント（温度センサ）

本章に使用している温度センサは、電流型温度センサ（AD592）を使用しており、以下の計測仕様となっています。
1.センサ出力　　温度1度当たり1μA の出力 （絶対温度 0 度（K:ケルビン）基準/摂氏にすると-273℃基準）
2.有効計測範囲　-25℃～+105℃ （248K～378K）
3.抵抗電圧出力　2.48V～3.78V（10KΩ 時）　[V=I×R ←(248μA～378KμA)←(248K～378K)←(-25℃～+105℃)]

“表示フォーマット”は、温度センサの精度から表示に必要な桁数を与えます。ここでは、デフォルト値（小数点以下2桁）のまま使用します。“単位”は、温度ですので”℃”を選択します。
上限スケール値（表示用）に 105(℃)、下限スケール値（表示用）に -25(℃)を設定します。
上限スケール値（通信用）に 227(℃)、下限スケール値（通信用）に -273(℃)を設定します。
※センサには温度誤差（個体差）があります。この補正を行なう場合には上下限スケール値（通信用）で調整します。例えばセンサ誤差が+2℃であったとした場合の、上限は 227+2=229、下限は-273+2=-271 と誤差分スケール調整します。



No.	設定項目	説明	設定
1	表示フォーマット	画面各所の表示に利用される	%8.2f
2	単位	単位登録画面の登録から選択	℃
3	グループ 1	大分類表示	(未設定)
4	グループ 2	小分類表示	(未設定)
5	入力系ダイアログ画面選択	取扱説明書参照: 0=環境依存 1=表示 2=表示+設定	0
6	上限スケール値 (表示用)	監視画面メータのスケールに影響	105.00
7	下限スケール値 (表示用)	監視画面メータのスケールに影響	-25.00
8	上限スケール値 (通信用)	通信プログラムがスケール(単位値)変換に使用	227.00
9	下限スケール値 (通信用)	通信プログラムがスケール(単位値)変換に使用	-273.00

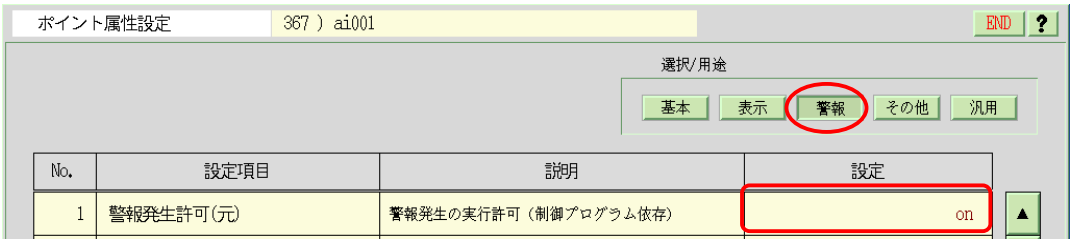
図 7.3.2 AI ポイント(室内温度)の属性設定

(3)警報設定

計測温度が設定上限温度を超えた場合に、警報として扱う為に警報関連の属性設定を行います。「ポイント属性設定」画面の“選択/用途”ボタンで“警報”ボタンのみ選択して下さい。

○警報発生許可(元)

“警報発生許可(元)” 項目が on になっていることを確認して下さい。デフォルト値は on になっていると思います。これは“通信制御ドライバ S1”が使用し、この値が on になっているポイントのみ警報発生処理に使用されます。



No.	設定項目	説明	設定
1	警報発生許可(元)	警報発生の実行許可 (制御プログラム依存)	on

図 7.3.3 “警報発生許可(元)”の属性設定

○警報の機能有効と監視許可

アナログポイントの警報属性は、デジタルポイントの場合とは異なり、警報レベルを上限警報3段階(H1,H2,H3)、下限警報3段階(L1,L2,L3)で設定することができるようになっていますが、警報レベルの使用方法はユーザに任されています。今回は、上限警報が1レベルあればよいとして、H3 で表示される上限警報属性のみを使用します。

“H3(***) 警報機能有効”が on になっていることを確認して下さい。デフォルト値は on になっていると思います。レベル別に警報機能を有効にするときに使用します。(この設定はポイント属性設定(再起動反映)で設定し、ポイント属性一時変更では使用できない初期設定項目です)

同様に、“H3 上限警報監視許可”を on にして下さい。(この設定はポイント属性一時変更でも使用でき、運転中にコンソール画面や Web 画面からアクセスして動的変更が可能となっています)

この2つの設定は、“通信制御ドライバ S1”が使用し、この値が共に on になっているポイントのみ警報発生処理に使用されます。

5	H3(***) 警報機能有効	レベル別警報監視 (制御プログラム依存)	on
6	H3 上限警報監視許可	現警報監視許可 (警報機能有効時プログラム依存)	on

図 7.3.4 “警報機能有効”、“警報監視許可”の属性設定

○警報値

“H3 警報値 (表示用)”に設定する値は、上限温度として警報を発生させたい温度のタイトルとしての表示値を設定します。ここでは、35℃に設定します。

“H3 警報値 (不感帯上値=a)”、“H3 警報値 (不感帯下値=b:a>b)”は、実際に警報判断に使用される値です。この2値によって幅を持たせているのは警報発生温度境界付近での計測温度の小刻みな変化によって生ずる警報発生解除のバタツキを抑えるために必要なものです。この幅の事を不感帯ともヒステリシス帯とも言います。ここでは、上限警報用なので表示用と不感帯上値を同じにし、1.0℃の不感帯値を設けることにして、上値 35.00、下値 34.00 に設定しています。

※不感帯上値(a)は、不感帯下値(b)よりも大きいな値(a>b)でなければなりません。

7	H3 警報値 (不感帯上値=a)	警報判断用	35.00
8	H3 警報値 (表示用)	画面表示に使用	35.00
9	H3 警報値 (不感帯下値=b:a>b)	警報判断用	34.00

図 7.3.5 “警報値”の属性設定

以下の図に、温度センサの計測値が上限警報値 35℃を超えたときの表示の例を示します。警報時に、警報発生時のカラーセット(事象別の塗色)が使用されて表示されます。(下記解説色で示す色はデフォルトカラーセットを使用した場合のものです)

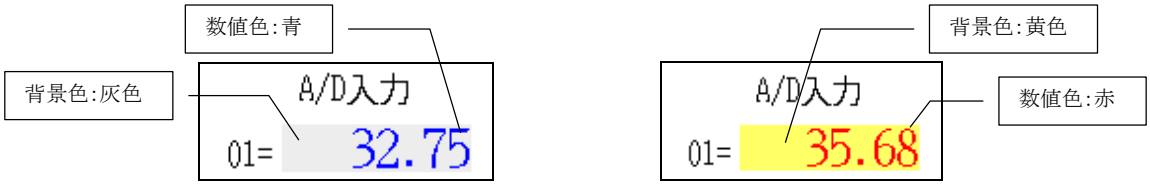


図 7.3.6「温度センサ」の計測値の監視パネル表示例(左:通常時 右:警報発生時)

○警報メール許可

本設定は、"通信制御ドライバ S1"が使用し、この値が on になっているポイントの警報メール送信処理に使用されますが、本章では、ユーザプログラム自身で警報メール送信処理を行なっているため、通信制御ドライバ S1 のメール発信機能は本章では使用しません。 従って、設定は off にしておいて下さい。

7	警報メール許可	制御プログラム依存	off

図 7.3.7 警報メールの許可設定

●DO ポイント(警報ブザー、換気扇)

DO ポイントの属性では、"状態文字"の属性設定を行います。
ON/OFF の状態を表す文字列は、デフォルト値(ON/OFF)で支障ありませんので、そのまま使用しています。

