KaracrixBuilderV3 システムマニュアル

23章　KCX ライブラリリファレンス

（章別取扱説明書 v1.11）

株式会社 エスアイ創房

■改定履歴

第 1.00 版　2008/12/01
第 1.10 版　2015/08/10　　簡易ソースライブラリ追加（KBv3.50）
第 1.11 版　2015/11/07　　メール添付文補足説明追加（KBv3.51）

■ 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

23章 ＫＣＸライブラリリファレンス

KCXライブラリは、C言語から各種データの入出力を行うための関数を中心に、各種ユーティリティ関数を用意しています。

23.1 KCX 基本ライブラリー覧

KCX 基本ライブラリは、KaracrixBuilder の制御プログラムを作成する時に必要となる各種機能を提供する C 言語ライブラリです。

KCX ライブラリの関数に使用する各種定義は、karacrix.h ヘッダファイル（後述の【karacrix.h について】参照）に記述されていますのでアプリケーションプログラムを記述される際には、これを必ずインクルードする必要があります。

以下に、現在ご利用できる関数の一覧を示します。

23.2 KCX 印刷ライブラリー覧

KCX 印刷ライブラリは、KaracrixBuilder の帳票プログラムを作成する時に必要となる各種機能を提供する C 言語ライブラリです。

なお、KCX 印刷ライブラリを使用したアプリケーションを作成する際も、KCX 基本ライブラリの初期化関数 kcxinit を先頭で実行する必要があります。

以下に、現在ご利用できる関数の一覧を示します。

関数名	機能 (頁)
kcxrep_init	KCX 印刷ライブラリの初期設定をする (80)
kcxrep_formread	帳票フォームを読み込む (81)
kcxrep_drawtext	文字列を帳票フォーム上に印字する (82)
kcxrep_newpage	改ページをする (83)
kcxrep_output	帳票をプリンタに出力する (84)
kcxobj_rep_listform_get	日月報処理方法を得る (85)

※KCX 印刷ライブラリは、KaracrixBuilder500B でサポートしています。

23.3 KCX 画像記録制御ライブラリー覧

KCX 画像記録制御ライブラリは、KaracrixBuilder の画像記録をプログラムでコントロールする機能を提供するライブラリです。

なお、KCX 画像記録制御ライブラリを使用したアプリケーションを作成する際も、KCX 基本ライブラリの初期化関数 kcxinit を先頭で実行する必要があります。

以下に、現在ご利用できる関数の一覧を示します。

関数名	機能 (頁)
kcximg_quality_set	画像を記録する時の画質(JPEG 圧縮)を設定する (86)
kcximg_recctl_set	画像を記録する時の動作モードを設定する (87)

23.4 KCX レガシー通信ライブラリー覧

KCX レガシー通信ライブラリは、外部IO装置と通信するための機能を提供します。現在サポートされているものは、シリアルポート通信用の機能です。

なお、KCX レガシー通信ライブラリを使用したアプリケーションを作成する際も、KCX 基本ライブラリの初期化関数 kcxinit を先頭で実行する必要があります。

以下に、現在ご利用できる関数の一覧を示します。

関数名	機能 (頁)
kcxsio_interface_init	シリアルデバイスの初期設定をする (88)
kcxsio_recvfd	シリアルデバイスからデータを受信する (89)
kcxsio_sendfd	シリアルデバイスへデータを送信する (90)

23.5 KCX ソケット通信ライブラリー覧

KCX ソケット通信ライブラリは、他ホストや外部IO装置と通信するための機能を提供します。現在サポートされているものは、UDP/IP、TCP/IP ソケット通信用の機能です。
このライブラリは、OS 標準のソケットライブラリを使い易いようにモジュール化しただけのものです。解説には関数内部の構成も示してありますので OS 標準ソケットライブラリを用いて自作可能です。
なお、KCX ソケット通信ライブラリを使用したアプリケーションを作成する際も、KCX 基本ライブラリの初期化関数 kcxinit を先頭で実行する必要があります。

関数名	機能 (頁)
kcxudp_smpl_server_init	UDP 通信のサーバ側の初期設定をする (91)
kcxudp_smpl_client_init	UDP 通信のクライアント側の初期設定をする (92)
kcxudp_smpl_sendto	UDP サーバへデータを送信する (93)
kcxudp_smpl_return_sendto	UDP クライアントへデータを応答送信する (94)
kcxudp_smpl_block_recvfrom	UDP データをブロック受信する (95)
kcxudp_smpl_tmout_recvfrom	UDP データをタイムアウト付きで受信する (96)
kcxudp_received_packet_clean	UDP ポートに蓄積受信したデータを削除する (97)
kcxtcp_smpl_server_init	TCP 通信のサーバ側の初期設定をする (98)
kcxtcp_smpl_accept_newsockid	TCP 通信の accept 手続きをする (99)
kcxtcp_smpl_client_init	TCP 通信のクライアント側の初期設定をする (100)
kcxtcp_smpl_socket_connect	TCP 通信の connect 手続きをする (101)
kcxtcp_smpl_tmout_recv	TCP データをタイムアウト付きで受信する (102)

※ TCP/IP 通信のデータ送信及びブロック受信には、send、recv システムコールをご使用下さい。

23.6 KCX Eメールライブラリー覧

KCX Eメールライブラリは、テキスト情報をEメール(SMTP)送信するための機能を提供します。また、送信する E メールにはオプション(append)で KaracrixBuilder の監視画面や計測トレンドグラフを添付させる事ができます。オプションの記述方法は、5 章(5.16 E メールによる監視・操作)の監視パネル要求コマンド及び計測トレンドグラフ要求コマンドと共通(@kcxget 部は省く)です。
なお、KCX Eメールライブラリを使用したアプリケーションを作成する際も、KCX 基本ライブラリの初期化関数 kcxinit を先頭で実行する必要があります。
以下に、現在ご利用できる関数の一覧を示します。

関数名	機能 (頁)
kcxsnd_email_text	1 行のメッセージを E メールで送信する (103)
kcxsnd_email_texts	複数行のメッセージを E メールで送信する (104)
kcxsnd_email_text_append	1 行のメッセージに添付情報を添えて E メールで送信する (105)
kcxsnd_email_texts_append	複数行のメッセージに添付情報を添えて E メールで送信する (106)
kcxsnd_email_texts_appends	複数行のメッセージに複数の添付情報を添えて E メールで送信する (107)

23.7 karacrix.h について

karacrix.h ヘッダファイルは、利用者がプログラムをコンパイルする毎にKaracrixBuilder によって一時的に自動生成され、プログラムにインクルードされます。
自動生成に使用される参照ファイルは、$KARACRIX/sys/sinc/kcxux_program.h です。
これを雛形に、コンパイルするプログラム種類別に必要情報（帳票プログラム時はフォーム情報、制御プログラム時は無編集）が追加され karacrix.h が生成されます。
生成される karacrix.h は、$KARACRIX/sys/stmp/karacrix.h に置かれます。
なお、$KARACRIX/sys/stmp/karacrix.h は、KaracrixBuilder が起動する度に初期化削除されます。

23.8 簡易ソースライブラリ (KaracrixBuilder-v3.50 以降)

KCXライブラリで提供される関数の名前が冗長的で、使いにくい場合があります。
その場合用に、KCX 関数を包んだ簡単ソースライブラリ・ヘッダファイル kcxsrc.h を用意しました。
なお、この簡易ソースライブラリは単純使用の場合でエラーが起きないことを前提に書いてあります。
kcxsrc.h は、$KARACRIX/sys/sinc/kcxsrc.h にヘッダーソースの形で置かれます。
このヘッダーソースをプログラムから読み込む場合の書き方は、以下の通りです。
#include <kcxsrc.h>
ご利用に合わせ kcxsrc.h を改造するなどしてご使用ください。
簡易ソースライブラリの仕様に関しては、kcxsrc.h を直接お読みください。

1. k_open 関数 (ポイントのオープン)

   int k_open (char *objidname)

   例) int objid;
   objid = k_open("di001");

2. k_dget 関数 (デジタル型ポイントの状態取得)

   int k_dget (int objid)

   例) int status;
   status = k_dget(objid);

3. k_aget 関数 (アナログ型ポイントのデータ取得)

   double k_aget (int objid)

   例) double aval;
   aval = k_aget(objid);

4. k_dset 関数 (デジタル型ポイントの状態設定)

   int k_dset (int objid, int status)

   例) int status;
   status = 0;
   status = 1;
   k_dset(objid, status);

5. k_aset 関数 (アナログ型ポイントの状態設定)

   int k_aset (int objid, double data)

   例) double aval;
   aval = 1.2;
   k_aset(objid, aval);

6. k_email10 関数（10 行メールの送信）

int k_email(char *adrs, char *subject,
char *t1, char *t2, char *t3, char *t4, char *t5,
char *t6, char *t7, char *t8, char *t9, char *t10,
int mon, int mtre)

char *adrs： 送信先Eメールアドレス
char *subject： メールタイトル
char *t1～*t10： メール本文 10 行分
int mon： 添付する監視画面の番号[1 枚分] (0:添付しない)
int mtre： 添付する計測トレンドの番号[1 枚分] (0:添付しない)

例) char text1[256],text2[256],text3[256],text4[256],text5[256];
char text6[256],text7[256],text8[256],text9[256],text10[256];
/*初期化クリア*/
strcpy(text1,""); strcpy(text2,""); strcpy(text3,""); strcpy(text4,"");
strcpy(text5,""); strcpy(text6,""); strcpy(text7,""); strcpy(text8,"");
strcpy(text9,""); strcpy(text10,"");
/*本文作成*/
strcpy(text1, "How do you do.");
k_email10("abc@xxxx.jp",
"Hello",
text1,text2,text3,text4,text5,text6,text7,text8,text9,text10,
0, 0);

7. k_logout 関数（ログの作成） ::何か実行した時にログ記録しておくとトラブル時の解析にとても役に立ちます。

void k_logout(int logtype, int color, char *text)

int logtype: 0=操作ログ, 1=警報ログ
int color: 0=黒文字 1= 赤文字 2=青文字 3=緑文字
char *text: ログ記録しておきたいテキストデータ及びメッセージ

例) char text[81];
k_logout(0, 0, "OpeText");
sprintf(text, "AiErrorVal= %f", 12.34);
k_logout(1, 2, text);

23.9 KCX ライブラリ関数

kcxinit 関数

機能: KaracrixBuilder システムと結合してプログラムの初期設定をする。（必須）

書式: kcxinit(argc, argv)

引数: int argc コマンドライン引数の数（C 言語標準記述）

char *argv[] コマンドライン引数/文字列ポインタ配列（C 言語標準記述）

返値: 未定義

解説: KaracrixBuilder システムと結合してアプリケーションプログラムの初期化を行ないます。
本関数は、main 関数プログラム内の最先頭の実行文として記述する必要があります。
引数(KaracrixBuilder からプログラム実行時に渡される結合データ)には、main 関数のコマンド引数(argc,argv)を必要とします。

使用例:

```
#include <karacrix.h>

main( argc, argv )
int    argc;
char  *argv[];
{
  int  objid;

    kcxinit( argc, argv );

    objid = kcxobj_open( "di001" );

}
```

kcxobj_open 関数

機能: オブジェクトをオープンし識別子(OBJID)を得る。

書式: objid = kcxobj_open(objidname)
引数: char *objidname オブジェクト ID 名（最大 16 文字）
返値: int objid オブジェクト ID:正値（エラー時:負値）

解説: オブジェクトを KaracrixBuilder 内で扱うための ID を取得します。ファイルを扱う時にファイルディスクリプタを作成するのと同様に、オブジェクトを扱う時にオブジェクト名に対応する ID を取得します。
但し、ファイルディスクリプタの様に OS 資源を消費するものではありません。オブジェクトデータが格納されているメモリのアドレス(配列値)が返されます。資源が消費されるものではありませんので何度オープンしても構いません。ID はシステムで保持しているアドレスですのでクローズする必要もありません。

使用例:

```
int  objid1,objid2;
char text[17];

  objid1 = kcxobj_open( "di001" );

  if( objid1 >= 0 ){
    /* オブジェクトオープン成功 */
  }

  strcpy( text, "ai001" );

  objid2 = kcxobj_open( text );

  if( objid2 < 0 ){
    /* オブジェクトオープン失敗 */
  }
```

kcxobj_tquery 関数

機能: オブジェクトの分類種別型、状態型、整実数値型、入出力型を得る。

書式: status = kcxobj_tquery(objid, objtype, objatttype, objvaltype, objiotype)

引数:	int	objid	オブジェクト ID	
	int	*objtype	分類種別型	
			KcxOBJtp_POINT_DI:	デジタル入力型
			KcxOBJtp_POINT_DO:	デジタル出力型
			KcxOBJtp_POINT_PI:	カウンタ入力整数型
			KcxOBJtp_POINT_AI:	アナログ入力実数型
			KcxOBJtp_POINT_AO:	アナログ出力実数型
			KcxOBJtp_POINT_IMG;	イメージ画像型
	Int	*objatttype	状態型	
			KcxOBJtp_DIGITAL:	デジタル型 (DI,DO)
			KcxOBJtp_ANALOG:	アナログ、カウンタ型 (AI,AO,PI)
	Int	*objvaltype	整実数値型	
			KcxINTEGER:	整数型 (DI,DO,PI)
			KcxFLOAT:	実数型 (AI,AO)
	Int	*objiotype	入出力型	
			KcxIN:	入力型 (DI,PI,AI)
			KcxOUT:	出力型 (DO,AO)
返値:	int	status	正常動作:	0 (エラー時:負値)

解説: オブジェクトの各属性情報を取得します。
kcxobj_open 関数で取得したオブジェクト ID を指定することにより、そのオブジェクトの各属性情報を取得します。