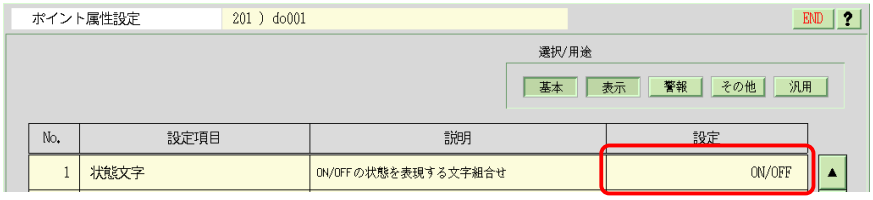


図 7.3.8 DO ポイントの属性設定

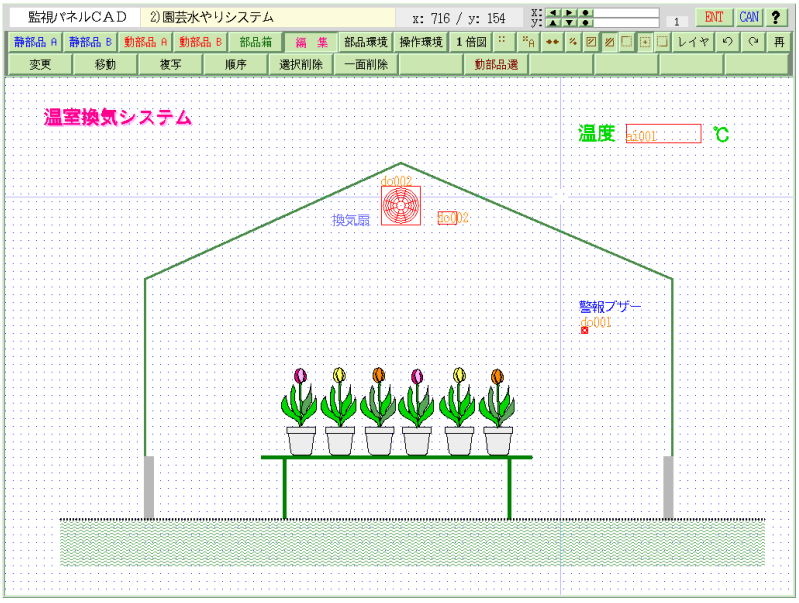
以上で、ポイントの登録は終了です。

ここで、今までの設定をシステムに保存反映させる場合には、「ポイント登録」画面で"END"ボタンを選択して「メインメニュー」へ戻り、KaracrixBuilder コンソールの "RST"ボタンを選択して KaracrixBuilder をリセットして下さい。しばらくすると KaracrixBuilder が自動的に再起動されます。

7.4　監視パネルの作成

監視パネルをつぎのように作成しました。
監視パネルの作成手順については本実用ガイド4、5章を参考にして下さい。

（監視パネル CAD）



（監視パネル実行画面表示）

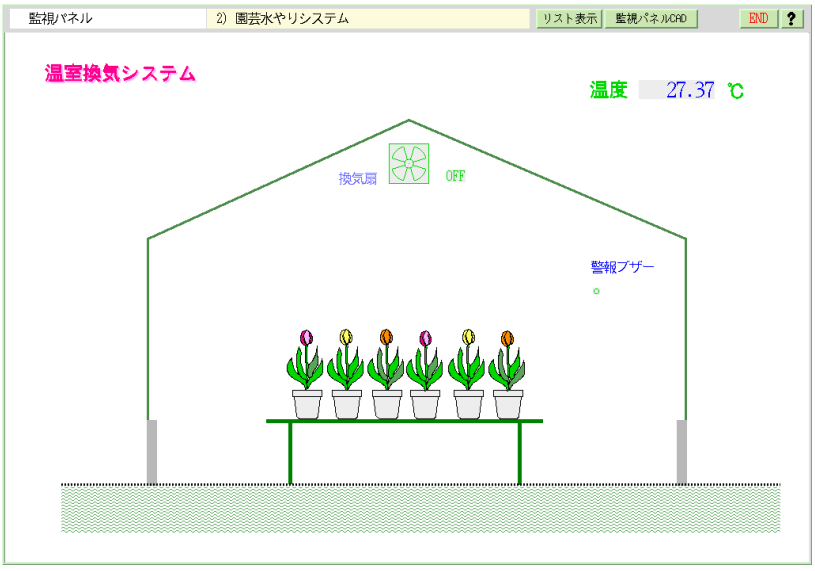


図 7.4.1 温室換気システム監視パネル(上:CAD 画面　下:監視パネル)

7.5　監視制御プログラムの作成

本システムでは、温室の温度を計測して設定温度によって換気扇を ON/OFF 制御する「温度制御」、設定上限温度を超えている場合に警報ブザーを鳴らしかつEメールを送信する「温度上限警報」、そして、毎日決められた時刻に指定のアドレスに監視パネル画像とトレンドグラフ画像を添付したを E メールを送信する「定時 E メール送信」の3つの処理を構成してみたいと思います。
3つの機能を盛り込んでいますので、1本のプログラムで構成すると複雑になることが予想されます。そこで今回は、機能ごとにタスク(プロセス)を分けて構成する手法を取りました。機能別にタスクを独立させることにより各プログラムがシンプルになり、システム全体の見通しも良くなります。また、プログラムの修正やデバックも機能別に行うことができるようなるため容易になります。

表 7.5.1 プログラム一覧

タスク番号	OBJID	プログラム概要
タスク1	ctl02	複数プログラム起動プログラム
タスク2	ctl03	温度制御プログラム
タスク3	ctl04	温度上限警報プログラム
タスク4	ctl05	定時 E メール送信プログラム
タスク5	ctl10	通信制御ドライバ S1 （ダウンロード＆インストール利用）

複数プログラム起動プログラム(タスク1)は、機能毎に独立して同時並行して動作するタスク2～5を順次起動させる親タスクとして機能します。但しこのプログラムは、起動の役目を終えると実行終了してしまいます。
図 7.5.1 に全体フローを、図 7.5.2、7.5.3、7.5.4 に各処理のフローを示します。

図 7.5.1 温室制御システム起動処理フロー図

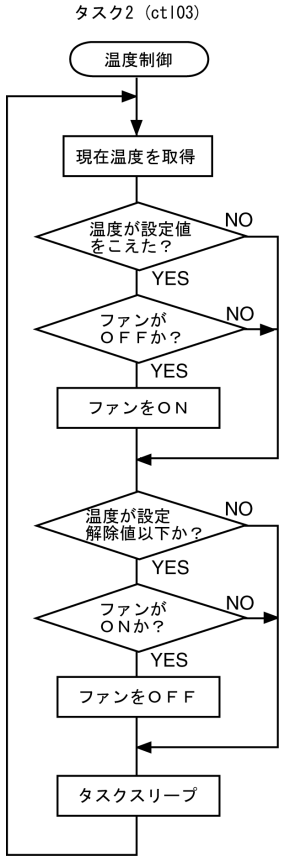


図 7.5.2 温度制御処理フロー図

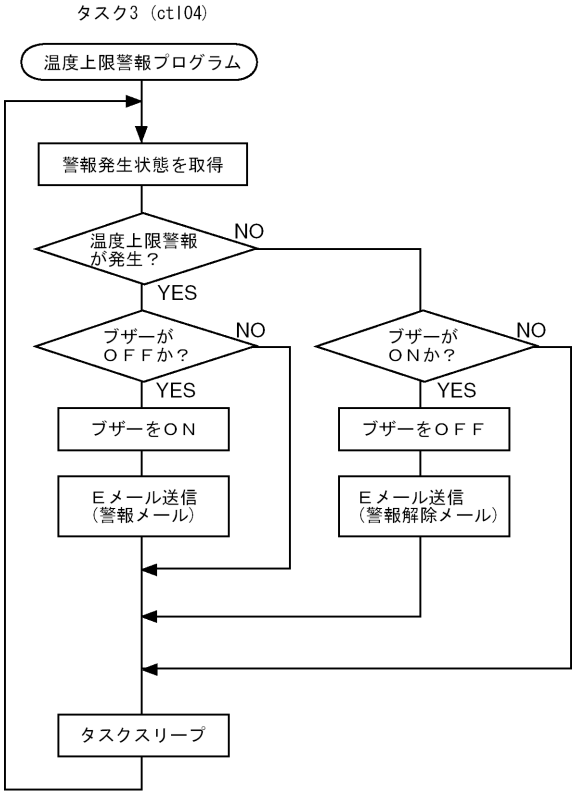


図 7.5.3 温度上限警報処理フロー図

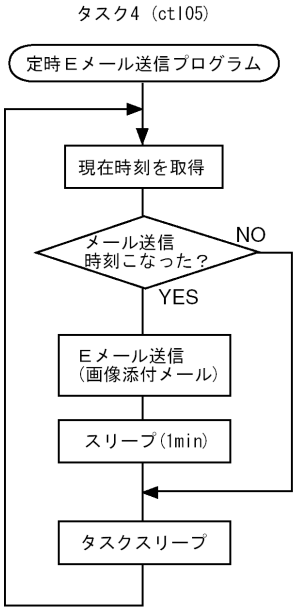


図 7.5.4 定時 E メール送信処理フロー図

それでは、処理フロー図をもとにプログラミングしてみましょう。「メインメニュー」から“制御プログラム”ボタンを選択して「制御プログラム登録」画面を表示して下さい。

タスク1の「複数プログラム起動プログラム」の登録は、No.2 の ctl02 オブジェクトに、
タスク2の「温度制御プログラム」の登録は、No.3 の ctl03 オブジェクトに、
タスク3の「温度上限警報プログラム」の登録は、No.4 の ctl04 オブジェクトに、
タスク4の「定時Eメール送信プログラム」の登録は、No.5 の ctl05 オブジェクトに、
それぞれ登録してください。
タスク5の「通信制御ドライバ S1」は、KARACRIX のホームページからインポートするなどして設定しておきます。なお、TK0040A とは予め通信試験を行なうなどして動作を確認しておいてください。

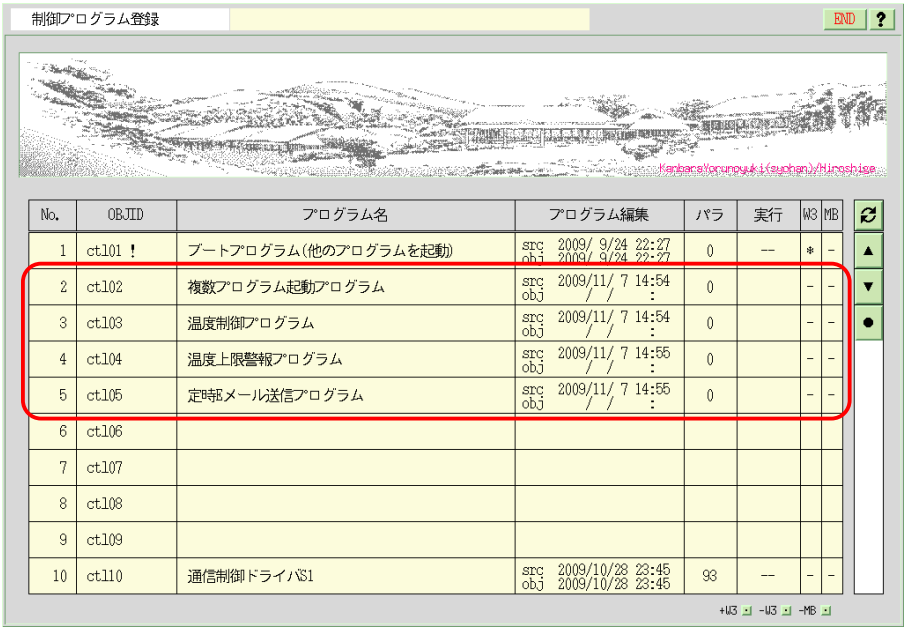


図 7.5.5 制御プログラム登録

以下で、各タスクの主要部分についてアルゴリズムの説明をしていきます。

●複数プログラム起動プログラム [タスク1(objid=ctl02)]

14-17 行目で、温度制御(ctl03)、温度上限警報(ctl04)、定時 E メール送信(ctl05)、通信制御ドライバ S1(ctl10) の各プログラムオブジェクトIDを取得しています。
20-26 行目で、これらプログラムを起動しています。また、起動されたプログラムは、本起動プログラムとは無関係に独立して同時並列稼動(マルチタスク処理)していきます。
30 行で本プログラムは終了し役割を終えます。

```
13      /* プログラムオブジェクト ID の取得 */
14      ondo_ctl_prg_objid  =  kcxobj_open( "ctl03" );
15      ondo_alm_prg_objid  =  kcxobj_open( "ctl04" );
16      email_ctl_prg_objid =  kcxobj_open( "ctl05" );
17      S1_prg_objid        =  kcxobj_open( "ctl10" );
18
19      /* コンパイル済のプログラム(オブジェクト)の起動 / (起動順番と時間は適宜設定)     */
20      kcxobj_stat_iwt( S1_prg_objid, 1 );   /* 通信制御ドライバ S1 の起動                */
21      sleep( 60 );                          /* 通信確立まで暫く待機(約 60 秒～300 秒程) */
22      kcxobj_stat_iwt( ondo_ctl_prg_objid,  1 );
23      sleep( 1 );
24      kcxobj_stat_iwt( ondo_alm_prg_objid,  1 );
25      sleep( 1 );
26      kcxobj_stat_iwt( email_ctl_prg_objid, 1 );
27
28      /* 上記プログラムを起動したら本プログラム終了 */
29
30 }
```

● 温度制御プログラム [タスク2(objid=ctl03)]

20-22 行目では、初期設定としてファンを停止させています。

```
20      fan_objid         =  kcxobj_open( "do001" );
21      fan_status        =  OFF;
22      kcxobj_sndistat_tokcx( fan_objid, fan_status );  /*具体的操作は S1 が実行*/
```

● 温度上限警報プログラム [タスク3(objid=ctl04)]

48-57 行目では、ブザーをONにするタイミングで、温度異常のメールを送信しています。

```
48          if( buzzer_status == ON ){
49            /* 1.ブザーが起動中だったら停止操作(OFF 値を送信キューに登録) */
50            kcxobj_sndistat_tokcx( buzzer_objid, ( buzzer_status = OFF ) );
51            /* 2.同時に警報解除の E メールを送信する                          */
52            /* 送信アドレス例として、tarou@yyy.xx.jp 題名は、温度異常解除   */
53            /* 電文(almtext)には現在温度を付加するとしました                 */
54            kcxobj_stat_frd( temperature_objid, &temperature_val ); /*温度取得*/
55            sprintf( almtext,"現在温度= %5.2f ° C ですよ", temperature_val );
56            kcxsnd_email_text( "tarou@yyy.xx.jp","","","温度異常解除",almtext );
57          }
```

● 定時 E メール送信プログラム　[タスク4(objid=ctl05)]

37 行目は、記述するメール本文の行数を設定しています。例えば 30 行書きたい場合には、mtext = 30; となります。本プログラム例では、(2)と記述してありますが、この( )は 2 を強調したいために用いただけで他意はありません。
39 行目の malloc 関数で、引数で示された半角文字数の長さ分のメモリ領域を確保しています。これを、mtext[i]に割り当てて使用します。
41 行目は、メールに添付したいコマンド(ファイル)分の数を設定します。

```
35      /* メール送信関数に使うバッファメモリの取得(mtexts=本文,aptexts=添付分)        */
36      /* ※送信したい文字列の行数だけ malloc()させるもので KCX ライブラリ関数仕様です */
37      mtexts  = (2);
38      for( i = 0;i < mtexts; i++ ){
39        mtext[i]  = (char *)malloc( 256 );
40      }
41      aptexts = (2);
42      for( i = 0;i < aptexts; i++ ){
43        aptext[i] = (char *)malloc( 96 );
44      }
```

67-83 行目で、メール送信します。

```
67        if( email_send_flag == ON ){
68
69          /* 送信アドレス例として、tarou@yyy.xx.jp 題名は、定時報告   */
70          /* 電文(almtext)には現在時間と送信回数を付加するとしました   */
71          sprintf( mtext[0], "現在時刻= %d 日 %d 時 %d 分 %d 秒",
72                                jikan.tm_mday,jikan.tm_hour,jikan.tm_min,jikan.tm_sec);
73          sprintf( mtext[1],"メール回数は %d 回目ですよ", ++repcount );
74          strcpy( aptext[0], "mon  1 " ); /*No.1 の監視パネル画像添付指示      */
75          strcpy( aptext[1], "mtre 1 " ); /*No.1 のトレンドグラフ画像添付指示 */
76
77          kcxsnd_email_texts_appends( "tarou@yyy.xx.jp","","","定時報告",
78                                      mtext, mtexts, aptext, aptexts );
79
80          /* E メールを同じ時間内に連続送信しない為の簡単な遅延(sleep)処置 */
81          sleep( 60 );  /*1 分*/
82
83        }
```

7.6　監視制御プログラムの実行

登録した全てのプログラムをコンパイルして、実行ファイルが作成できたら実行してみます。「制御プログラム登録」画面に戻って下さい。

●プログラムコンパイルの確認

使用する全てのプログラムをコンパイルして、実行ファイル(obj)が出来ているかを確認します。

●プログラムを実行する

No.2 の ctl02 オブジェクト「複数プログラム起動プログラム」の“実行”欄をクリックすると、「実行ダイアログ」が表示されますので“RUN”ボタンを選択して実行します。

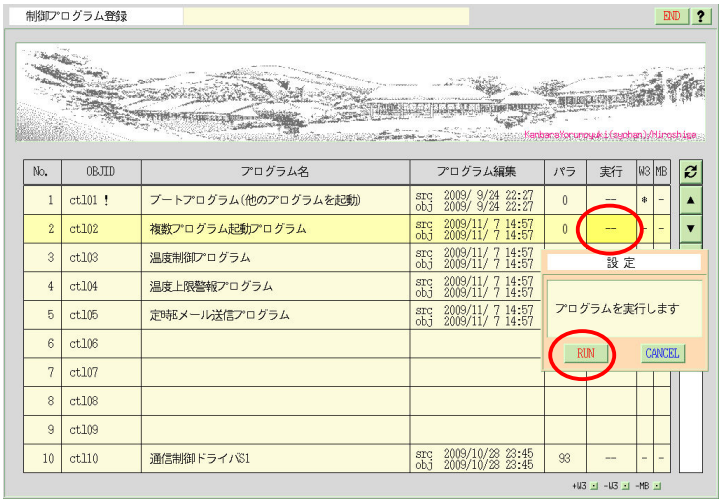


図 7.6.1　起動プログラムの実行

ctl02 の実行後、ctl10、ctl03、ctl04、ctl05 のプログラムが次々と自動起動されればタスクの実行は成功です。後は、各プログラムの処理が正常かを確認してください。