使用例:

```
int  objid,objtype,objatttype,objvaltype,objiotype,status;

  objid  = kcxobj_open( "di001" );
  if( objid >= 0 ){
    status = kcxobj_tquery( objid,&objtype,&objatttype,&objvaltype,&objiotype );
    if( status >= 0 ){
      switch( objatttype ){
      case KcxOBJtp_DIGITAL:
           break;
      case KcxOBJtp_ANALOG:
           break;
      }
    }
  }
```

kcxobj_objnames_get 関数

機能: オブジェクト ID からオブジェクトの ID 名と名称を得る。

書式: status = kcxobj_objnames_get(objid, objidname, objname)

引数:	int	objid	オブジェクト ID
	char	*objidname	オブジェクト ID 名 (最大 16 文字)
	char	*objname	オブジェクト名称 (最大 31 文字)
返値:	int	status	正常動作:0 (エラー時:負値)

解説: オブジェクト ID からオブジェクトの ID 名とオブジェクト名称を取得します。

使用例:

```
int   objid;
int   status;
char  objidname[17];
char  objname  [32];

  objid  = kcxobj_open( "di001" );

  status = kcxobj_objnames_get( objid, objidname, objname );

  if( status >= 0 ){
    printf("オブジェクト ID 名 = %s ¥n", objidname );
    printf("オブジェクト名称 = %s ¥n", objname   );
  }
```

kcxobj_objidtagnames_get 関数

機能: オブジェクト ID からオブジェクトの ID 名と TAG(タグ)名と名称を得る。

書式: status = kcxobj_objidtagnames_get(objid, objidname, objtagname, objname)

引数:	int	objid	オブジェクト ID
	char	*objidname	オブジェクト ID 名 (最大 16 文字)
	char	*objtagname	オブジェクト TAG 名 (最大 16 文字)
	char	*objname	オブジェクト名称 (最大 31 文字)
返値:	int	status	正常動作:0 (エラー時:負値)

解説: オブジェクト ID からオブジェクトの ID 名、TAG(タグ)名、オブジェクト名称を取得します。

使用例:

```
int   objid;
int   status;
char  objidname [17];
char  objtagname[17];
char  objname   [32];

  objid  = kcxobj_open( "di001" );

  status = kcxobj_objidtagnames_get( objid, objidname, objtagname, objname );

  if( status >= 0 ){
    printf("オブジェクト ID 名  = %s ¥n", objidname  );
    printf("オブジェクト TAG 名 = %s ¥n", objtagname );
    printf("オブジェクト名称  = %s ¥n", objname    );
  }
```

kcxobj_prg_objidname_get 関数

機能: 実行中のアプリケーションプログラムのオブジェクト ID と ID 名を得る。

書式: objid = kcxobj_prg_objidname_get(objidname)

引数: char *objidname オブジェクト ID 名 (最大 16 文字)

返値: int objid オブジェクト ID (エラー無し:実行中の自身を呼ぶ為)

解説: 現在実行しているアプリケーションプログラム自身のプログラム・オブジェクトIDとID名を取得します。

使用例:

```
int   objid;
char  objidname[17];

  objid = kcxobj_prg_objidname_get( objidname );

  printf("This program ID      executed now is %d ¥n", objid     );
  printf("This program ID name executed now is %s ¥n", objidname );
```

kcxobj_point_objids_get 関数

機能: システムがポイント・オブジェクトに割り当てている始点と終点の ID を得る。

書式: kcxobj_point_objids_get(top_objid, end_objid)

引数: int *top_objid 始点のポイント・オブジェクト ID

int *end_objid 終点のポイント・オブジェクト ID

返値: 未定義

解説: ポイント登録画面で登録するオブジェクトは、(メモリ上に)連続してアドレス配置されます。
この始点のポイント・オブジェクト ID(整数値)と終点のオブジェクト ID(整数値)を得ます。
プログラムの for ループ文等でオブジェクトを連続して処理する時などに使用します。

使用例:

```
int  top_objid;
int  end_objid;
int  objid,objtype,objatttype,objvaltype,objiotype;

  kcxobj_point_objids_get( &top_objid, &end_objid );

  /* 状態値全クリア設定 */
  for( objid = top_objid; objid <= end_objid; objid++ ){
    kcxobj_tquery( objid,&objtype,&objatttype,&objvaltype,&objiotype );
    switch( objvaltype ){
    case KcxINTEGER:
         kcxobj_stat_iwt( objid, 0 );
         break;
    case KcxFLOAT:
         kcxobj_stat_fwt( objid, 0.0 );
         break;
    default:
         break;
    }
  }
```

kcxobj_defdlgwintype_set 関数

機能:オブジェクトの状態表示ダイアログ画面の種類を設定する。

書式: status = kcxobj_defdlgwintype_set(objid, type)

引数:

int	objid	オブジェクト ID
int	type	表示タイプ
		0: システムパラメータ設定依存
		1: データ表示のみ
		2: データ表示＋設定

返値: int status 正常動作:0 (エラー時:負値)

解説: 監視画面等でオブジェクトを選択した時に表示される状態表示用ダイアログ画面の種類を設定します。
データ表示のみの設定(1)を行なうと状態表示のみダイアログが表示され、データ表示＋設定の設定(2)を行なうと状態表示と状態値の設定を行なえるダイアログが表示されます。
システムパラメータ設定依存の設定(0)を行うと、「システムパラメータ設定」画面の「入力系ダイアログ画面選択」の定義(1 あるいは 2)によってダイアログの種類が決まります。
本設定は、通信制御プログラムが入力型オブジェクトの実対象(スイッチ等装置)と通信してその状態を書き換えるなどして管理している場合に、監視画面等から手動による状態の書き換えを禁止させる時等に(動的)使用します。
システム起動時の初期値は、ポイント登録画面の属性値「入力系ダイアログ画面選択」が 使用されます。
(「4 章モニタ画面による監視と操作 4.2.2 入力系ポイント状態ダイアログ画面の表示切替について」も参照)

使用例:

```
int  di_objid, ai_objid;

  /*
   *  「状態表示ダイアログ画面」を「データ表示のみ」にする。
   */
  di_objid = kcxobj_open( "di001" );
  kcxobj_defdlgwintype_set( di_objid, 1 );

  /*
   *  「状態表示ダイアログ画面」を「システムパラメータ設定依存」にする。
   */
  ai_objid = kcxobj_open( "ai001" );
  kcxobj_defdlgwintype_set( ai_objid, 0 );
```

kcxobj_dio_almornot_point_atbut_ird 関数

機能: デジタル型オブジェクトの通常点警報点定義を得る。

書式: status = kcxobj_dio_almornot_point_atbut_ird(objid, idata)

引数: int objid オブジェクト ID

int *idata 通常点警報点定義値

返値: int status 正常動作:0 (エラー時:負値)

解説: ポイント登録画面の属性値「通常点警報点定義」を取得します。
デフォルト(推奨)では、
通常点警報点定義値が 0 の場合、通常点を示し、
通常点警報点定義値が 1 の場合、警報点を示します。
※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  idata;
int  status;

  objid  = kcxobj_open( "di001" );

  status = kcxobj_dio_almornot_point_atbut_ird( objid, &idata );

  if( idata != 0 ){
    printf("警報点 ¥n");
  }else{
    printf("通常点 ¥n");
  }
```

kcxobj_stat_ird 関数

機能: 整数型オブジェクトの(主)状態値を得る。

書式: status = kcxobj_stat_ird(objid, idata)

引数: int objid オブジェクト ID

int *idata 状態値

返値: int status 正常動作:0 (エラー時:負値)

解説: 整数型オブジェクトの状態(KaracrixBuilder 内部メモリ)を調べる時に使用します。
この関数が使えるのは、objvaltype が KcxOBJtp_INTEGER のものに限ります。
objatttype が KcxOBJtp_DIGITAL の場合、
状態値が 0 で OFF、状態値が 1 で ON 状態を示します。

使用例:

```
int  objid;
int  idata;
int  status;

  objid  = kcxobj_open( "di001" );

  status = kcxobj_stat_ird( objid, &idata );
```

kcxobj_stat_iwt 関数

機能: 整数型オブジェクトに(主)状態値を設定する。

書式: status = kcxobj_stat_iwt(objid, idata)

引数: int objid オブジェクト ID

int idata 状態値

返値: int status 正常動作:0 (エラー時:負値)

解説: 整数型オブジェクトの状態(KaracrixBuilder 内部メモリ)を書き換える時に使用します。
この関数が使えるのは、objvaltype が KcxOBJtp_INTEGER のものに限ります。
objatttype が KcxOBJtp_DIGITAL の場合、
状態を OFF にしたい時は 0 値を、状態を ON にしたい時は 1 値を与えます。
※ここの状態は、監視画面、トレンドグラフや状態一覧等一般の主状態に使用されます。

使用例:

```
int  objid1,objid2;
int  idata;
int  status;

  objid1 = kcxobj_open( "di001" );
  objid2 = kcxobj_open( "di002" );

  idata  = 1;  /*オブジェクトに状態値を書込む*/
  status = kcxobj_stat_iwt( objid1, idata );

  status = kcxobj_stat_iwt( objid2, 0 );
```

kcxobj_stat_frd 関数

機能: 実数型オブジェクトの(主)状態値を得る。

書式: status = kcxobj_stat_frd(objid, fdata)

引数: int objid オブジェクト ID

double *fdata 状態値

返値: int status 正常動作:0 (エラー時:負値)

解説: 実数型オブジェクトの状態(KaracrixBuilder 内部メモリ)を調べる時に使用します。
この関数が使えるのは、objvaltype が KcxOBJtp_FLOAT のものに限ります。

使用例:

```
int    objid;
double fdata;
int    status;

  objid  = kcxobj_open( "ai001" );

  status = kcxobj_stat_frd( objid, &fdata );
```

kcxobj_stat_fwt 関数

機能: 実数型オブジェクトに(主)状態値を設定する。

書式: status = kcxobj_stat_fwt(objid, fdata)

引数:	int	objid	オブジェクト ID
	double	fdata	状態値
返値:	int	status	正常動作:0 (エラー時:負値)

解説: 実数型オブジェクトの状態(KaracrixBuilder 内部メモリ)を書き換える時に使用します。
この関数が使えるのは、objvaltype が KcxOBJtp_FLOAT のものに限ります。
※ここの状態は、監視画面、トレンドグラフや状態一覧等一般の主状態に使用されます。

使用例:

```
int    objid1,objid2;
double fdata;
int    status;

  objid1 = kcxobj_open( "ai001" );
  objid2 = kcxobj_open( "ai002" );

  fdata  = 3.1415;  /*オブジェクトの状態値を書き換える*/
  status = kcxobj_stat_fwt( objid1, fdata );

  status = kcxobj_stat_fwt( objid2, 1.234 );
```

kcxobj_atbut_ird 関数

機能: オブジェクトの汎用整数値属性を得る。

書式: status = kcxobj_atbut_ird(objid, no, idata)

引数:	int	objid	オブジェクト ID
	int	no	属性番号(1,2,3,4,5,6,7,8)
	int	*idata	整数値
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトには、数値演算及び表示やプログラム間通信等に使用する多目的整数値バッファメモリが 8 つ用意されています。

使用例:

```
int  objid;
int  idata;

  objid = kcxobj_open( "di001" );

  kcxobj_atbut_ird( objid, 1, &idata );

  printf("整数値=[%d]¥n", idata );
```

kcxobj_atbut_iwt 関数

機能: オブジェクトの汎用整数値属性を設定する。

書式: status = kcxobj_atbut_iwt(objid, no, idata)

引数:	int	objid	オブジェクト ID
	int	no	属性番号(1,2,3,4,5,6,7,8)
	int	idata	整数値
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトには、数値演算及び表示やプログラム間通信等に使用する多目的整数値バッファメモリが 8 つ用意されています。
数値は、監視画面からも呼び出して表示させることができます。

使用例:

```
int  objid;
int  idata;

  objid = kcxobj_open( "di001" );

  idata = 123;

  kcxobj_atbut_iwt( objid, 1, idata );
```

kcxobj_atbut_frd 関数

機能: オブジェクトの汎用実数値属性を得る。

書式: status = kcxobj_atbut_frd(objid, no, fdata)

引数:	int	objid	オブジェクト ID
	int	no	属性番号(1,2,3,4,5,6,7,8)
	double	*fdata	実数値
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトには、数値演算及び表示やプログラム間通信等に使用する多目的実数値バッファメモリが 8 つ用意されています。

使用例:

```
int    objid;
double fdata;

  objid = kcxobj_open( "di001" );

  kcxobj_atbut_frd( objid, 1, &fdata );

  printf("実数値=[%f]¥n", fdata );
```

kcxobj_atbut_fwt 関数

機能: オブジェクトの汎用実数値属性を設定する。

書式: status = kcxobj_atbut_fwt(objid, no, fdata)

引数:	int	objid	オブジェクト ID
	int	no	属性番号 (1,2,3,4,5,6,7,8)
	double	fdata	実数値
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトには、数値演算及び表示やプログラム間通信等に使用する多目的実数値バッファメモリが 8 つ用意されています。
数値は、監視画面からも呼び出して表示させることができます。

使用例:

```
int    objid;
double fdata;

  objid = kcxobj_open( "di001" );

  fdata = 3.14;

  kcxobj_atbut_fwt( objid, 1, fdata );
```

kcxobj_atbut_crd 関数

機能: オブジェクトの汎用文字列属性を得る。

書式: status = kcxobj_atbut_crd(objid, no, text)

引数:	int	objid	オブジェクト ID
	int	no	属性番号(1,2,3,4)
	char	*text	文字内容(最大 63 文字)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトには、メッセージ表示やプログラム間通信等に使用する多目的文字列バッファメモリが 4 つ用意されています。
kcxobj_atbut_cwt 関数による書き込みとは排他制御されています。

使用例:

```
int   objid;
char  text[64];

  objid = kcxobj_open( "di001" );

  kcxobj_atbut_crd( objid, 1, text );

  printf("メッセージ内容=[%s]¥n", text );
```

kcxobj_atbut_cwt 関数

機能: オブジェクトの汎用文字列属性を設定する。

書式: status = kcxobj_atbut_cwt(objid, no, text)

引数:	int	objid	オブジェクト ID
	int	no	属性番号 (1,2,3,4)
	char	*text	文字内容 (最大 63 文字)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトには、メッセージ表示やプログラム間通信等に使用する多目的文字列バッファメモリが 4 つ用意されています。
kcxobj_atbut_crd 関数による読み込みとは排他制御されています。
メッセージ内容は、監視画面からも呼び出して表示させることができます。

使用例:

```
int   objid;
char  text [64];
char  text2[64];

  objid = kcxobj_open( "di001" );

  strcpy( text, "現在 A001 エラー発生中です" );

  kcxobj_atbut_cwt( objid, 1, text );

  /*kcxobj_atbut_cwt()関数書込の確認*/

  kcxobj_atbut_crd( objid, 1, text2 );

  printf("確認=[%s]¥n", text2 );
```

kcxobj_item_table_cwt 関数

機能: オブジェクトの汎用属性項目名(テーブル)を設定する。

書式: status = kcxobj_item_table_cwt(objid, text)

引数: int objid オブジェクト ID

char *text テーブル名 (英数最大 8 文字[大小文字区別])

返値: int status 正常動作: 0 (エラー時:負値)

解説: オブジェクトの汎用(整数,実数,文字)属性を説明する項目名を設定します。
但し、この項目名はオブジェクトの属性個々に直接設定するのではなく、オブジェクトタイプ別に集めて参照テーブルとして記述したファイルから取得して使用されます。
(「18 章 18.4 ポイント汎用属性名称のインポート」を参照)
なお、ファイル名は下記のようにフォーマットと保存ディレクトリが定められています。

- ファイル名フォーマット:「kcxitem_objmem.????.tbl」
- ファイルは、KaracrixBuilder の$KARACRIX/usr/ustbl ディレクトリに置きます。

関数に設定するファイル名は、ファイル名全体ではなく上記フォーマットの????部(英数 8 文字以内任意)に当たる文字列のみを指定します。
例えば、$KARACRIX/usr/ustbl/kcxitem_objmem.ABC.tbl というファイルを指定する場合、ファイルを示すテーブル名に「ABC」を設定します。
KaracrixBuilder 起動時のデフォルト状態に戻す場合には、空文字("")を設定します。

使用例:

```
int   objid1,objid2,objid3;
char  text[9];

  objid1 = kcxobj_open( "di001" );
  objid2 = kcxobj_open( "di002" );
  objid3 = kcxobj_open( "di003" );

  kcxobj_item_table_cwt( objid1, "ABC" );

  strcpy( text, "12345XYZ" );
  kcxobj_item_table_cwt( objid2, text );


  kcxobj_item_table_cwt( objid3, "kcxsamp" );


  kcxobj_item_table_cwt( objid3,  ""  );  /*汎用属性項目をデフォルトに設定*/
```

kcxobj_scale_frd 関数

機能: アナログ型オブジェクトの表示用スケールを得る。

書式: status = kcxobj_scale_frd(objid, hi, low)

引数:

int	objid	オブジェクト ID
double	*hi	上限値
double	*low	下限値

返値: int status 正常動作: 0 (エラー時:負値)

解説: ポイント登録画面で設定したアナログ型オブジェクトの表示用スケールの上下限値を調べる時に使用します。
この関数が使えるのは、objatttype が KcxOBJtp_ANALOG のものに限ります。

使用例:

```
int    objid;
double hi, low;
int    status;

  objid  = kcxobj_open( "ai001" );

  status = kcxobj_scale_frd( objid, &hi, &low );
```

kcxobj_scale2_frd 関数

機能: アナログ型オブジェクトの各種スケールを得る。

書式: status = kcxobj_scale2_frd(objid, type, hi, low)

引数:

int	objid	オブジェクト ID
int	type	種別 (表示用:0, インターフェイス用:1)
double	*hi	上限値
double	*low	下限値

返値: int status 正常動作: 0 (エラー時:負値)

解説: アナログ型オブジェクトの表示用およびインターフェイス用スケールの上下限値を調べる時に使用します。

type に 0 を与えると、表示用のスケールが得られます。
(type を 0 とした場合、kcxobj_scale_frd 関数と同じ結果が得られます)

type に 1 を与えると、インターフェイス用のスケールが得られます。
このインターフェイス用スケールには、使用するセンサー(信号源)の工業単位出力(範囲)値が設定されているものとします。そして計測値を決定する計測プログラムは、この値を利用して通信取得している正規化データ(AD コンバータ値:0-1023,0-4095,0-65534 等)からスケールを案分する等して計測値を求めます。なお計測値を決めるに当たり、センサーの出力が非線形であったりまた有効範囲が有る等の場合、この補正に関する調整はアプリケーションプログラムの役割です。
センサー出力が線形でそのままアナログ表示スケールと同じ場合、本インターフェイス用のスケールをあえて利用する必要のない場合もあります。本値の使用、未使用に関しても使い方は全てアプリケーションに一任されています。
インターフェイス用のスケール値は必要に応じてプログラムのみが使用します。
この関数が使用できるのは、objatttype が KcxOBJtp_ANALOG のものに限ります。

使用例:

```
int    objid1,objid2,status;
double hi,low;

 objid1 = kcxobj_open( "ai001" );
 objid2 = kcxobj_open( "ai002" );

 status = kcxobj_scale2_frd( objid1, 0, &hi, &low );
 status = kcxobj_scale2_frd( objid2, 1, &hi, &low );
```

kcxobj_scale2_fwt 関数

機能: アナログ型オブジェクトの各種スケールを設定する。

書式: status = kcxobj_scale2_fwt(objid, type, hi, low)

引数:	int	objid	オブジェクト ID
	int	type	種別 (表示用:0, インターフェイス用:1)
	double	hi	上限値
	double	low	下限値
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: アナログ型オブジェクトの表示用およびインターフェイス用スケールの上下限値を書き換える時に使用します。

type が 0 の場合、表示用のスケール値として設定します。
この場合、KaracrixBuilder の監視画面のアナログメータのスケールや計測および記録トレンドグラフ画面の新規登録時の Y 軸スケールに影響を与えます。これは、表示用のスケール値が上記画面で使われているためです。設定には注意が必要です。

type が 1 の場合、インターフェイス用のスケール値として設定を書き換えます。システム起動時の初期値は、それぞれ
ポイント登録属性値「上限スケール値(表示用)」「下限スケール値(表示用)」
ポイント登録属性値「上限スケール値(通信用)」「下限スケール値(通信用)」
が使用されます。
この関数が使用できるのは、objatttype が KcxOBJtp_ANALOG のものに限ります。

使用例:

```
int    objid;
double hi;
double low;
int    status;

 objid  = kcxobj_open( "ai001" );
 hi     = 32.0;
 low    = 16.0;
 status = kcxobj_scale2_fwt( objid, 0, hi, low );
```

kcxobj_unit_query 関数

機能: アナログ型オブジェクトの単位情報を得る。

書式: status = kcxobj_unit_query(objid, unit_code, unit_name)

引数:	int	objid	オブジェクト ID
	int	*unit_code	単位コード
	char	*unit_name	単位名(最大 63 文字)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: ポイント登録画面で設定された単位情報を取得します。

使用例:

```
int   objid;
int   status;
int   unit_code;
char  unit_name[64];

  objid  = kcxobj_open( "ai001" );

  status = kcxobj_unit_query( objid, &unit_code, unit_name );

  printf("単位コード= %d ¥n", unit_code );
  printf("単位名    = %s ¥n", unit_name );
```

kcxobj_analog_format_get 関数

機能: アナログ型オブジェクトの数値変換フォーマットを得る。

書式: status = kcxobj_analog_format_get(objid, format)

引数: int objid オブジェクト ID

char *format 数値変換フォーマット (最大 8 文字)

返値: int status 正常動作:0 (エラー時:負値)

解説: ポイント登録画面で設定された表示フォーマットを取得します。

アナログ型 = %6.2f, %6.3f, %8.2f, %8.3f, %10.2f, %10.3f 等

カウンタ型 = %3d, %5d, %7d, %9d 等

使用例:

```
int  objid;
char format[9];
char text[128];

  objid = kcxobj_open( "ai001" );

  kcxobj_analog_format_get( objid, format );

  /* format -> %6d,%8d,%7.2f 等 */

  sprintf( text, format, 3.14 );
```

kcxobj_acm_maxcounts_ird 関数

機能: カウンタ型オブジェクトの積算カウンタ値の上限を得る。

書式: status = kcxobj_acm_maxcounts_ird(objid, idata)

引数:	int	objid	オブジェクト ID
	int	*idata	カウンタ上限値
返値:	int	status	正常動作:0 (エラー時:負値)

解説: カウンタ型オブジェクトのポイント登録属性値「積算カウンタ上限値」を取得します。
アナログ型オブジェクトの場合、実数型の kcxobj_acm_maxvalue_frd 関数を用いてください。

使用例:

```
int  objid;
int  idata;

  objid = kcxobj_open( "pi001" );

  kcxobj_acm_maxcounts_ird( objid, &idata );
```

kcxobj_acm_maxcounts_rate_ird 関数

機能: カウンタ型オブジェクトの積算カウンタの乗率を得る。

書式: status = kcxobj_acm_maxcounts_rate_ird(objid, idata)

引数: int objid オブジェクト ID
int *idata カウンタ乗率値 (10000 ベース)

返値: int status 正常動作:0 (エラー時:負値)

解説: カウンタ型オブジェクトのポイント登録属性値「積算カウンタ乗率」を取得します。
アナログ型オブジェクトの場合、実数型の kcxobj_acm_maxvalue_rate_frd 関数を用いて下さい。

乗率値は、10000 で乗率 1 とすることを推奨しています。
この条件下では、乗率が 10000 の時、積算カウンタ値は乗率の掛からないそのままの値
(1 倍)になる事を示します。
同じ条件で乗率値を 100 とした場合、カウンタ値は 0.01 倍の重み付けを意味し、
1000000 とした場合には 100 倍の重み付けを意味することになります。

例えば、1 カウンタ値が 1.2 ワット(W)というように重みがある場合に使用します。
この場合の乗率値は 10000 をベースとした場合、12000 です。
なおベース値の扱いは、アプリケーションに一任されています。

使用例:

```
int    objid;
int    count;
int    idata;
double fdata;

  objid = kcxobj_open( "pi001" );

  kcxobj_acm_maxcounts_rate_ird( objid, &idata );

  count = 12345;
  fdata = (double)count * ( (double)idata / 10000.0 );
```

kcxobj_acm_maxvalue_frd 関数

機能: アナログ型オブジェクトの実数積算値の上限を得る。

書式: status = kcxobj_acm_maxvalue_frd(objid, fdata)

引数:	int	objid	オブジェクト ID
	double	*fdata	積算上限値
返値:	int	status	正常動作:0 (エラー時:負値)

解説: アナログ型オブジェクトのポイント登録属性値「実数積算上限値」を取得します。
カウンタ型オブジェクトの場合、整数型の kcxobj_acm_maxcounts_ird 関数を用いてください。

一般的に積算には数値誤差の無い整数値を用います。但し、値に重み付けが有る場合等には乗率(rate)を乗じ実数にしなければならない時があります。この場合、カウンタ型オブジェクト値の実数用の代替擬似ポイントとしてアナログ型オブジェクト値が用いられます。
なお代替使用する場合には、「実数積算上限値」属性に、関連するカウンタ型オブジェクトのカウンタ上限値と同じものを実数にして環境を整合させておく必要があります。

使用例:

```
int    objid;
double fdata;

  objid = kcxobj_open( "ai001" );

  kcxobj_acm_maxvalue_frd( objid, &fdata );
```

kcxobj_acm_maxvalue_rate_frd 関数

機能: アナログ型オブジェクトの実数積算の乗率を得る。

書式: status = kcxobj_acm_maxvalue_rate_frd(objid, fdata)

引数:	int	objid	オブジェクト ID
	double	*fdata	積算乗率値 (1.0 ベース)
返値:	int	status	正常動作:0 (エラー時:負値)

解説: アナログ型オブジェクトのポイント登録属性値「実数積算乗率」を取得します。
カウンタ型オブジェクトの場合、整数型の kcxobj_acm_maxcounts_rate_ird 関数を用いてください。

カウンタ型オブジェクト値の実数用の代替擬似ポイントとしてアナログ型オブジェクト値が用いられる場合、「実数積算乗率」属性に、関連するカウンタ型オブジェクトのカウンタ乗率値と同じものを実数にして環境を整合させておく必要があります。

使用例:

```
int    objid;
double fdata;

  objid = kcxobj_open( "ai001" );

  kcxobj_acm_maxvalue_rate_frd( objid, &fdata );
```

kcxobj_lock_ird 関数

機能: オブジェクトのロック状態を得る。

書式: status = kcxobj_lock_ird(objid, idata)

引数:	int	objid	オブジェクト ID
	int	*idata	状態値 (ロック OFF:0, ロック ON:1)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトのロック状態を調べる時に使用します。
例えば、警報用メール送信の一時停止等にも利用されます。
※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  status;
int  idata;

 objid  = kcxobj_open( "di001" );

 status = kcxobj_lock_ird( objid, &idata );
```

kcxobj_lock_iwt 関数

機能: オブジェクトにロック状態を設定する。

書式: status = kcxobj_lock_iwt(objid, idata)

引数:	int	objid	オブジェクト ID
	int	idata	状態値 (ロック OFF:0, ロック ON:1)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトのロック状態を書き換える時に使用します。
ロック状態は、一覧画面やダイアログ画面等に反映されます。
システム起動時のデフォルトは、ロック OFF 状態です。

使用例:

```
int  objid;
int  status;
int  lock_status;

 objid       = kcxobj_open( "di001" );
 lock_status = 0;  /*ロック OFF 状態に書き換える*/
 status      = kcxobj_lock_iwt( objid, lock_status );
 lock_status = 1;  /*ロック ON 状態に書き換える*/
 status      = kcxobj_lock_iwt( objid, lock_status );
```

kcxobj_online_ird 関数

機能: オブジェクトの通信状態を得る。

書式: status = kcxobj_online_ird(objid, idata)