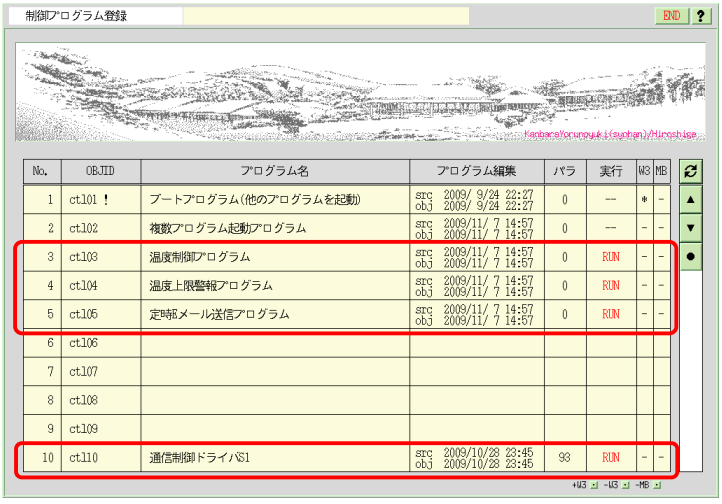


図 7.6.2　ユーザープログラムの自動起動

7.7　警報メールの受信

温度上限警報のメール送信プログラムが動作すると、プログラム内で記述し指定(ハードコーディング)したメールアドレスに、メッセージを添付して E メールを送信します。
以下に温度上限警報プログラムからの E メールの受信画面の例を示します。

(温度異常警報発生)

図 7.7.1 温度上限警報 E メールよる受信画面

7.8　E メールによる定時監視パネル画像の受信

定時 E メール送信プログラムが動作すると、プログラム内で記述し指定(ハードコーディング)したメールアドレスに、指定した監視パネル画像とトレンドグラフ画像を添付して E メールを送信します。
以下に定時 E メールの受信画面の例を示します。

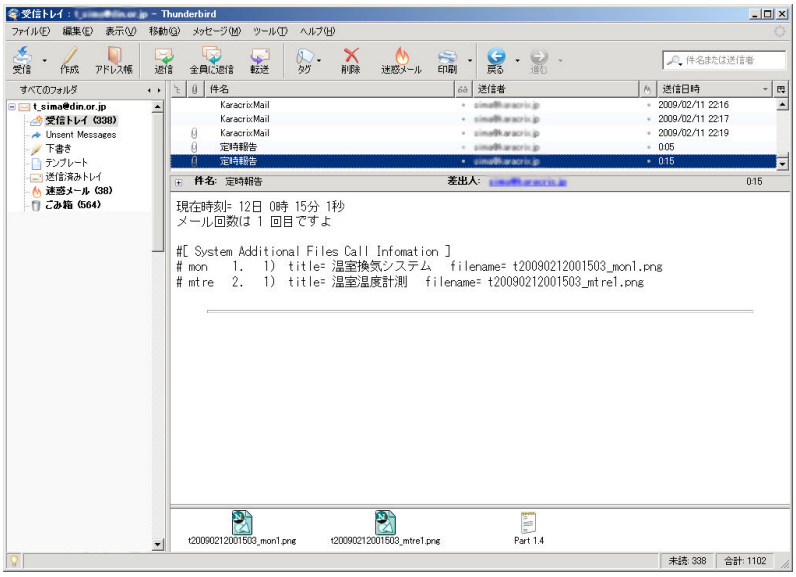


図 7.8.1　定時 E メールよる受信画面(本文)

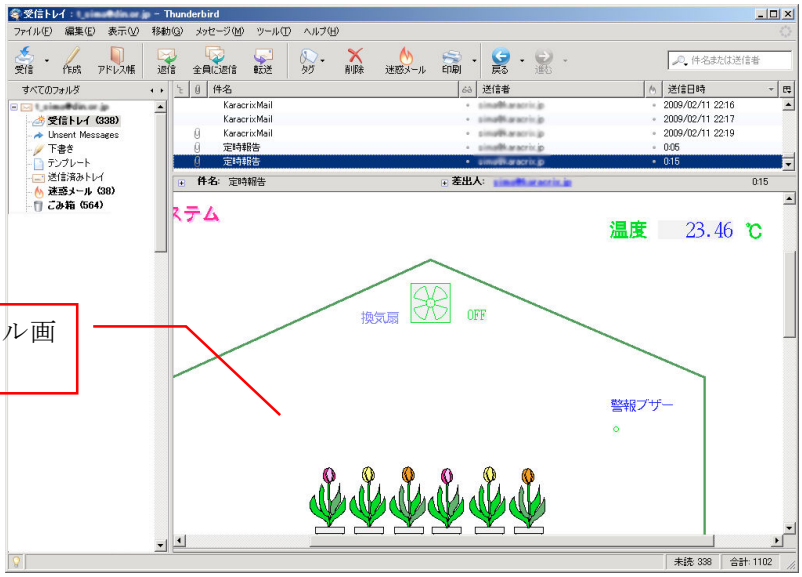


図 7.8.2　定時 E メールよる受信画面(監視パネル添付画像)

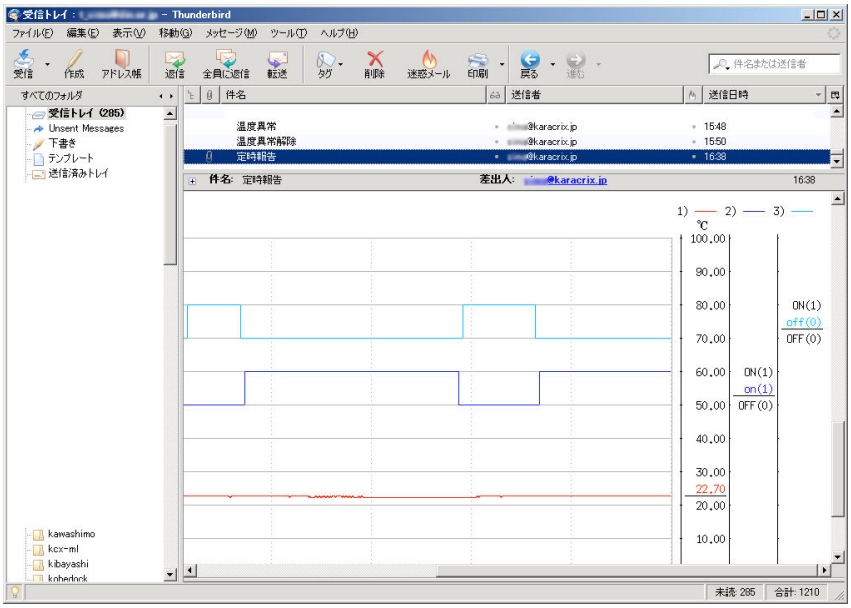


図 7.8.3 定時 E メールよる受信画面(トレンドグラフ添付画像)

7.9　操作履歴と警報履歴

KaracrixBuilder には、操作履歴と警報履歴というものがあります。これは、ポイントがどの様に操作(あるいは状態変化)されたのか、また警報に関しては、どの様なタイミングで何が発生したのかなど、システムの運転状況を時系列的に知るためにあります。この履歴を参照することによってシステムが設計通りに動作しているか、また、正常運転しているかなどを確かめることが出来ます。履歴情報はシステム(プログラム)を改善する上でも重要なものになります。
通信制御ドライバ S1 を稼動させて情報を履歴に残す場合には以下の設定が必要となります。

○DO ポイント(操作履歴を保存する設定)

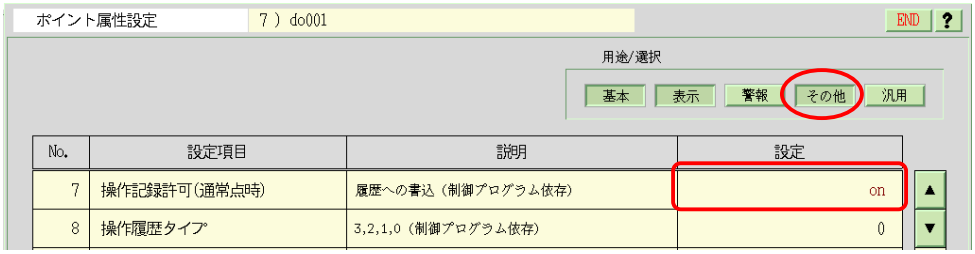


図 7.9.1 “操作記録許可(通常点時)”の属性設定

○AI ポイント(警報履歴を保存する設定)

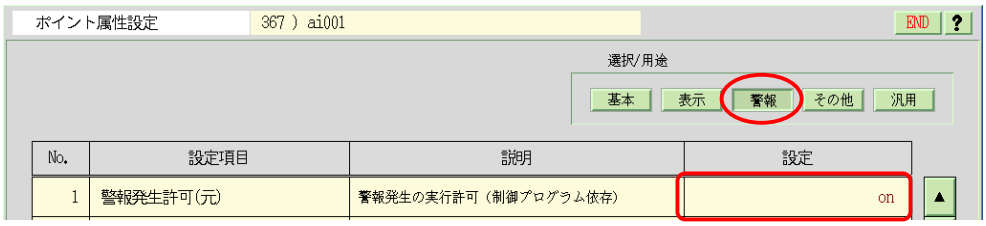


図 7.9.2 “警報発生許可(元)”の属性設定

履歴は、KaracrixBuilder システム内部からの情報も書き込まれます。
例えば、システムの起動時間やメールサーバとの通信エラーなどに使われています。
次頁に、本システムを動作させたときの操作履歴、警報履歴の例を示します。

図 7.9.3 操作履歴の表示例

図 7.9.4 警報履歴の表示例

7.10 計測記録とトレンドグラフ

KaracrixBuilder には、操作警報履歴の記録とは別に、計測されているデータを定期的にハードディスクに記録し残す機能があります。そしてこの設定を行うことによって、長時間軸の記録トレンドグラフを見ることができるようになります。また記録されたデータは、他の OS(ウインドウズ等)でも使用できるように一般的な CSV ファイルとして取得できます。このファイルは、Web ブラウザを用い遠隔よりダウンロードすることができます。計測したデータを元に自動制御プログラムの改善に役立てる重要なデータともなり得ますので記録して見てはいかがでしょう。

製品版の KaracrixBuilder-500B では、この記録データを使用して日月報告書(Web ブラウザより PDF ダウンロード可)を作成することができます。

計測記録設定は「システム環境設定メニュー」→「計測記録設定」ボタンの選択で行ないます。

図 7.10.1 計測記録の設定例

次頁に、本システムを動作させたときの記録トレンドグラフと帳票の表示例を示します。

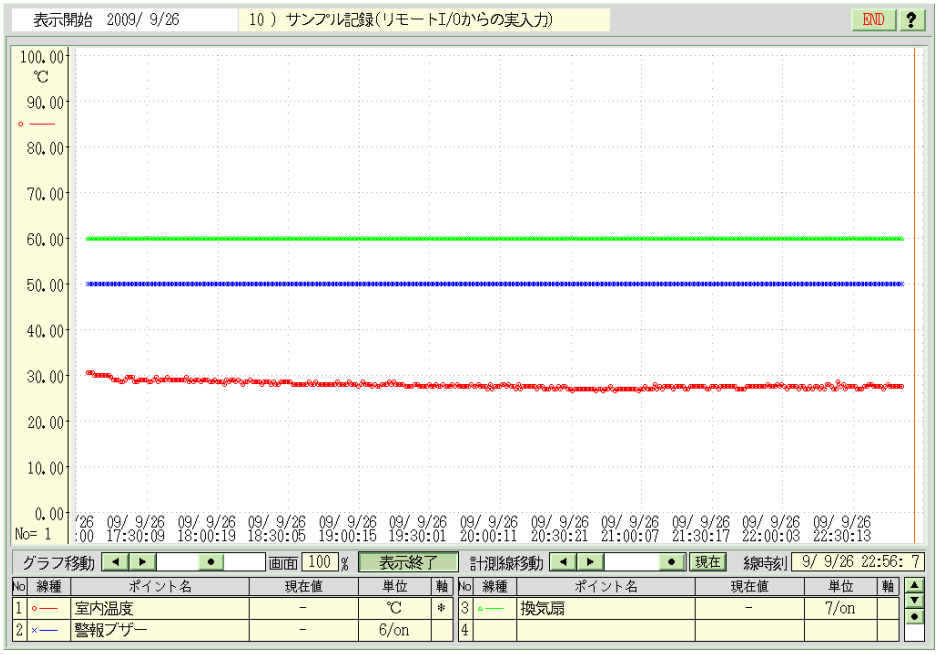


図 7.10.2 記録トレンドグラフの表示例

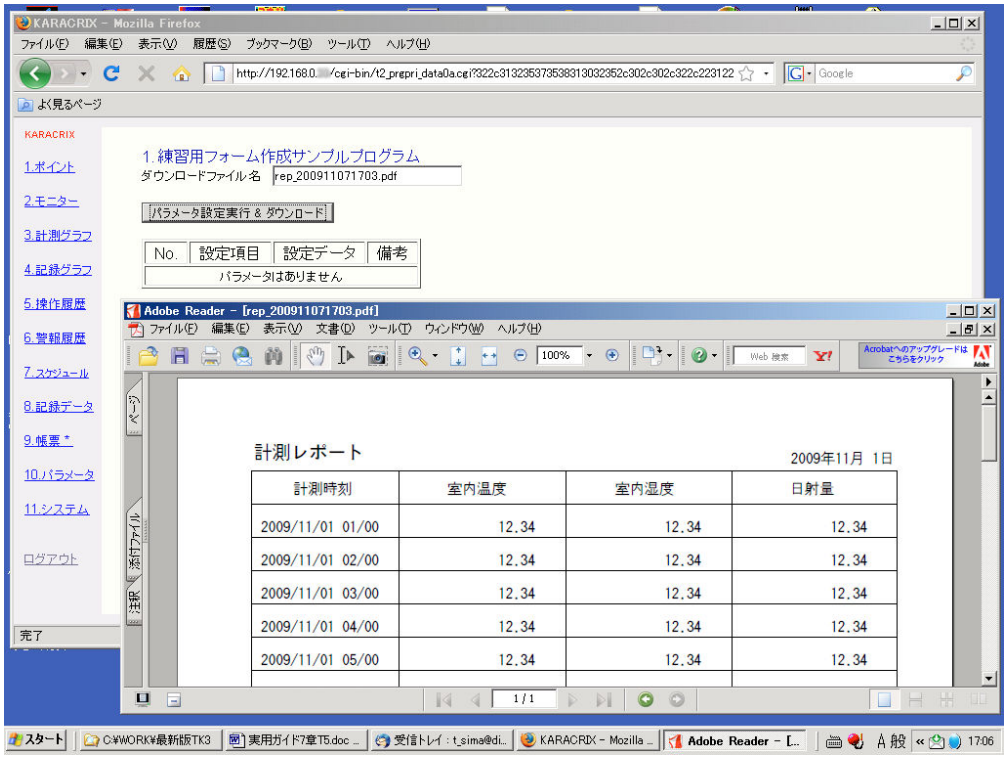


図 7.10.3 帳票(日報データ)のダウンロード表示例

7.11 プログラムリスト

[タスク１(objid=ctl02)] 複数プログラム(タスク)起動用プログラム

```
 1 #include <karacrix.h>
 2
 3 main( int argc, char *argv[] )
 4 {
 5   int  ondo_ctl_prg_objid;   /* 温度制御プログラムのオブジェクト ID           */
 6   int  ondo_alm_prg_objid;   /* 温度警報プログラムのオブジェクト ID           */
 7   int  email_ctl_prg_objid;  /* 定時 E メール送信プログラムのオブジェクト ID    */
 8   int  S1_prg_objid;         /* 通信制御ドライバ S1 プログラムのオブジェクト ID */
 9
10     /* KARACRIX ライブラリの初期化(先頭に必須) */
11     kcxinit( argc, argv );
12
13     /* プログラムオブジェクト ID の取得 */
14     ondo_ctl_prg_objid  =  kcxobj_open( "ctl03" );
15     ondo_alm_prg_objid  =  kcxobj_open( "ctl04" );
16     email_ctl_prg_objid =  kcxobj_open( "ctl05" );
17     S1_prg_objid        =  kcxobj_open( "ctl10" );
18
19     /* コンパイル済のプログラム(オブジェクト)の起動 / (起動順番と時間は適宜設定)   */
20     kcxobj_stat_iwt( S1_prg_objid, 1 );   /* 通信制御ドライバ S1 の起動              */
21     sleep( 60 );                          /* 通信確立まで暫く待機(約 60 秒～300 秒程) */
22     kcxobj_stat_iwt( ondo_ctl_prg_objid,  1 );
23     sleep( 1 );
24     kcxobj_stat_iwt( ondo_alm_prg_objid,  1 );
25     sleep( 1 );
26     kcxobj_stat_iwt( email_ctl_prg_objid, 1 );
27
28     /* 上記プログラムを起動したら本プログラム終了 */
29
30 }
```