引数:	int	objid	オブジェクト ID
	int	*idata	状態値(オフライン:0, オンライン:1)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトの I/O 通信のオン(オフ)ライン状態を調べる時に使用します。

使用例:

```
int  objid;
int  status;
int  idata;

 objid  = kcxobj_open( "di001" );
 status = kcxobj_online_ird( objid, &idata );
```

kcxobj_online_iwt 関数

機能: オブジェクトの通信状態を設定する。

書式: status = kcxobj_online_iwt(objid, idata)

引数: int objid オブジェクト ID

int idata 状態値(オフライン:0, オンライン:1)

返値: int status 正常動作: 0 (エラー時:負値)

解説: オブジェクトの I/O 通信のオン(オフ)ライン状態を書き換える時に使用します。
通信状態は、一覧画面等に反映されます。
本状態をオフラインにすると以下の機能にも影響を与えます。

1. オブジェクトの状態が自動計測記録されている場合、
   生成される記録ファイル(sys_itvYYMM.log)のレコードには、
   (オフラインの為)削除フラグ(編集可)が上がった状態で記録されます。
2. 計測トレンドグラフでは、グラフ線色にオフライン用の色(デフォルト灰色)が用いられます。
3. 記録トレンドグラフには、オフライン状態の計測点の描画がされません。

※システム起動時のデフォルトは、オンライン状態です。

使用例:

```
int  objid;
int  status;
int  online_status;

 objid         = kcxobj_open( "di001" );
 online_status = 0;  /*オフライン状態に書き換える*/
 status        = kcxobj_online_iwt( objid, online_status );
 online_status = 1;  /*オンライン状態に書き換える*/
 status        = kcxobj_online_iwt( objid, online_status );
```

kcxobj_log_perm_ird 関数

機能: オブジェクトの計測記録許可を得る。

書式: status = kcxobj_log_perm_ird(objid, idata)

引数: int objid オブジェクト ID

int *idata 許可値(計測記録保留:0, 計測記録許可:1)

返値: int status 正常動作: 0 (エラー時:負値)

解説: KaracrixBuilder システムの計測記録ロガー(sys_itvYYMM.log)に対する記録許可値を読み込みます。

使用例:

```
int  objid;
int  status;
int  perm;

 objid  = kcxobj_open( "di001" );

 status = kcxobj_log_perm_ird( objid, &perm );
```

kcxobj_log_perm_iwt 関数

機能: オブジェクトの計測記録許可を設定する。

書式: status = kcxobj_log_perm_iwt(objid, idata)

引数: int objid オブジェクト ID

int idata 許可値(計測記録保留:0, 計測記録許可:1)

返値: int status 正常動作: 0 (エラー時:負値)

解説: KaracrixBuilder システム内の計測記録ロガー(sys_itvYYMM.log)の記録実行を制御します。
(ロガーはシステム内で本設定を参照しながら非同期動作しています)
計測データに信頼性が無く(例えば通信オフライン時)、その様なデータ記録をパス(回避)させたい場合等、記録を一時的に停止させる時に使用します。
なお記録の制御を行うオブジェクトは、計測記録実行の対象 (記録条件設定画面の実行欄) が「ON」になっている必要があります。 (「21 章 21.2 計測データ記録条件設定」を参照)

使用例:

```
int  objid;
int  status;

 objid  = kcxobj_open( "di001" );

 status = kcxobj_log_perm_iwt( objid, 0 );
 sleep( 10 );
 status = kcxobj_log_perm_iwt( objid, 1 );
```

kcxobj_protectsec_ird 関数

機能: 出力型オブジェクトの操作保護時間を得る。

書式: status = kcxobj_protectsec_ird(objid, idata)

引数: int objid オブジェクト ID

int *idata 時間(秒)

返値: int status 正常動作: 0 (エラー時:負値)

解説: オブジェクトの操作保護時間を調べる時に使用します。

この関数が使えるのは、objiotype が、KcxOUT の出力型のものに限ります。

※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  status;
int  isec;

 objid  = kcxobj_open( "do001" );
 status = kcxobj_protectsec_ird( objid, &isec );
```

kcxobj_protectsec_iwt 関数

機能: 出力型オブジェクトに操作保護時間を設定する。

書式: status = kcxobj_protectsec_iwt(objid, idata)

引数:	int	objid	オブジェクト ID
	int	idata	時間(秒)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトの操作保護時間を書き換える時に使用します。
この関数が使えるのは、objiotype が、KcxOUT の出力型のものに限ります。
システム起動時の初期値は、ポイント登録画面の属性値「連続操作禁止時間(秒)」が使用されます。

使用例:

```
int  objid;
int  status;
int  isec;

 objid  = kcxobj_open( "do001" );
 isec   = 120;  /*操作保護時間を 120 秒に書き換えます*/
 status = kcxobj_protectsec_iwt( objid, isec );
```

kcxobj_operator_get 関数

機能: オブジェクトのオペレータ(操作者)を得る。

書式: status = kcxobj_operator_get(objid, opetime, opename)

引数:

int	objid	オブジェクト ID
time_t	*opetime	オペレート時間 (UNIX 時間)
char	*opename	オペレータ名 (最大 80 文字)

返値:

int	status	0: 初期状態
		1: システム(予約)
		2: スケジューラ操作
		3: リモート(Web,E メール l)操作
		4: 手動(画面)操作
		5: ユーザプログラム操作

解説: 出力型ポイントオブジェクト及びプログラムオブジェクトを操作した者とその時刻を調べる時に使用します。

(返り値は、「付録Aディレクトリ構成」 内の$KARACRIX/sys/stbl/kcxname_operator.tbl も参考)

E メールによってオブジェクトが操作された場合、そのメールアドレスをオペレータ名(opename 引数)から取得できます。 但し本関数から得られるオペレータ名は次のＥメール操作時に更新されます。

使用例:

```
int     objid;
time_t  opetime;
char    opename[81];
int     status;

  objid  = kcxobj_open( "do001" );

  status = kcxobj_operator_get( objid, &opetime, opename );

  if( status == 3 ){
    /*注意：3 が EMail によるものかのチェックは別途必要*/
    kcxsnd_email_text( opename,"","","ReplyToYou","DidYouControl?");
  }
```

kcxobj_operator_set 関数

機能: オブジェクトのオペレータ(操作者)を設定する。

書式: status = kcxobj_operator_set(objid, idata, opename)

引数:	int	objid	オブジェクト ID
	int	idata	オペレータコード
	char	*opename	オペレータ名 (最大 80 文字)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトを操作した者を書き換えます。
オペレータコードには、「24 章 システムとファイル仕様(資料 4 オペレータコードテーブルファイル)」内の左辺(idcode)の番号から選択して使用して下さい。
オペレータ名を空にする場合には、空文字列(0 ターミネート)を与えます。
オペレータ名を更新させない場合には、opename に NULL ポインターを与えてください。
通常、オブジェクトの実対象を操作する時に用いる kcxobj_sndstatope_fromkcx 関数とセットで使用します。

使用例:

```
int   objid;
int   status;
int   operator;
char  opename[81];
KcxIntFlt_t  udata;

 objid    = kcxobj_open( "ai001" );

 /*オペレータコードとオペレータ名を更新*/
 operator = 3;
 strcpy( opename, "abc@xxx.jp" );
 status   = kcxobj_operator_set( objid, operator, opename );


/*オペレータコードを更新、同時にオペレータ名を空にする*/
 operator = 4;
 strcpy( opename, "" );
 status   = kcxobj_operator_set( objid, operator, opename );
```

```
/*オペレータコードを更新、オペレータ名は更新させずに保留*/
operator = 5;
status   = kcxobj_operator_set( objid, operator, (char *)NULL );

/*kcxobj_sndstatope_fromkcx()関数から得たオペレータコードで更新*/
switch( kcxobj_sndstatope_fromkcx( &objid, &udata, &operator ) ){
case KcxINTEGER:
case KcxFLOAT:
     kcxobj_operator_set( objid, operator, (char *)NULL );
     break;
case KcxNULL:
default:
     break;
}
```

kcxobj_alm_perm_ird 関数

機能: オブジェクトの警報監視許可を得る。

書式: status = kcxobj_alm_perm_ird(objid, idata)

引数: int objid オブジェクト ID

int *idata 許可値 (許可:1 以上、不許可:0)

返値: int status 正常動作: 0 (エラー時:負値)

解説: オブジェクトを警報監視するか否かの許可を調べる時に使用します。

※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  status;
int  perm;

 objid  = kcxobj_open( "di001" );
 status = kcxobj_alm_perm_ird( objid, &perm );

 objid  = kcxobj_open( "ai001" );
 status = kcxobj_alm_perm_ird( objid, &perm );
```

kcxobj_alm_perm_iwt 関数

機能: オブジェクトに警報監視許可を設定する。

書式: status = kcxobj_alm_perm_iwt(objid, idata)

引数: int objid オブジェクト ID

int idata 許可値 (許可:1 以上、不許可:0)

返値: int status 正常動作: 0 (エラー時:負値)

解説: オブジェクトの警報監視許可を書き換える時に使用します。
なお、この許可は KaracrixBuilder 内部のシステムにも使われています。不許可にするとシステムはこれに連動し内部の警報履歴ファイル生成機能を(オブジェクト毎)停止させます。
同時に関連関数となる kcxobj_alm_log_10uwt 関数も連動し内部処理がキャンセルされるようになりますので設定にはご注意下さい。
システム起動時の初期値は、ポイント登録画面の下記設定値が使用されます。
デジタル型オブジェクトの場合、ポイント登録属性の「警報発生許可」値となります。
アナログ型オブジェクトの場合、ポイント登録属性の「警報発生許可(元)」値となります。

使用例:

```
int  objid;
int  status;
int  perm;

 objid  = kcxobj_open( "di001" );
 perm   = 0;  /*警報監視許可(及び警報履歴生成)しない*/
 status = kcxobj_alm_perm_iwt( objid, perm );
 perm   = 1;  /*警報監視許可(及び警報履歴生成)する*/
 status = kcxobj_alm_perm_iwt( objid, perm );
```

kcxobj_alm_stat_ird 関数

機能: オブジェクトの警報状態を得る。

書式: status = kcxobj_alm_stat_ird(objid, idata)

引数: int objid オブジェクト ID

int *idata 状態値 (警報発生中:1 以上、警報停止中:0)

返値: int status 正常動作: 0 (エラー時:負値)

解説: オブジェクトの警報状態を調べる時に使用します。

※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  status;
int  alarm_status;

 objid  = kcxobj_open( "di001" );
 status = kcxobj_alm_stat_ird( objid, &alarm_status );
```

kcxobj_alm_stat_iwt 関数

機能:オブジェクトに警報状態を設定する。

書式:status = kcxobj_alm_stat_iwt(objid, idata)

引数:	int	objid	オブジェクト ID
	int	idata	状態値 (警報発生中:1 以上、警報停止中:0)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: オブジェクトの警報状態を書き換える時に使用します。
警報状態を書き換えると、一覧画面やダイアログ画面等にも状態反映されます。
システム起動時の初期値は、0 値です。
本関数は警報履歴データを生成する kcxobj_alm_log_10uwt 関数とは関連ありません。

使用例:

```
int  objid;
int  status;
int  alarm_status;

 objid        = kcxobj_open( "di001" );
 alarm_status = 0;  /*警報停止中に書き換える*/
 status       = kcxobj_alm_stat_iwt( objid, alarm_status );
 alarm_status = 1;  /*警報発生中に書き換える*/
 status       = kcxobj_alm_stat_iwt( objid, alarm_status );
```

kcxobj_alm_level_ird 関数

機能: オブジェクトの警報レベルを得る。

書式: status = kcxobj_alm_level_ird(objid, idata)

引数:

int	objid	オブジェクト ID
int	*idata	レベル値

返値:

int	status	正常動作: 0 (エラー時:負値)

解説: ポイント登録画面の属性値「履歴警報レベル」を取得します。
オブジェクトの警報レベルを調べる時に使用します。デジタル型オブジェクトの場合に主に用いられます。アナログ型オブジェクトの場合、予備(オプション)として使われ、段階別(HML)アナログ上下限警報レベルを用いない場合の備えとなっています。
※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  status;
int  idata;

 objid  = kcxobj_open( "di001" );
 status = kcxobj_alm_level_ird( objid, &idata );
```

kcxobj_alm_onlogic_ird 関数

機能: デジタル型オブジェクトの警報 I/O 条件を得る。

書式: status = kcxobj_alm_onlogic_ird(objid, idata)

引数: int objid オブジェクト ID

int *idata I/O 条件値

返値: int status 正常動作: 0 (エラー時:負値)

解説: ポイント登録画面の属性値「警報値」を取得します。
デジタル型オブジェクトを警報とする I/O のインターフェイス条件を調べる時に使用します。
I/O の状態が、例えば 1 の閉(ショート)の場合に警報発生、0 の開(オープン)の場合に警報停止等の警報判断に用います。
※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  status;
int  alarm_onlogic;

 objid  = kcxobj_open( "di001" );
 status = kcxobj_alm_onlogic_ird( objid, &alarm_onlogic );
```

kcxobj_alm_ai_limit_frd 関数

機能: アナログ型オブジェクトの上下限警報に関するデータを得る。

書式: status = kcxobj_alm_ai_limit_frd(objid, type, enable, perm, dpy, his_hi, his_low)

引数:			
	int	objid	オブジェクト ID
	int	type	種別 (警報レベル)
			上限(H***) : 33
			上限(M**) : 32
			上限(L*) : 31
			下限(L*) : 21
			下限(M**) : 22
			下限(H***) : 23
	int	*enable	機能有効無効 (有効:1,無効:0)
	int	*perm	警報許可 (許可:1,不許可:0)
	double	*dpy	表示値
	double	*his_hi	ヒステリシス上限値 (計算時に使用)
	double	*his_low	ヒステリシス下限値 (計算時に使用)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: 上下限警報レベル別の、機能有効無効値、警報許可値、警報表示値、警報ヒステリシス上下限値を調べるときに使用します。
アナログ警報を発生させるアプリケーションプログラムが、オブジェクトの警報発生及び停止の判断に本設定値を使用することを想定しています。
本関数で取得した値の使用方法は、アプリケーションに一任されます。
[補足]
警報許可(perm)は、機能有効無効(enable)が、有効(1)時に使用可能となります。
例えば、種別(type)32 と 22 の機能有効無効(enable)のみ有効(1)にした場合、警報許可(perm) は、種別(type)32 と 22 のみ有効となり、警報システムとしては、上限1段、下限1段の警報監視システムとなります。
また、種別(type)33,32,22,23 の機能有効無効(enable)を有効(1)にした場合、警報許可(perm)は、種別(type)33,32,22,23 のみ有効となり、警報システムとしては、上限 2 段、下限 2 段の警報監視システムとなります。

使用例:

```
int     objid,type,  enable,perm,status;
double dpy,his_hi,his_low;

 objid  = kcxobj_open( "ai001" );
 type   = 33;
 status = kcxobj_alm_ai_limit_frd( objid, type, &enable, &perm, &dpy, &his_hi, &his_low );
```

kcxobj_alm_ai_limit_fwt 関数

機能: アナログ型オブジェクトに上下限警報に関するデータを設定する。

書式: status = kcxobj_alm_ai_limit_fwt(objid, type, enable, perm, dpy, his_hi, his_low)

引数:			
	int	objid	オブジェクト ID
	int	type	種別 (警報レベル)
			上限(H***) : 33
			上限(M**) : 32
			上限(L*) : 31
			下限(L*) : 21
			下限(M**) : 22
			下限(H***) : 23
	int	enable	機能有効無効 (有効:1,無効:0)
	int	perm	警報許可 (許可:1,不許可:0)
	double	dpy	表示値
	double	his_hi	ヒステリシス上限値 (計算時に使用)
	double	his_low	ヒステリシス下限値 (計算時に使用)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: 上下限警報レベル別の、機能有効無効値、警報許可値、警報表示値、警報ヒステリシス上下限値を書き換える時に使用します。
システム起動時の初期値は、ポイント登録画面の下記属性設定値が使用されます。
有効無効値属性は「警報機能有効」、許可値属性は「警報監視」、警報表示値属性は「警報値(表示用)」、警報ヒステリシス上下限値属性は「警報値(不感帯[上/下]値)」が使用されます。

使用例:

```
int    objid,type,enable,perm,status;
double dpy,his_hi,his_low;

 objid   = kcxobj_open( "ai001" );
 type    = 33;
 enable  = 1;
 perm    = 1;
 his_hi  = 102.0;
 dpy     = 100.0;
 his_low  =  98.0;
 status  = kcxobj_alm_ai_limit_fwt( objid, type, enable, perm, dpy, his_hi, his_low );
```

kcxobj_event_opelog_perm_ird 関数

機能: オブジェクトの状態変化記録許可を得る。

書式: status = kcxobj_event_opelog_perm_ird(objid, idata)

引数: int objid オブジェクト ID

int *idata 許可値

返値: int status 正常動作: 0 (エラー時:負値)

解説: デジタル型オブジェクトのポイント登録属性値「状態変化記録許可」を取得します。
ポイント属性の「通常点警報点」が通常点の場合の利用を想定しています。この場合、オン・オフ状態が頻繁に変化するものへの許可はご注意下さい。膨大なオンオフ記録(ログ)が生成され他のオブジェクトの履歴検索の処理速度の低下を招く場合があります。
なお、「通常点警報点」が警報点の場合には、アプリケーションである警報マネージャプログラムが「履歴警報レベル」の方を使い管理しますので、本許可を用いることは通常ありません。
※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  status;
int  opelog_perm;

  objid  = kcxobj_open( "di001" );
  status = kcxobj_event_opelog_perm_ird( objid, &opelog_perm );
```

kcxobj_event_opelog_level_ird 関数

機能: オブジェクトの状態変化操作履歴レベルを得る。

書式: status = kcxobj_event_opelog_level_ird(objid, idata)

引数: int objid オブジェクト ID

int *idata レベル値

返値: int status 正常動作: 0 (エラー時:負値)

解説: デジタル型オブジェクトのポイント登録属性値「状態変化操作履歴レベル」を取得します。
操作履歴へ出力する際のレベル(3,2,1,0)値に使われることを想定しています。
※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  status;
int  opelog_level;

  objid  = kcxobj_open( "di001" );
  status = kcxobj_event_opelog_level_ird( objid, &opelog_level );
```

kcxobj_event_email_perm_get 関数

機能: オブジェクトのメール送信許可を得る。

書式: status = kcxobj_event_email_perm_get(objid, type, perm)

引数:	int	objid	オブジェクト ID
	int	type	種別 (デジタル型オブジェクトの場合) 警報レベルというものが存在していないので未定義としての 0 値を指定 (アナログ型オブジェクトの場合) 警報レベル別に下記値を指定 上限(H***) : 33 上限(M**) : 32 上限(L*) : 31 下限(L*) : 21 下限(M**) : 22 下限(H***) : 23
	int	*perm	許可値 (許可:1,不許可:0)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: ポイント登録画面の属性値「警報メール許可」を取得します。
※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  type;
int  perm;
int  status;

  objid  = kcxobj_open( "di001" );
  type   = 0;
  status = kcxobj_event_email_perm_get( objid, type, &perm );

  objid  = kcxobj_open( "ai001" );
  type   = 33;
  status = kcxobj_event_email_perm_get( objid, type, &perm );
```

kcxobj_event_log_key_get 関数

機能: オブジェクトの履歴書式キー番号を得る。

書式: status = kcxobj_event_log_key_get(objid, type, key)

引数: int objid オブジェクト ID

int type 種別

(デジタル型オブジェクトの場合)

警報レベルというものが存在していないので未定義としての 0 値を指定

(アナログ型オブジェクトの場合)

警報レベル別に下記値を指定

上限(H***) : 33

上限(M**) : 32

上限(L*) : 31

下限(L*) : 21

下限(M**) : 22

下限(H***) : 23

int *key キー番号

返値: int status 正常動作: 0 (エラー時:負値)

解説: ポイント登録画面の属性値「履歴書式キー番号(Ev_KEY)」を取得します。

履歴へ出力する際のキー番号に使われることを想定しています。

使用例:

```
int  objid;
int  type;
int  key;
int  status;

  objid  = kcxobj_open( "di001" );
  type   = 0;
  status = kcxobj_event_log_key_get( objid, type, &key );

  objid  = kcxobj_open( "ai001" );
  type   = 33;
  status = kcxobj_event_log_key_get( objid, type, &key );
```

kcxobj_control_code_ird 関数

機能: 警報制御コマンドコードを得る。

書式: status = kcxobj_control_code_ird(objid, type, idata)

引数:	int	objid	オブジェクト ID
	int	type	種別 (デジタル型オブジェクトの場合) 警報レベルというものが存在していないので未定義としての 0 値を指定 (アナログ型オブジェクトの場合) 警報レベル別に下記値を指定 上限(H***) : 33 上限(M**) : 32 上限(L*) : 31 下限(L*) : 21 下限(M**) : 22 下限(H***) : 23
	int	*idata	コマンドコード値
返値:	int	status	正常動作:0 (エラー時:負値)

解説: ポイント登録画面の属性値「警報制御コマンドコード」を取得します。
※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  status;
int  code;

  objid  = kcxobj_open( "di001" );

  status = kcxobj_control_code_ird( objid, 0, &code );

  objid  = kcxobj_open( "ai001" );

  status = kcxobj_control_code_ird( objid, 33, &code );
```

kcxobj_itv_log_uwt 関数

機能: 計測記録データを追加作成する。

書式: status = kcxobj_itv_log_uwt(objid, udata, rate)

引数:	int	objid	オブジェクト ID
	KcxIntFlt_t	udata	状態値 オブジェクトが整数型(udata.i)か実数型(udata.f)かで共用体の使用する型を替える必要があります。
	KcxIntFlt_t	rate	カウンタ乗率等のレート (予備データの為未使用可、この場合0クリアデータ推奨)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: 計測記録を1レコード、システムの計測記録ファイル(sys_itvYYMM.log)に追加します。
計測データ記録条件設定画面によって実行(ON)設定されているオブジェクトのデータは、KaracrixBuilder システム内部においても本関数を使用して記録しています。
なお本関数を使用したプログラムでは、計測記録設定画面(分単位)では扱っていない秒単位の記録に使用することもできます。

※本関数は KCX ライブラリの一部として組み込まれていますが、
$KARACRIX/sys/ssrc/kcxlib_src.c にソースプログラムとして見ることが出来ます。
仕様を変更して応用したい場合には、このソースプログラム関数を自分のプログラム内に別名で複写し改造する等してご使用ください。

使用例:

```
int          objid_ai,objid_pi,status;
KcxIntFlt_t  udata,rate;

 objid_ai = kcxobj_open( "ai001" );
 objid_pi = kcxobj_open( "pi001" );

 udata.f = 5.67;
 rate.f  = 1.2;
 status  = kcxobj_itv_log_uwt( objid_ai, udata, rate );

 udata.i = 789;
 rate.i  = 1;
 status  = kcxobj_itv_log_uwt( objid_pi, udata, rate );
```

kcxobj_alm_log_10uwt 関数

機能: 警報履歴記録データを追加作成する。

書式: status = kcxobj_alm_log_10uwt(objid, ev_key, ev_stat, udata)

引数:			
	int	objid	オブジェクト ID
	int	ev_key	イベント・キー番号 (何が)
	int	ev_stat	イベント・ステータス (どうした)
	KcxIntFlt_t	*udata	記録値 (記録したい詳細データを格納)
			※udata のメモリ配列を 10 固定としてください
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: 記録データを1レコード、システムの警報履歴ファイル(sys_almYYMM.log)に追加します。
(追加は警報監視許可(kcxobj_alm_perm_iwt 関数参考)されている場合に有効)
記録するデータは、int 型の ev_key、ev_stat と共用構造体の udata です。
udata は、構造体(KcxIntFlt_t udata[10])であり、long long int 型で 10 個、double 型で 10 個、char 型で 80 個の 80 バイト分のデータを格納することができます。
生成した警報履歴ファイルは、アプリケーションプログラムから読み込んで警報の集計等に利用することもできます。

本関数に関連する設定画面
1. ポイント登録/属性設定画面→履歴書式キー番号(Ev_KEY)
2. 履歴コード文字変換テーブル画面

本関数で生成されるファイルに関連する情報
1. 付録 J「警報履歴ファイルのフォーマット」参照

使用例:

```
int           objid,ev_key,ev_stat,status;
KcxIntFlt_t   udata10[10];

  objid = kcxobj_open( "di001"  );

  kcxobj_alm_perm_iwt( objid, 1 ); /*警報許可を有効にしておく必要がある*/

  memset((void *)udata10,0,sizeof(udata10)); /*udata10 を 0 初期化*/
  ev_key       = 53;
  ev_stat      = 1;
  udata10[0].i = 123;
  udata10[1].f = 4.56;
  status       = kcxobj_alm_log_10uwt( objid, ev_key, ev_stat, udata10 );
```

kcxobj_ope_log_10uwt 関数

機能: 操作履歴記録データを追加作成する。

書式: status = kcxobj_ope_log_10uwt(objid, ev_key, ev_stat, udata)

引数:			
	int	objid	オブジェクト ID
	int	ev_key	イベント・キー番号 (何が)
	int	ev_stat	イベント・ステータス (どうした)
	KcxIntFlt_t *udata		記録値 (記録したい詳細データを格納)
			※udata のメモリ配列を 10 固定としてください
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: 記録データを1レコード、システムの操作履歴ファイル(sys_opeYYMM.log)に追加します。
記録するデータは、int 型の ev_key、ev_stat と共用構造体の udata です。
udata は、構造体(KcxIntFlt_t udata[10])であり、long long int 型で 10 個、double 型で 10 個、char 型で 80 個の 80 バイト分のデータを格納することができます。
生成した操作履歴ファイルは、アプリケーションプログラムから読み込んで操作の集計等に利用することもできます。

※出力型オブジェクトの場合、本関数によりオペレータコードも同時に履歴レコードに記録されます。このコードデータは kcxobj_operator_set 関数で設定したものが使われます。関数実行の順番にご注意ください。

本関数に関連する設定画面
1. ポイント登録/属性設定画面→履歴書式キー番号(Ev_KEY)
2. 履歴コード文字変換テーブル画面

本関数で生成されるファイルに関連する情報
1. 付録 J「操作履歴ファイルのフォーマット」参照

使用例:

```
int          objid,ev_key,ev_stat,status;
KcxIntFlt_t  udata10[10];

  objid       = kcxobj_open( "di001" );

  memset((void *)udata10,0,sizeof(udata10)); /*udata10 を 0 初期化*/
  ev_key      = 2;
  ev_stat     = 1;
  udata10[0].i = 123;
  udata10[1].f = 4.56;
  status      = kcxobj_ope_log_10uwt( objid, ev_key, ev_stat, udata10 );
```

kcxstr_text_to_udata10 関数

機能: 文字列を履歴記録用データバッファへ複写する。

書式: status = kcxstr_text_to_udata10(mode, text1, text2, udata)

引数:	int	mode	書き込みモード (0:text1 全書込、2:text1&text2 書込)
	char	*text1	書き込み文字列 1 (終端 0 デリミタされていること)
	char	*text2	書き込み文字列 2 (終端 0 デリミタされていること)
	KcxIntFlt_t	*udata	履歴記録用被データ領域
			※udata のメモリ配列を 10 固定としてください
返値:	int	status	複写した文字数

解説: mode に 0 を選択した場合、text1 から 80 バイト以下(デリミタ迄)の文字列が udata[0-9] へコピーされます。データが 80 バイトに満たない場合、余り領域(udata)は 0 パディングされます。
mode に 2 を選択した場合、text1 から 40 バイト以下(デリミタ迄)の文字列が udata[0],udata[1],udata[2],udata[3],udata[4] へ、
text2 からも 40 バイト以内(デリミタ迄)の文字列が udata[5],udata[6],udata[7],udata[8],udata[9] へコピーされます。
40 バイトに満たない部分の余りには供に 0 でパディングされます。

使用例:

```
char         text1[BUFSIZ];
char         text2[BUFSIZ];
KcxIntFlt_t  udata[10];

  strcpy( text1, "snd= tarou@domain.nippon" );
  kcxstr_text_to_udata10( 0, text1, "", udata );

  strcpy( text1, "255.255.255.255" );
  strcpy( text2, "error-ip="       );

  kcxstr_text_to_udata10( 2, text1, text2, udata );
```

kcxobj_sndstat_fromkcx 関数

機能: KaracrixBuilder から出力(操作)オブジェクトデータを得る。

書式: status = kcxobj_sndstat_fromkcx(objid, udata)

引数:	int	*objid	オブジェクト ID
	KcxIntFlt_t	*udata	出力値(操作データ)
返値:	int	status	KcxNULL: 操作データが無かった事を示します。
			KcxINTEGER: 整数型オブジェクトの操作データを取得した事を示します。
			KcxFLOAT: 実数型オブジェクトの操作データを取得した事を示します。

解説: 操作ダイアログやアプリケーションプログラムからの関数呼び出し
(kcxobj_sndistat_tokcx 関数、kcxobj_sndfstat_tokcx 関数)
によってコマンドキュー(送信用 FIFO メモリ)に入れられたオブジェクト操作データを、この関数によって先頭(FIFO:FirstInFirstOut)から一つずつ取得します。

以下の解説に当たり、kcxobj_sndstatope_fromkcx 関数の解説を先にお読み下さい。
KaracrixBuilder には出力型オブジェクトにオペレータという属性が存在しています。
kcxobj_sndstatope_fromkcx 関数では、プログラムは操作データをもとにオブジェクトの実対象(アクチュエータ等)を操作し、かつ正常を確認した後に、kcxobj_operator_set 関数を用いてオペレータ属性を更新しています。

本 kcxobj_sndstat_fromkcx 関数は、プログラム側でオペレータ属性を更新するのではなく、関数を呼んだ時点で内部でオペレータを確定し属性を更新する仕様です。実対象の応答を待って設定していないことにご注意ください。

使用例:

```
int          objid,ival,status;
double       fval;
KcxIntFlt_t  udata;

  status = kcxobj_sndstat_fromkcx( &objid, &udata );

  switch( status ){
  case KcxINTEGER:
       ival =  udata.i;
       break;
  case KcxFLOAT:
       fval =  udata.f;
       break;
  case KcxNULL:
       break;
  }
```

kcxobj_sndstatope_fromkcx 関数

機能： KaracrixBuilder システムから出力(操作)オブジェクトデータとそのオペレータ(操作者)を得る。

書式： status = kcxobj_sndstatope_fromkcx(objid, udata, operator)

引数：	int	objid	オブジェクト ID	
	KcxIntFlt_t	*udata	出力値(操作データ)	
	int	*operator	オペレータコード	
返値：	int	status	KcxNULL:	操作データが無かった事を示します。
			KcxINTEGER:	整数型オブジェクトの操作データを取得した事を示します。
			KcxFLOAT:	実数型オブジェクトの操作データを取得した事を示します。

解説： 操作ダイアログやアプリケーションプログラムからの関数呼び出し
(kcxobj_sndistat_tokcx 関数、kcxobj_sndfstat_tokcx 関数)
によってコマンドキュー(送信用 FIFO メモリ)に入れられたオブジェクト操作データを、この関数によって先頭(FIFO:FirstInFirstOut)から一つずつ取得します。

KaracrixBuilder には出力型オブジェクトにオペレータという属性が存在します。これは、オブジェクトを最後に操作した者(kcxobj_operator_get 関数)の識別属性で、プログラムは操作データをもとにオブジェクトの実対象(アクチュエータ等)を操作後、この属性を更新させる必要が有ります。kcxobj_operator_set 関数を用いて行います。

使用例:

```
int          objid;
int          ival;
double       fval;
KcxIntFlt_t  udata;
int          operator;
int          status;
int          ctl_success;

  status = kcxobj_sndstatope_fromkcx( &objid, &udata, &operator );

  switch( status ){
  case KcxINTEGER:
       ival = udata.i;
       /* 取得したデータをもとにオブジェクトの実対象(アクチュエータ等)  */
       /* を処理するプログラムをここに記述します。 そして、実対象が正常 */
       /* に操作出来た場合、kcxobj_operator_set()関数を実行します。      */
       /* ctl_success = 1 (成功)                                        */
       /* ctl_success = 0 (失敗)                                        */
       if( ctl_success == 1 ){
         kcxobj_operator_set( objid, operator, (char *)NULL );
       }
       break;
  case KcxFLOAT:
       fval = udata.f;
       /* 取得したデータをもとにオブジェクトの実対象(アクチュエータ等)  */
       /* を処理するプログラムをここに記述します。 そして、実対象が正常 */
       /* に操作出来た場合、kcxobj_operator_set()関数を実行します。      */
       /* ctl_success = 1 (成功)                                        */
       /* ctl_success = 0 (失敗)                                        */
       if( ctl_success == 1 ){
         kcxobj_operator_set( objid, operator, (char *)NULL );
       }
       break;
  case KcxNULL:
       break;
  }
```

kcxobj_sndistat_tokcx 関数

機能: KaracrixBuilder へ、整数型オブジェクトデータを送る。

書式: status = kcxobj_sndistat_tokcx(objid, idata)

引数: int objid オブジェクト ID

int idata 出力値(操作データ)

返値: int status 正常動作:0 (エラー時:負値)

解説: 整数型オブジェクトの操作データをコマンドキュー(送信用 FIFO メモリ)に入れる時に使用します。
データに付加されるオペレータ属性は「ユーザプログラム操作」となります。
本関数によって設定されたデータは下記関数より取り出されます。
kcxobj_sndstat_fromkcx 関数
kcxobj_sndstatope_fromkcx 関数
この関数が使えるのは、objvaltype が KcxOBJtp_INTEGER のものに限ります。

※コマンドキューの数は、KaracrixBuilder24A, KaracrixBuilder50B の場合 200、KaracrixBuilder500B の場合 300 です。キューに空きが無くなった場合、最も古い操作データが 1 つ削除され本関数によるデータが追加されます。

使用例:

```
int   objid;
int   idata;
int   status;

  objid  = kcxobj_open( "do001" );

  idata  = 1;  /*コマンドキューに値を入れる*/
  status = kcxobj_sndistat_tokcx( objid, idata );
```

kcxobj_sndfstat_tokcx 関数

機能: KaracrixBuilder へ、実数型オブジェクトデータを送る。

書式: status = kcxobj_sndfstat_tokcx(objid, fdata)

引数: int objid オブジェクト ID

double fdata 出力値(操作データ)

返値: int status 正常動作:0 (エラー時:負値)

解説: 実数型オブジェクトの操作データをコマンドキュー(送信用 FIFO メモリ)に入れる時に使用します。
データに付加されるオペレータ属性は「ユーザプログラム操作」となります。
本関数によって設定されたデータは下記関数より取り出されます。
kcxobj_sndstat_fromkcx 関数
kcxobj_sndstatope_fromkcx 関数
この関数が使えるのは、objvaltype が KcxOBJtp_FLOAT のものに限ります。

※コマンドキューの数は、KaracrixBuilder24A, KaracrixBuilder50B の場合 200、KaracrixBuilder500B の場合 300 です。キューに空きが無くなった場合、最も古い操作データが 1 つ削除され本関数によるデータが追加されます。

使用例:

```
int     objid;
double  fdata;
int     status;

  objid  = kcxobj_open( "ai001" );

  fdata  = 1.28;  /*コマンドキューに値を入れる*/
  status = kcxobj_sndfstat_tokcx( objid, fdata );
```

kcxprg_para_data_get 関数

機能： プログラムパラメータを得る。

書式： n = kcxprg_para_data_get(mode, keyword, paradata, maxdatas)

引数：	int	mode	読取モード
	char	*keyword[]	パラメータのキーワード文字(配列)
	char	*paradata[]	キーワードに定義付けられた文字データ(配列)
	int	maxdatas	keyword[](paradata[])の最大格納(配列)数
返値：	int	n	取得したキーワード数（パラメータファイル不在時: 負値)

解説： アプリケーションプログラムにパラメータ定義がある場合、そのキーワードと定義値(文字列)を対で取得します。

読み取りは、下記モード(mode)により異なります。

1. mode が 0 の場合

   関数初回実行時に、パラメータを読み取りデータを取得しその数を返します。

   2 回目以降には、パラメータが編集変更されていない限り 0 を返します。

   パラメータ編集変更された場合には新たにデータを再取得しその数を返します。

   その後、パラメータが編集変更されていない限りまた数 0 を返し続けます。

2. mode が 1 の場合

   関数実行時毎に、パラメータを読み取りデータを取得しその数を返します。

※keyword 及び paradata 変数への実体メモリの確保と割り付けは、kcxprg_para_data_get 関数内で行っています。アプリケーション側で行う必要はありませんが、メモリの開放(free)、メモリの付け換え(再 malloc)を行なってはいけません。

※本関数は　KCX ライブラリの一部として組み込まれていますが、
$KARACRIX/sys/ssrc/kcxlib_src.c にソースプログラムとして見ることができます。
仕様を変更して使用したい場合には、このソースプログラム関数を自分のプログラム内に別名で複写し改造する等してご使用ください。

使用例:

```
#define PRGPARADATAs  (1000)  /*パラメータ登録画面の設定数以上のものが必要*/

int   i,keywords;
char *keyword [PRGPARADATAs];
char *paradata[PRGPARADATAs];
char  keyword_data[BUFSIZ];

  for(;;){
    keywords = kcxprg_para_data_get( 0, keyword, paradata, PRGPARADATAs );
    if( keywords >= 1 ){
      for(i=0;i<keywords;i++){
        if( strcmp( keyword[i], "alarm-mode" ) == 0 ){
          strcpy( keyword_data, paradata[i] );
          printf("キーワード[%s]の、データ[%s]を得ました。¥n",keyword[i],keyword_data);
        }
      }
    }
  }
```

kcxprg_debug_msg_cwt 関数

機能: プログラムのメッセージをパラメータ編集画面に表示する。

書式: status = kcxprg_debug_msg_cwt(no, text)

引数: int no 行番号(0、1)

char *text メッセージ(最大 127 文字)

返値: int status 正常(KcxOK)、エラー(KcxERR)

解説: プログラム毎のメッセージを、「制御パラメータ入力」画面→「パラメータ編集」画面の「プログラムメッセージ欄」に表示します。
1 行目は行番号 0、2 行目は行番号 1 で指定します。
主にプログラムのデバッグ時に使用されます。

使用例:

```
char  text[128];

  strcpy( text, "abc123" );

  kcxprg_debug_msg_cwt( 0, text );
```

kcxprg_run_stat_cwt 関数

機能: プログラムのメッセージを Web 画面等に表示する。

書式: kcxprg_run_stat_cwt(text)
引数: char *text 文字列 (最大 127 文字)
返値: 未定義

解説: プログラム毎のメッセージを、Web 画面及び「制御パラメータ入力」画面内の「プログラムメッセージ欄」に表示します。
プログラムの稼働状況を外部表示する時に使用します。

使用例:

```
char  text[128];

  strcpy( text, "This program is in the state of warning now." );

  kcxprg_run_stat_cwt( text );
```

kcxcns_msg_cwt 関数

機能: メインメニュー画面の情報(Status)欄にメッセージを表示する。

書式: kcxcns_msg_cwt(line, color, text)

引数: int line 表示させる行番号 (0-3)
int color 表示色番号 (0,1,2,3,11,12,13)
char *text 文字列 (最大 70 文字)

返値: 未定義

解説: メインメニュー画面の共通情報(Status)欄に、文字列を表示します。
行番号 1 は最上位行を、3 は最下位行を意味します。
行番号 0 は特別な番号で、これを指定した場合には表示が下方向にスクロールし No.1 の行位置に設定文字列が追加表示されます。
表示色は、下記に示す色が準備されています。
0 : 黒色
1 : 赤黒色 2 : 青黒色 3 : 緑黒色
11 : 赤明色 12 : 青明色 13 : 緑明色
※上記外の値は、0 の黒色とみなされます。

使用例:

```
char  msgtext[71];
int   line;
int   color;

  line  = 0;  /*0,1,2,3*/
  color = 2;  /*0,1,2,3,11,12,13*/
  strcpy( msgtext, "こんにちは" );
  kcxcns_msg_cwt( line, color, msgtext );
```

kcxtim_tsleep 関数

機能: プログラム(プロセス)を一定時間休止する。

書式: kcxtim_tsleep(idata)
引数: int idata 休止時間(マイクロ秒)
返値: 未定義

解説: プログラム内で for や while 文でループ処理をする場合、プログラムがループ処理で CPU タイムを独占してシステムに負荷を与えないよう、プログラムに WAIT を入れる必要があります。WAIT 時間には約 10 ミリ(10000 マイクロ)秒以上が必要(OS による)です。尚、休止時間が 10 ミリ秒以下の場合 Linux/Unix の構造上、機能は曖昧になりますし、またこれ以上の場合においても Linux/Unix 環境において正確でない事があります。
10000 → 10 ミリ秒(0.01 秒)
100000 → 100 ミリ秒(0.1 秒)
500000 → 500 ミリ秒(0.5 秒)

※秒単位で休止させる場合、sleep 関数をお使い下さい。

使用例:

```
int    objid;
double fdata;

  objid = kcxobj_open( "ai001" );

  while(1){

    kcxobj_stat_frd( objid, &fdata );

    /*100 ミリ秒休止する*/
    kcxtim_tsleep( 100000 );

  }
```

kcxtim_whattime 関数

機能: 現在の時刻を得る。

書式: kcxtim_whattime(jikan)
引数: struct tm *jikan tm 構造体変数のアドレス
返値: 未定義

解説: 現在時刻を取得します。
この関数は、引数に Linux/Unix の tm 構造体を使用していますが、取得される年月日時分秒は、使いやすいように加工してあります。戻り値は、以下のようになります。