[タスク２(objid=ctl03)]　温度制御プログラム

```
 1 #include <karacrix.h>
 2
 3 #define  ON     (1)    /* DO 起動用 */
 4 #define  OFF    (0)    /* DO 停止用 */
 5
 6 main( int argc, char *argv[] )
 7 {
 8   int    temperature_objid;        /* 温度センサーのポイントオブジェクト ID */
 9   double temperature_currentval;   /* 温度センサーの状態値を格納            */
10   double temperature_funon_val;    /* 警戒温度設定値                        */
11   double temperature_funoff_val;   /* 警戒温度設定解除値                    */
12   int    fan_objid;                /* ファンのポイントオブジェクト ID       */
13   int    fan_status;               /* ファンの状態を格納                    */
14
15     /* KARACRIX ライブラリの初期化(先頭に必須) */
16     kcxinit( argc, argv );
17
18     /* ポイントオブジェクト ID の取得とファン初期停止の操作 */
19     temperature_objid =  kcxobj_open( "ai001" );
20     fan_objid         =  kcxobj_open( "do001" );
21     fan_status        =  OFF;
22     kcxobj_sndistat_tokcx( fan_objid, fan_status );  /*具体的操作は S1 が実行*/
23
24     /* 温度制御パラメータの設定 */
25     temperature_funon_val   =  35.0;  /* ファン ON 温度[℃]  */
26     temperature_funoff_val  =  30.0;  /* ファン OFF 温度[℃] */
27     /* 以下の関数を使用して、動的変更可能な属性にパラメータを入れておくのも良い
28     kcxobj_atbut_frd( temperature_objid, 1, &temperature_funoff_val );
29     kcxobj_atbut_frd( temperature_objid, 2, &temperature_funon_val  );
30     */
31
32     while( 1 ){  /* 監視制御ループ */
33
34       /* 現在の温度を取得 */
35       kcxobj_stat_frd( temperature_objid, &temperature_currentval );
36
37       /* 警戒設定温度を越えたら --> ファン起動 */
38       if( temperature_currentval > temperature_funon_val ){
39         if( fan_status == OFF ){
40           /* ファンが停止中だったら起動操作(操作値を送信キューに登録) */
41           fan_status = ON;
42           kcxobj_sndistat_tokcx( fan_objid, fan_status );
43         }
44       }
45
46       /* 警戒設定解除温度より下がったら --> ファン停止 */
47       if( temperature_currentval < temperature_funoff_val ){
48         if( fan_status == ON ){
49           /* ファンが起動中だったら停止操作(操作値を送信キューに登録) */
50           fan_status = OFF;
51           kcxobj_sndistat_tokcx( fan_objid, fan_status );
52         }
53       }
54
55       /* 必須:CPU を３秒停止させる(負荷を和らげる為) */
56       sleep( 3 );
57
58     }/*while*/
59
60 }
```

[タスク３(objid=ctl04)] 温度上限警報プログラム

```
 1 #include <karacrix.h>
 2
 3 #define  ON     (1)    /* DO起動用 */
 4 #define  OFF    (0)    /* DO停止用 */
 5
 6 main( int argc, char *argv[] )
 7 {
 8   int    temperature_objid;  /* 温度センサーのポイントオブジェクトID */
 9   double temperature_val;    /* 温度センサーの状態値を格納          */
10   int    temperature_alarm_status;  /* 温度センサーの警報ステータス  */
11   int    buzzer_objid;       /* ブザーのポイントオブジェクトID       */
12   int    buzzer_status;      /* ブザーの状態を格納                  */
13   char   almtext[256];       /* Eメール電文                         */
14
15     /* KARACRIXライブラリの初期化(先頭に必須) */
16     kcxinit( argc, argv );
17
18     /* ポイントオブジェクトIDの取得とブザー初期停止の操作 */
19     temperature_objid =  kcxobj_open( "ai001" );
20     buzzer_objid      =  kcxobj_open( "do002" );
21     buzzer_status     =  OFF;
22     kcxobj_sndistat_tokcx( buzzer_objid, buzzer_status );
23
24     while( 1 ){  /* 監視制御ループ */
25
26       /* 通信制御ドライバS1の警報機能が検出した温度の警報発生状態を取得する */
27       kcxobj_alm_stat_ird( temperature_objid, &temperature_alarm_status );
28
29       if( temperature_alarm_status >= 1 ){
30
31         /* 警報発生状態 */
32
33         if( buzzer_status == OFF ){
34           /* 1.ブザーが停止中だったら起動操作(ON値を送信キューに登録) */
35           kcxobj_sndistat_tokcx( buzzer_objid, ( buzzer_status = ON ) );
36           /* 2.同時に警報のEメールを送信する                      */
37           /* 送信アドレス例として、tarou@yyy.xx.jp 題名は、温度異常  */
38           /* 電文(almtext)には現在温度を付加するとしました        */
39           kcxobj_stat_frd( temperature_objid, &temperature_val ); /*温度取得*/
40           sprintf( almtext,"現在温度= %5.2f ° C ですよ", temperature_val );
41           kcxsnd_email_text( "tarou@yyy.xx.jp","","","温度異常",almtext );
42         }
43
44       }else{
45
46         /* 警報停止状態 */
47
48         if( buzzer_status == ON ){
49           /* 1.ブザーが起動中だったら停止操作(OFF値を送信キューに登録) */
50           kcxobj_sndistat_tokcx( buzzer_objid, ( buzzer_status = OFF ) );
51           /* 2.同時に警報解除のEメールを送信する                  */
52           /* 送信アドレス例として、tarou@yyy.xx.jp 題名は、温度異常解除  */
53           /* 電文(almtext)には現在温度を付加するとしました        */
54           kcxobj_stat_frd( temperature_objid, &temperature_val ); /*温度取得*/
55           sprintf( almtext,"現在温度= %5.2f ° C ですよ", temperature_val );
56           kcxsnd_email_text( "tarou@yyy.xx.jp","","","温度異常解除",almtext );
57         }
58
59       }
60
61       /* 必須:CPUを３秒停止させる(負荷を和らげる為) */
62       sleep( 3 );
63
64     }/*while*/
65
66 }
```

[タスク４(objid=ctl05)]　定時Ｅメール送信プログラム

```
 1 #include <karacrix.h>
 2
 3 #define  ON     (1)    /* DO起動用 */
 4 #define  OFF    (0)    /* DO停止用 */
 5
 6 main( int argc, char *argv[] )
 7 {
 8   struct tm jikan;               /* 時間データを格納する構造体  */
 9   int    email_send_flag;        /* Ｅメール送信フラグ          */
10   int    email_start_hour_1;     /* １回目のＥメール送信時刻(時) */
11   int    email_start_minute_1;   /* １回目のＥメール送信時刻(分) */
12   int    email_start_hour_2;     /* ２回目のＥメール送信時刻(時) */
13   int    email_start_minute_2;   /* ２回目のＥメール送信時刻(分) */
14   int    email_start_hour_3;     /* ３回目のＥメール送信時刻(時) */
15   int    email_start_minute_3;   /* ３回目のＥメール送信時刻(分) */
16   int    i,repcount;             /* Ｅメール送信回数カウンタ     */
17   char  *mtext [400];            /* Ｅメール本文バッファポインタ */
18   char  *aptext[4];              /* Ｅメール添付バッファポインタ */
19   int    mtexts,aptexts;         /* 本文添付バッファ数           */
20
21     /* KARACRIXライブラリの初期化(先頭に必須) */
22     kcxinit( argc, argv );
23
24     /* 制御パラメータの設定 */
25     email_start_hour_1    =   8;   /*  8時  時刻[時]:0から23まで  */
26     email_start_minute_1  =   0;   /* 30分  時刻[分]:0から59まで */
27     email_start_hour_2    =  12;   /* 12時  時刻[時]:0から23まで */
28     email_start_minute_2  =  30;   /* 30分  時刻[分]:0から59まで */
29     email_start_hour_3    =  17;   /* 17時  時刻[時]:0から23まで */
30     email_start_minute_3  =   0;   /*  0分  時刻[分]:0から59まで */
31
32     /* メール送信回数をカウントするメモリをＯクリアする */
33     repcount = 0;
34
35     /* メール送信関数に使うバッファメモリの取得(mtexts=本文,aptexts=添付分)       */
36     /* ※送信したい文字列の行数だけmalloc()させるものでKCXライブラリ関数仕様です */
37     mtexts  = (2);
38     for( i = 0;i < mtexts; i++ ){
39       mtext[i]  = (char *)malloc( 256 );
40     }
41     aptexts = (2);
42     for( i = 0;i < aptexts; i++ ){
43       aptext[i] = (char *)malloc( 96 );
44     }
45
46     while( 1 ){  /* 監視制御ループ */
47
48       /* 現在の年月日時分秒を取得 */
49       kcxtim_whattime( &jikan );
50
51       /* Ｅメール送信時刻の検査 */
52       email_send_flag =  OFF;
53       if(( jikan.tm_hour == email_start_hour_1   ) &&
54          ( jikan.tm_min  == email_start_minute_1 ) ){
55         email_send_flag =  ON;
56       }
57       if(( jikan.tm_hour == email_start_hour_2   ) &&
58          ( jikan.tm_min  == email_start_minute_2 ) ){
59         email_send_flag =  ON;
60       }
```

```
61        if(( jikan.tm_hour == email_start_hour_3   ) &&
62           ( jikan.tm_min  == email_start_minute_3 ) ){
63          email_send_flag =  ON;
64        }
65
66        /* E メールの送信 */
67        if( email_send_flag == ON ){
68
69          /* 送信アドレス例として、tarou@yyy.xx.jp 題名は、定時報告  */
70          /* 電文(almtext)には現在時間と送信回数を付加するとしました  */
71          sprintf( mtext[0], "現在時刻= %d 日 %d 時 %d 分 %d 秒",
72                             jikan.tm_mday,jikan.tm_hour,jikan.tm_min,jikan.tm_sec);
73          sprintf( mtext[1],"メール回数は %d 回目ですよ", ++repcount );
74          strcpy( aptext[0], "mon  1 " ); /*No.1 の監視パネル画像添付指示      */
75          strcpy( aptext[1], "mtre 1 " ); /*No.1 のトレンドグラフ画像添付指示 */
76
77          kcxsnd_email_texts_appends( "tarou@yyy.xx.jp","","","定時報告",
78                                      mtext, mtexts, aptext, aptexts );
79
80          /* E メールを同じ時間内に連続送信しない為の簡単な遅延(sleep)処置 */
81          sleep( 60 );  /*1 分*/
82
83        }
84
85        /* 必須:タスクを１０秒停止させる(CPU 負荷を和らげる為) */
86        sleep(10);
87
88      }/*while*/
89
90 }
```

付 録

1． KaracrixBuilder を PC 起動時に自動的に立ち上げる方法

KaracrixBuilder を実際に運用する場合、PC に電源を投入し OS(CentOS)が立ち上がったところで、KaracrixBuilder も自動的に起動させることができると便利です。ここでは、CentOS を使用している場合の設定手順について解説します。動作の流れは以下の通りです。

1. 電源投入後 CentOS が立ち上がったところで、あなたのユーザ名※で自動ログインさせます。
2. 自動ログインしたところで、プログラム(KaracrixBuilder システム)を自動起動させます。

※KaracrixBuilder をインストールしたユーザアカウント名

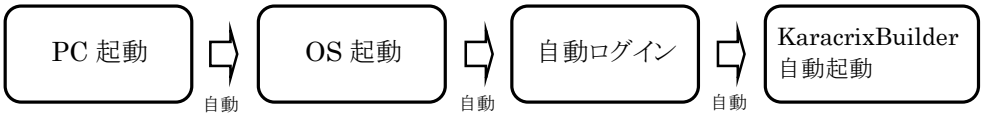


図 7.12.1 自動立ち上げフロー

●自動ログインの設定

CentOS のメニューバーのシステムメニューから「ログイン画面の設定」画面を表示します。

図 7.12.2 メニューバー(1)

「システム」→「管理」→「ログイン画面」(ルートのパスワード必要)

◇ユーザ登録

“ユーザ”タブを選択して下さい。下記の画面で①の「追加(A)」ボタンをクリックするとポップアップ画面が表示されます。そこであなたのユーザ名が自動ログインの対象になるよう設定追加します。ここでは、「karacrix」というユーザ名を追加する例で説明します。ユーザ名が登録されると「対象となるユーザ名(N):」画面部に表示されますので確認します。②の「閉じる」ボタンを一旦押し画面終了します。

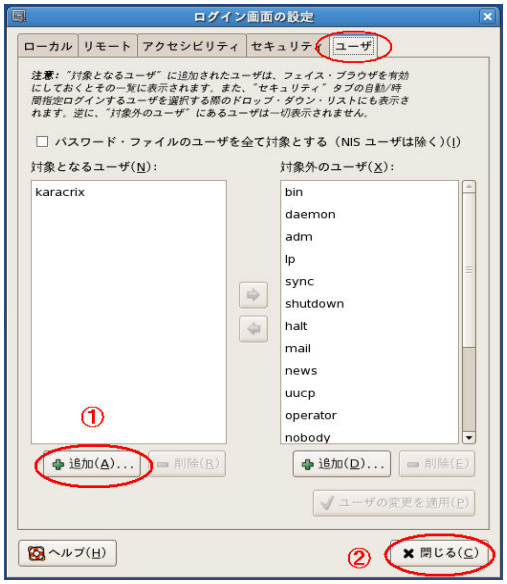


図 7.12.3 ログイン設定画面(1)

再び「ログイン画面の設定」画面を表示します。

図 7.12.4 メニューバー(2)

「システム」→「管理」→「ログイン画面」（ルートのパスワード必要）

◇ログインユーザ設定

次に“セキュリティ”タブを選択して下さい。

①にチェックを入れ、時間指定ログインを有効にします。

②より、あなたのユーザ名を選択します。(karacrix がユーザ名の例)

③で、あなたのユーザ名が指定されたことを確認します。

④で、自動ログインするまでの時間を入れます。時間は、10～30 秒程度が良いでしょう。これ以下ですとルートでログインし直す時などログイン待ち時間が取れず困ることになります。

⑤の「閉じる」ボタンで設定完了です。

※設定内容を確認後、CentOS あるいは PC をここで再起動して自動ログイン迄できることを確認して下さい。

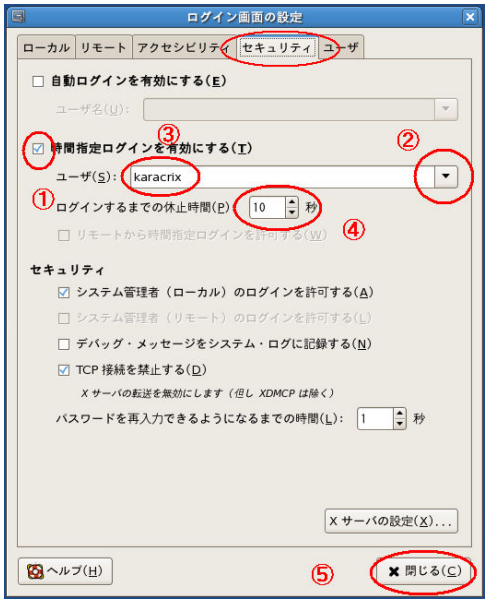


図 7.12.5 ログイン設定画面(2)

◇KaracrixBuilder 自動起動の設定

次にシステムメニューから「セッション」画面を表示して下さい。

図 7.12.6 メニューバー(3)

「システム」→「設定」→「他の個人設定」→「セッション」

①の「追加(A)」ボタンをクリックします。

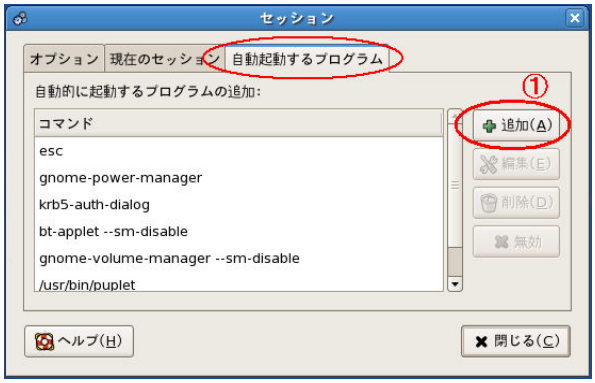


図 7.12.7 セッション画面(1)