```
struct tm {
  int tm_sec;       0-59[sec](但し、閏秒の調整時に 0-61 の値を取ります)
  int tm_min;       0-59[min]
  int tm_hour;      0-23[hour]
  int tm_mday;      1-31[day]
  int tm_mon;       1-12[month]
  int tm_year;      西暦年(例：2008)
  int tm_wday;      0-6(曜日)
  int tm_yday;      1-365(通算日)
  以下省略
};
```

使用例:

```
struct tm jikan;

  kcxtim_whattime( &jikan );

  if( ( jikan.tm_year == (2100) )&&( jikan.tm_mon == (1) ) ){

    fprintf( stderr, "22 世紀になりました！¥n" );
  }
```

kcxstr_printm 関数

機能: デバッグ表示エリアに文字列を表示する。

書式: status = kcxstr_printm(no, text)

引数:	int	no	表示エリア番号(0～9)
	char	*text	文字列 (最大 31 文字)
返値:	int	status	常動作:0 (エラー時;負値)

解説: 実行モニタ画面の表示エリアの指定した番号の行に文字列を表示します。
プログラム実行過程で各種変数値など表示したい場合や、デバッグなどを行なう時に使用します。

使用例:

```
int    no;
char   text[32];
int    status;

  no = 0;
  strcpy( text, "デバッグに便利です！" );
  kcxstr_printm( no, text );

  no = 1;
  sprintf( text, "%f", 3.14 );
  kcxstr_printm( no, text );
```

kcxstr_scanfm 関数

機能: プログラムに対話データを入力する。

書式: status = kcxstr_scanfm(text)
引数: char *text 文字列(最大 31 文字)
返値: int status 正常動作:0 (エラー時:負値)

解説: アプリケーションプログラム実行時に任意の場所で、データ入力を行なう場合に使用します。
プログラム実行過程で各種パラメータを変更して動作を変更したい場合や、デバッグなどを行なう時に使用します。
本関数を実行すると、実行モニタ画面の実行行がマーキングされ、INPUT ボタンを選択すると文字列入力ダイアログが表示されます。
※本関数はプログラムをデバッグモードでコンパイルした時のみに有効です。

使用例:

```
int    selval;
char   text[32];
int    status;

  kcxstr_scanfm( text );   /*例えば 0-2 の数字を入力*/

  selval = (int)strtol( text, NULL, 10 );

  switch( selval ){
  case 0:
       /*各種処理*/
       break;
  case 1:
       break;
  case 2:
       break;
  }
```

kcxstr_md5hash 関数

機能: ハッシュ値を得る。

書式: kcxstr_md5hash(input_text, hash_text)

引数: char *input_text 入力文字列データ (0 ターミネート)

char *hash_text ハッシュ値 (128 ビット[16 バイト固定バイナリデータ])

返値: 未定義

解説: 可変長の入力文字列データから、16 バイトのハッシュ値を計算します。

使用例:

```
char  text[1024];
char  hash[16];

  strcpy( text, "karacrix password is 2000" );
  kcxstr_md5hash( text, hash );
```

kcxstr_hex_xcoder 関数

機能: 2 進数バイト列と 16 進数文字列を相互に変換する。

書式: kcxstr_hex_xcoder(mode, srctext, srclen, dsttext)

引数:

int	mode	モード(0:16 進数文字列出力、1:2 進数バイト列出力)
char	*srctext	入力データ
int	srclen	入力データ長
char	*dsttext	出力データ (入力データ長の 2 倍以上の大きさが必要)

返値: 未定義

解説: mode に、0 を選択した場合、srctext の各バイトデータは、上位 4 ビットと下位 4 ビットに分けられ、それぞれ 16 進数表記である'0'-'9','a'-'f'の ascii 文字に変換されます。
なお、出力データ長は入力データ長の 2 倍になりますので注意が必要です。
mode に、1 を選択した場合、mode が 0 の場合の逆の処理が実行されます。

使用例:

```
char  src[1024];
char  dst[1024];
char  hash[16];
char  md5sumText[32+1];

  strcpy( src, "12345678" );
  kcxstr_hex_xcoder( 0, src, strlen(src), dst );

  kcxstr_md5hash( "123abc", hash );
  kcxstr_hex_xcoder( 0, hash, 16, md5sumText );
```

kcxstr_productcode_get 関数

機能: KaracrixBuilder 製品シリーズの製品コードを得る。

書式: kcxstr_productcode_get(text)

引数: char *text 製品コード文字列 (現在 3 文字/最大 63 文字)

"101" → KaracrixBuilder-24A-small

"102" → KaracrixBuilder-50B-S

"103" → KaracrixBuilder-500B-L

"104" → KaracrixBuilder-2000

返値: 未定義

解説: 使用中の KaracrixBuilder の製品コードを取得します。
製品機能の依存を考慮したアプリケーションプログラムの作成に用います。

使用例:

```
char  textcode[64];

  kcxstr_productcode_get( textcode );

  if( strcmp( textcode, "101" ) == 0 ) printf("This is KaracrixBuilder-24A ¥n");
  if( strcmp( textcode, "102" ) == 0 ) printf("This is KaracrixBuilder-50B ¥n");
  if( strcmp( textcode, "103" ) == 0 ) printf("This is KaracrixBuilder-500B ¥n");
  if( strcmp( textcode, "104" ) == 0 ) printf("This is KaracrixBuilder-2000 ¥n");
```

kcxrep_init 関数

機能: KCX 印刷ライブラリの初期設定をする。

書式: kcxrep_init()　　[旧名称:kcxpris_priinit]
引数: なし
返値: 未定義

解説: 本関数は、印刷ライブラリ使用時に最初に記述する必要があります。
印刷ライブラリの初期設定を行ない新しい印刷ページを用意します。

使用例:

```
#include <karacrix.h>

main( argc, argv )
int    argc;
char  *argv[];
{

    kcxinit( argc, argv );

    kcxrep_init();

}
```

kcxrep_formread 関数

機能: 帳票フォームを読み込む。

書式: kcxrep_formread(formid_name) [旧名称:kcxpris_formread]
引数: char *formid_name 帳票フォームの書式コード（名称省略可）
返値: 未定義

解説: 書式コード指定された帳票フォームのデータをページに読込みます。
書式コード名を省略した場合には、帳票プログラム登録画面でフォーマット選択した(デフォルト)取込書式を指定したものとみなされます。
本関数を複数コールした場合、読込まれた帳票フォームはオーバーレイされて印刷されます。

使用例:

```
kcxrep_formread( "" );          /*デフォルトの取込書式が使用されます*/
kcxrep_formread( "form2" );   /*"form2"書式を使用します*/
```

kcxrep_drawtext 関数

機能: 文字列を帳票フォーム上に印字する。

書式: kcxrep_drawtext(orgin,x,y,position,text) ［旧名称:kcxpris_drawtext］

引数:

struct	KcxPDrPriOrgnSX	*origin	原点名 (struct KcxPDrPriOrgnSX はシステム内部使用)
int	x,y		点原点の場合、xy 供に 0 を使用します。 表原点の場合、座標位置(0 起点)を使用します。
char	*position		出力位置("R":原点右側,"L":原点左側)
char	*text		印字する文字列

返値: 未定義

解説: 文字以外の数値を印字する場合には数値を予め文字列(text)に変換しておきます。
text の内容は、帳票フォームに定義されている origin の位置に印字出力されます。
原点名は、帳票 CAD で登録したものを使用します。
原点名は、CAD 内で宣言されておりアプリケーションプログラム内で宣言する必要はありません。
原点名はポインタ記述する必要があるため、変数の頭にアドレスを示す&を置いてください。

使用例:

```
char   text[128];

  sprintf( text, "データ= %5.2f", 3.14 );

  /*点原点表原点の場合*/
  kcxrep_drawtext( &p0, 0,0, "R", text );  /* 原点 p0(0,0) の右方向に印字出力します */
  kcxrep_drawtext( &p0, 0,0, "L", text );  /* 原点 p0(0,0) の左方向に印字出力します */

  /*表原点の場合*/
  kcxrep_drawtext( &p1, 1,0, "L", text );  /* 原点 p1(1,0) の左方向に印字出力します */
  kcxrep_drawtext( &p1, 2,0, "R", text );  /* 原点 p1(2,0) の右方向に印字出力します */
  kcxrep_drawtext( &p1, 3,0, "R", text );  /* 原点 p1(3,0) の右方向に印字出力します */

  /*表原点の場合*/
  kcxrep_drawtext( &p2, 0,1, "R", text );  /* 原点 p2(0,1) の右方向に印字出力します */
  kcxrep_drawtext( &p2, 1,3, "L", text );  /* 原点 p2(1,3) の左方向に印字出力します */
  kcxrep_drawtext( &p2, 5,7, "R", text );  /* 原点 p2(5,7) の右方向に印字出力します */
```

kcxrep_newpage 関数

機能: 改ページをする。

書式: kcxrep_newpage()　[旧名称:kcxpris_newpage]
引数: なし
返値: 未定義

解説: 現在印字しているページを改ページします。
またページを1つの帳票印字ファイルにも出力します。

使用例:

```
#include <karacrix.h>

main( argc, argv )
int   argc;
char *argv[];
{
  int  i;

    kcxinit( argc, argv );

    kcxrep_init();

    for( i = 0; i < 2; i++ ){

      /*
       * 各種描画処理
       */

      kcxrep_newpage();
    }

    kcxrep_output();

}
```

kcxrep_output 関数

機能: 帳票をプリンタに出力する。

書式: kcxrep_output()　　[旧名称:kcxpris_output]
引数: なし
返値: 未定義

解説: 帳票印字出力ファイルをプリンタ(スプール)に送ります。

使用例:

```
#include <karacrix.h>

main( argc, argv )
int    argc;
char  *argv[];
{

    kcxinit( argc, argv );

    kcxrep_init();

   /*
    * 各種描画処理
    */

    kcxrep_output();

}
```

kcxobj_rep_listform_get 関数

機能: 日月報処理方法を得る。

書式: status = kcxobj_rep_listform_get(objid, idata)

引数: int objid オブジェクト ID

int *idata 処理方法値

返値: int status 正常動作:0 (エラー時:負値)

解説: ポイント登録画面の属性値「日月報処理方法」を取得します。
帳票プログラムに処理のオプションを属性で与えることを想定しています。
※本関数で取得した値の使用方法は、アプリケーションに一任されます。

使用例:

```
int  objid;
int  idata;

  objid = kcxobj_open( "pi001" );

  kcxobj_rep_listform_get( objid, &idata );
```

kcximg_quality_set 関数

機能: 画像を記録する時の画質(JPEG 圧縮クウォリティ)を設定する。

書式: kcximg_quality_set(idata)
引数: int idata クウォリティ値(1～100)
返値: 未定義

解説: アプリケーションプログラムで自動画像記録を行なう場合に画像記録の画質を変更する時に使用します。
具体的には、JPEG 圧縮ライブラリへのクウォリティパラメータを変更します。
クウォリティ値は、100 が最も高画質になり、0 が最低画質になります。クウォリティ値を高く設定する程、作成される JPEG 画像ファイルのサイズは大きくなります。

使用例:

```
int  jyoken;  /*画質モード(明暗に連動させ動的に条件値変更させる)*/

  kcximg_quality_set( 70 );

  switch( jyoken ){
  case 0:
       /*通常*/
       kcximg_quality_set( 70 );
       break;
  case 1:
       /*低画質モード*/
       kcximg_quality_set( 10 );
       break;
  case 2:
       /*高画質モード*/
       kcximg_quality_set( 100 );
       break;
  }
```

kcximg_recctl_set 関数

機能: 画像を自動記録する時の動作モードを設定する。

書式: kcximg_recctl_set(idata)

引数: int idata 動作モード
0: 連続記録を停止する
1: 連続記録を開始する
2: 1 回記録する(記録開始まで 0～1 秒の遅れが生じます)

返値: 未定義

解説: アプリケーションプログラムで自動画像記録を行なうための動作モードを設定します。
動作モード 0、1 では、本関数を実行した時点から連続的に画像記録が開始、終了します。
画像サンプリング間隔は、画像記録条件設定画面での設定によります。
動作モード 2 では、本関数を実行した時点の入力画像が 1 回のみ記録されます。
センサの状態変化を監視して画像記録を行なったり、定時観測などの用途に便利な機能です。

使用例:

```
int  sensorval;  /*パラメータ*/

  kcximg_quality_set( 70 );     /*クウォリティ値をセット*/

  kcximg_recctl_set( 1 );       /*連続記録開始*/
  ・・・
  kcximg_recctl_set( 0 );       /*連続記録終了*/

  if( sensorval > 0 ){
    /*1 回(ワンショット)記録モードで記録*/
    kcximg_recctl_set( 2 );
  }
```

kcxsio_interface_init 関数

機能: シリアルデバイスの初期設定をする。

書式: kcxsio_interface_init(devid, rate, frame, parity, stopbit)

引数:

	int	devid	通信デバイスファイル記述子
	int	rate	通信速度 (bps: 1200、2400、4800、9600、19200 など)
	int	frame	通信フレーム (bit: 7、8)
	int	parity	パリティ (無し: 0、有り: 1)
	int	stopbit	ストップビット (bit: 1、2)

返値: 未定義

解説: open 関数で取得した通信デバイスの初期化を行います。

使用例:

```
int  rs232c;

  rs232c = open( "/dev/tty00", O_RDWR );