「自動起動プログラムの追加」画面が表示されますので、①の「参照(B)」ボタンをクリックします。

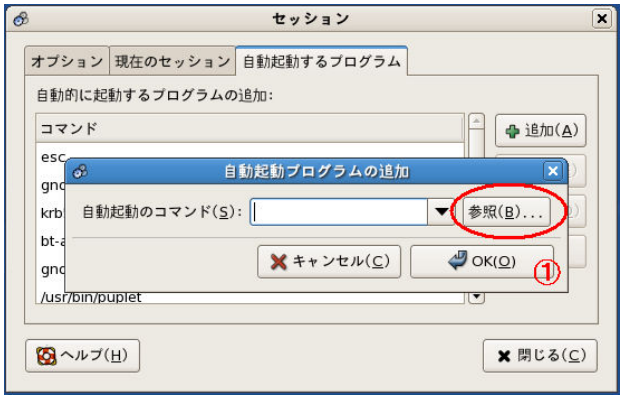


図 7.12.8 セッション画面(2)

KaracrixBuilder をインストールして展開すると、その中に起動シェル(karacrix.sh)が入っています。この起動シェル(karacrix.sh)が、KaracrixBuilder を実行しますので、この起動シェル(karacrix.sh)を自動起動させるコマンドに指定します。「自動起動のコマンド」画面で、①の様に展開した KaracrixBuilder(karacrix24A)の karacrix.sh をコマンドに指定してください。コマンドを指定したら、②の「開く(O)」ボタンをクリックします。

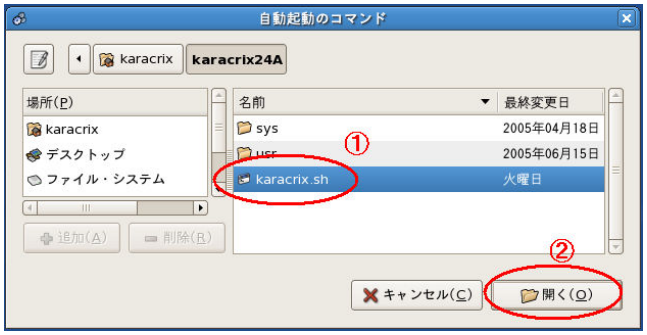


図 7.12.9 自動起動コマンド画面(1)

本ガイドでは、KaracrixBuilder を展開(インストール)したディレクトリパスは以下の通りです。
/home/**karacrix/**karacrix24A/karacrix.sh

「自動起動プログラムの追加」画面が表示されますので、①の(フルパス)コマンドを確認して下さい。普通にインストールしていれば、上記の**/karacrix/**の所があなたのユーザ名になっているはずです。②の「OK」で登録されます。

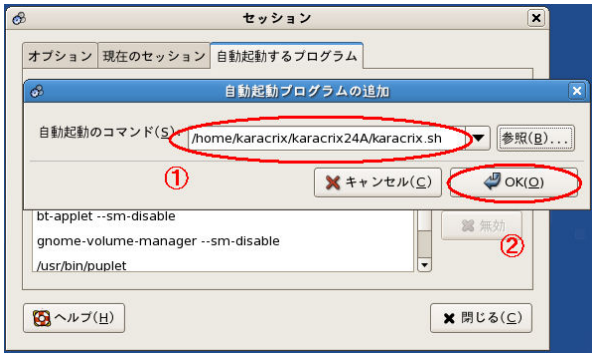


図 7.12.10 自動起動コマンド画面(2)

「セッション」画面の“自動起動するプログラム”タブを選択して下さい。そこで、自動起動するコマンドが、①のように画面で確認できれば設定完了です。②の「閉じる(C)」ボタンで終了して下さい。

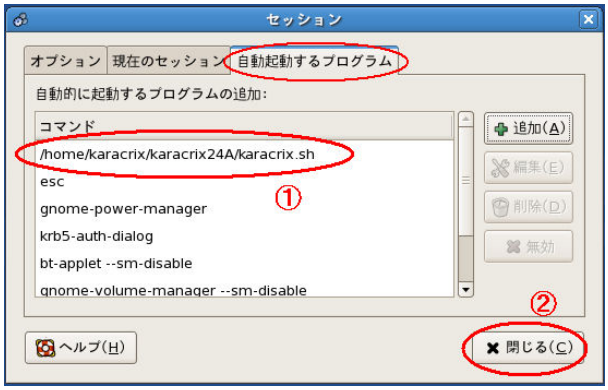


図 7.12.11 自動起動コマンド画面(3)

KaracrixBuilder が正常動作する状態であれば、CentOS あるいは PC をここで再起動してみてください。自動あるいは手動ログイン後、KaracrixBuilder が自動的に立ち上がるはずです。

2. 制御プログラムを PC 起動から一気に自動で立ち上げる方法

前項の解説で、PCの電源投入時に自動的に KaracrixBuilder が起動するところまで来ました。ここからさらに進めて、あなたの作成した制御プログラムを KaracrixBuilder 起動時に自動起動させる方法を説明します。KaracrixBuilder では、「制御プログラム登録」画面の先頭に登録されている ctl01 プログラムがコンパイルされて実行可能な場合、KaracrixBuilder 起動と同時に ctl01 プログラムも自動起動するような仕組みになっています。このことから、KaracrixBuilder では ctl01 プログラムのことをブートプログラムと呼んでいます。この機能を利用して、あなたのプログラムを自動起動してみます。

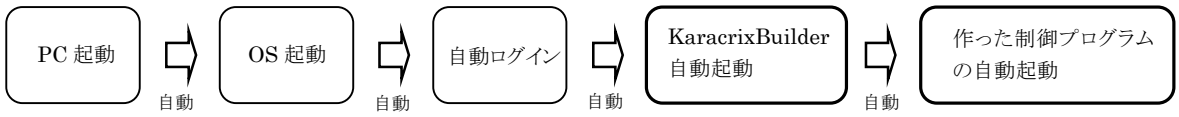


図 7.12.12 制御プログラム自動立ち上げフロー

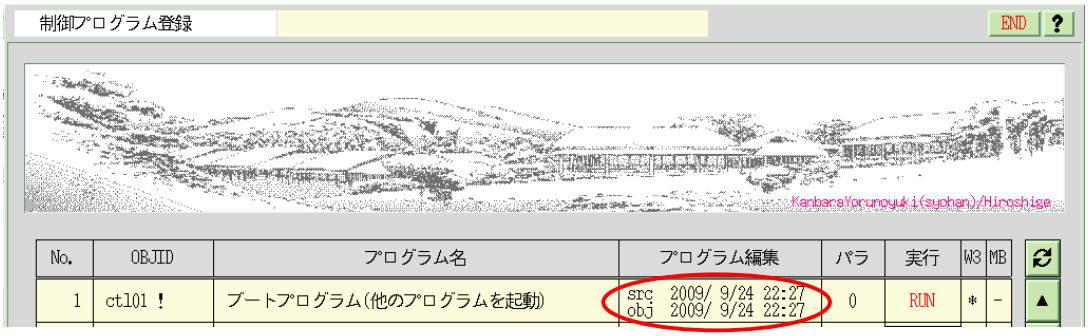


図 7.12.13 制御プログラム登録画面

ctl01 プログラムから、あなたが作った制御プログラムを起動するための記述は、ctl01 プログラム内にプログラム起動例として記述してありますので参照できます。

作った制御プログラムを起動する手順としては、

①先ず作った制御プログラムのプログラムオブジェクト ID をオープンします。
②次に、このプログラムオブジェクトの状態を 1 に設定します。
　この状態を 1 にする事が、プログラムの起動を意味しています。
　ちなみにプログラムを停止させる場合には、この状態を 0 に設定します。
③ブートプログラムである ctl01 プログラムからは、複数のプログラムを時間を見計いながら順次起動させる事もできます。また共通に使用するファイルデータやポイントオブジェクトの状態や属性値の動的な初期化をここで行っておくのも良いでしょう。

以下に、ブートプログラム(ctl01)内にプログラム起動の記述を書き込む例を示します。

```
objid_S1prg = kcxobj_open( "ctl??" );          /*ドライバ S1 を開きその OBJID を得る*/
objid_Yours = kcxobj_open( "ctl??" );          /*あなたのプログラムを開きその OBJID を得る*/
kcxobj_stat_iwt( objid_S1prg, 1 );             /*ドライバ S1 を起動させる*/
sleep( 60 );                                   /*ドライバ S1 を走らせ例えば 60 秒待つ*/
kcxobj_stat_iwt( objid_Yours, 1 );             /*そしてあなたのプログラムを起動*/
```

通信制御ドライバ S1 と作った制御プログラムを並列実行させて使用する場合には、注意が必要です。ドライバ S1 は、初期警報の発報を抑止するためアイドリング状態を設けています。また、リモート I/O 装置との初期通信でエラーリトライで時間を使っていることもあります。従って、通信制御ドライバ S1 を起動してしばらく時間を空けてからあなたのプログラムを実行するようにして下さい。時間は、環境によりますが約60～300秒待てば良いでしょう。