  /*
   * 通信ボーレート: 4800 bps
   * 通信フレーム:   8 bit
   * パリティ:       無し
   * ストップビット: 1 bit
   */
  kcxsio_interface_init( rs232c, 4800, 8, 0, 1 );
```

kcxsio_recvfd 関数

機能: シリアルデバイスからデータを受信する。

書式: len = kcxsio_recvfd(devid, rcvbuff, rcvlen, cterm, timeout)

引数:	int	devid	通信デバイスのファイル記述子
	char	*rcvbuff	受信データバッファ
	int	rcvlen	指定受信データ長(byte) [受信データバッファ以内]
	int	cterm	デリミタ文字(1byte 文字)
	int	timeout	タイムアウト時間(秒)
返値:	int	len	受信データ数(byte) (エラー時:負値)

解説: RS232C 等のデバイスから、データを受信する時に使用します。
rcvbuff は、受信したデータを格納するバッファの先頭アドレスを指定します。
受信を終了させる条件として、
(1) 受信データが、指定受信データ長に達した
(2) 指定したデリミタ文字データを受信した
(3) (1),(2)に適合しないまま、タイムアウト時間を経過した
の3条件があります。
rcvlen は、通常受信データバッファの大きさを指定します。
cterm は、復帰コードとしての 13 か、改行コードとしての 10 を指定する場合が多いです。
タイムアウトを起こさせたくない場合には、timeout に 0 を指定します。

使用例:

```
#define  LF    (10)

  int   rs232c;
  int   rcvlen;
  char  rcvbuff[BUFSIZ];

    rcvlen = kcxsio_recvfd( rs232c, rcvbuff, BUFSIZ, LF, 0 );
```

kcxsio_sendfd 関数

機能: シリアルデバイスへデータを送信する。

書式: len = kcxsio_sendfd(devid, sndbuff, sndlen)

引数:	int	devid	通信デバイスのファイル記述子
	char	*sndbuff	送信データバッファ
	int	sndlen	送信データ長(byte)
返値:	int	len	送信データ数(byte) (エラー時:負値)

解説: RS232C 等のデバイスへ、データを送信する時に使用します。
sndbuff には、送信するデータを格納したバッファの先頭アドレスを指定します。
sndlen には、送信するデータ長を指定します。

使用例:

```
#define  CR    (13)

  int   rs232c;
  int   sndlen;
  char  sndbuff[BUFSIZ];

    sndbuff[0] = 'a';
    sndbuff[1] = 'i';
    sndbuff[2] = 'n';
    sndbuff[3] = CR;

    sndlen = kcxsio_sendfd( rs232c, sndbuff, 4 );
```

kcxudp_smpl_server_init 関数

機能: UDP 通信のサーバ側の初期設定をする。(UDP サーバ使用)

書式: status = kcxudp_smpl_server_init(kcxudp,port)

引数:	KcxUdpInfoSX	*kcxudp	通信情報格納構造体
	int	port	受信待ちポート番号 (/etc/services 参考)
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: UDP 通信を行なうポート番号の設定をします。

関数の内部処理(エラー処理等除く):

```
kcxudp->sockid = socket( AF_INET,SOCK_DGRAM,0 );
kcxudp->sockaddr.sin_family = AF_INET;
kcxudp->sockaddr.sin_port   = htons( kcxudp->myport = port );
kcxudp->sockaddr.sin_addr.s_addr = htonl( INADDR_ANY );
bind( kcxudp->sockid,&kcxudp->sockaddr,sizeof(struct sockaddr) );
```

使用例:

```
KcxUdpInfoSX  kcxudp;

  /*受信ポート番号を 10000 とした場合*/
  kcxudp_smpl_server_init( &kcxudp, 10000 );
```

kcxudp_smpl_client_init 関数

機能: UDP 通信のクライアント側の初期設定をする。(UDP クライアント使用)

書式: status = kcxudp_smpl_client_init(kcxudp,svrhostname,svrport,myport)

引数:			
	KcxUdpInfoSX	*kcxudp	通信情報格納構造体
	char	*svrhostname	通信したい UDP サーバのホスト名(IP アドレス記述可)
	int	svrport	通信したい UDP サーバの受信待ちポート番号
	int	myport	クライアント側のサーバからの応答受信待ちポート番号
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: UDP 通信を行なうホスト名とポート番号の設定をします。
関数の内部処理(エラー処理等除く):

```
kcxudp->myport     = myport;
kcxudp->sendtoport = svrport;
strcpy( kcxudp->svrhostname,svrhostname );
kcxudp->sockid = socket( AF_INET,SOCK_DGRAM,0 );
kcxudp->sockaddrTo.sin_addr.s_addr = (int)inet_addr( kcxudp->svrhostname );
if( kcxudp->sockaddrTo.sin_addr.s_addr != (-1) ){
  kcxudp->sockaddrTo.sin_family = AF_INET;
}else{
  hentp = gethostbyname( kcxudp->svrhostname );
  kcxudp->sockaddrTo.sin_family = hentp->h_addrtype;
  memcpy( &kcxudp->sockaddrTo.sin_addr,hentp->h_addr,hentp->h_length );
}
kcxudp->sockaddrTo.sin_port = htons( kcxudp->sendtoport );
kcxudp->sockaddr.sin_family = AF_INET;
kcxudp->sockaddr.sin_port   = htons( kcxudp->myport );
bind( kcxudp->sockid,&kcxudp->sockaddr,sizeof(struct sockaddr) );
```

使用例:

```
KcxUdpInfoSX kcxudp1;
KcxUdpInfoSX kcxudp2;

  /*サーバのポート番号 10000、クライアントの受信待ちポート番号 10001*/
  /*サーバを URL(例 : test.karacrix.jp)で指定する場合*/
  kcxudp_smpl_client_init( &kcxudp1,"test.karacrix.jp", 10000, 10001 );

  /*サーバを IP アドレス(例 : 192.168.0.100)で指定する場合*/
  kcxudp_smpl_client_init( &kcxudp2, "192.168.0.100", 10000, 10001 );
```

kcxudp_smpl_sendto 関数

機能: データを UDP サーバに送信する。(UDP クライアント使用)

書式: len = kcxudp_smpl_sendto(kcxudp,sndbuff,sndlen)

引数:	KcxUdpInfoSX	*kcxudp	通信情報格納構造体
	void	*sndbuff	送信データバッファ
	int	sndlen	送信データバッファ長(byte)
返値:	int	len	送信データ数(byte) (エラー時:負値)

解説: データを UDP サーバへ送信します。

関数の内部処理(エラー処理等除く):

```
sendto( kcxudp->sockid,sndbuff,sndlen,0,
        &kcxudp->sockaddrTo,sizeof(struct sockaddr) );
```

使用例:

```
KcxUdpInfoSX  kcxudp;
char          sndbuff[BUFSIZ];

  kcxudp_smpl_client_init( &kcxudp, "???.???.co.jp", 10000, 10001 );

  strcpy( sndbuff, "how do you do." );

  kcxudp_smpl_sendto( &kcxudp, (void *)sndbuff, strlen(sndbuff) );
```

kcxudp_smpl_return_sendto 関数

機能: データを UDP クライアントに応答送信する。(UDP サーバ使用)

書式: len = kcxudp_smpl_return_sendto(kcxudp,sndbuff,sndlen)

引数:	KcxUdpInfoSX	*kcxudp	通信情報格納構造体
	void	*sndbuff	送信データバッファ
	int	sndlen	送信データバッファ長(byte)
返値:	int	len	送信データ数(byte) (エラー時:負値)

解説: kcxudp_smpl_block_recvfrom 関数、あるいは kcxudp_smpl_tmout_recvfrom 関数によって受信した UDP クライアントへデータを応答送信します。

関数の内部処理(エラー処理等除く):

```
sendto( kcxudp->sockid,sndbuff,sndlen,0,
        &kcxudp->sockaddrFrom,sizeof(struct sockaddr) );
```

使用例:

```
KcxUdpInfoSX  kcxudp;
char          sndbuff[BUFSIZ];

  kcxudp_smpl_server_init( &kcxudp, 10000 );

  strcpy( sndbuff, "I am fine, and you?" );

  kcxudp_smpl_return_sendto( &kcxudp, (void *)sndbuff, strlen(sndbuff) );
```

kcxudp_smpl_block_recvfrom 関数

機能: UDP データをブロック受信する。(DUP サーバクライアント両使用)

書式: len = kcxudp_smpl_block_recvfrom(kcxudp,rcvbuff,rcvlen)

引数:	KcxUdpInfoSX	*kcxudp	通信情報格納構造体
	void	*rcvbuff	受信データバッファ
	int	rcvlen	受信データバッファ長(byte)
返値:	int	len	受信データ数(byte) (エラー時:負値)

解説: UDP データを受信するまで待機します。

関数の内部処理(エラー処理等除く):

```
fromlen = sizeof( struct sockaddr );
recvfrom( kcxudp->sockid,rcvbuff,rcvlen,0,&kcxudp->sockaddrFrom,&fromlen );
```

使用例:

```
KcxUdpInfoSX  kcxudp;
int           len;
char          rcvbuff[BUFSIZ];

  len = kcxudp_smpl_block_recvfrom( &kcxudp, (void *)rcvbuff, BUFSIZ );
```

kcxudp_smpl_tmout_recvfrom 関数

機能: UDP データをタイムアウト付き受信する。(DUP サーバクライアント両使用)

書式: len = kcxudp_smpl_tmout_recvfrom(kcxudp,rcvbuff,rcvlen,timeout)

引数:			
	KcxUdpInfoSX	*kcxudp	通信情報格納構造体
	void	*rcvbuff	受信データバッファ
	int	rcvlen	受信データバッファ長(byte)
	int	timeout	タイムアウト時間(秒)
返値:	int	len	受信データ数(byte) (エラー時:負値)

解説: timeout で指定した時間を過ぎると、UDP データの受信待ちを解除します。
関数の内部処理(エラー処理等除く):
alarm(timeout);
fromlen = sizeof(struct sockaddr);
recvfrom(kp->sockid,rcvbuff,rcvlen,0,&kp->sockaddrFrom,&fromlen);

使用例:

```
KcxUdpInfoSX  kcxudp;
int           len;
char          rcvbuff[BUFSIZ];

  /*タイムアウト 10[sec]*/
  len = kcxudp_smpl_tmout_recvfrom( &kcxudp, (void *)rcvbuff, BUFSIZ, 10 );
```

kcxudp_received_packet_clean 関数

機能: UDP ポートに蓄積受信しているデータを全て削除する。

書式: n = kcxudp_received_packet_clean(kcxudp)

引数:	KcxUdpInfoSX	*kcxudp	通信情報格納構造体
返値:	int	n	削除したパケット数

解説: UDP ポートの受信済のパケットデータを全て廃棄します。
ある装置に UDP コマンドを送ってその応答データを得るシステムで、UDP コマンドを送る前に既に不要なパケットデータが UDP のポートに入っていた場合、余計な処理をしなければならなくなります。このような場合の簡単な解決方法として本関数を用いると良いかもしれません。

使用例:

```
KcxUdpInfoSX  kcxudp;
int           len;
char          sndbuff[BUFSIZ];
char          rcvbuff[BUFSIZ];

  /*受信バッファクリア*/
  kcxudp_received_packet_clean( &kcxudp );

  /*送信*/
  kcxudp_smpl_sendto( &kcxudp, (void *)sndbuff, strlen(sndbuff) );

  /*受信*/
  kcxudp_smpl_block_recvfrom( &kcxudp, (void *)rcvbuff, BUFSIZ );
```

kcxtcp_smpl_server_init 関数

機能: TCP 通信のサーバ側の初期設定をする。(TCP サーバ使用)

書式: status = kcxtcp_smpl_server_init(kcxtcp, port)

引数:	KcxTcpInfoSX	*kcxtcp	通信情報格納構造体
	int	port	受信待ちポート番号
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: TCP 通信を行なうポート番号の設定をします。

関数の内部処理(エラー処理等除く):

```
kcxtcp->sockid = socket( AF_INET,SOCK_STREAM,0 );
kcxtcp->sockaddr.sin_family = AF_INET;
kcxtcp->sockaddr.sin_port    = htons( kcxtcp->port = port );
kcxtcp->sockaddr.sin_addr.s_addr = htonl( INADDR_ANY                );
for(i=0;i<retry;i++){
 /*コネクション開放後一定時間アドレスの再使用が禁止される為のリトライ*/
   bind( kcxtcp->sockid,&kcxtcp->sockaddr,sizeof(struct sockaddr) );
   sleep(1);
}
```

使用例:

```
KcxTcpInfoSX  kcxtcp;

  kcxtcp_smpl_server_init( &kcxtcp, 10000 );
```

kcxtcp_smpl_accept_newsockid 関数

機能: TCP アクセプト(接続手続)する。(TCP サーバ使用)

書式: newsockid = kcxtcp_smpl_accept_newsockid(kcxtcp)

引数:	KcxTcpInfoSX	*kcxtcp	通信情報格納構造体
返値:	int	newsockid	新しいソケット ID(正常時:正値、エラー時:負値)

解説: クライアントからの接続を待ちます。接続されると通信用の新しいソケット ID を生成します。

関数の内部処理(エラー処理等除く):

return accept(kcxtcp->sockid,&kcxtcp->sockaddr,sizeof(struct sockaddr));

使用例:

```
KcxTcpInfoSX  kcxtcp;
int           sockid;
int           rcvlen,sndlen;
char          rcvbuff[BUFSIZ];
char          sndbuff[BUFSIZ];

  kcxtcp_smpl_server_init( &kcxtcp, 10000 );

  for(;;){
    if(( sockid = kcxtcp_smpl_accept_newsockid( &kcxtcp )) < 0 ){
      sleep(1);
      continue;
    }
    rcvlen = recv( sockid, rcvbuff, BUFSIZ, 0 );
    /*受信データから送信データを作成して送り返す場合*/
    send( sockid, sndbuff, sndlen, 0 );
    close( sockid );
  }
```

kcxtcp_smpl_client_init 関数

機能: TCP 通信のクライアント側の初期設定をする。(TCP クライアント使用)

書式: status = kcxtcp_smpl_client_init(kcxtcp,svrhostname,port)

引数:	KcxTcpInfoSX	*kcxtcp	通信情報格納構造体
	char	*svrhostname	サーバのホスト名(IP アドレス記述可)
	int	port	サーバの受信待ちポート番号
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: TCP 通信を行なうホスト名、ポート番号の設定をします。

関数の内部処理(エラー処理等除く):

```
kcxtcp->port = port;
strcpy( kcxtcp->svrhostname,svrhostname );
kcxtcp->sockaddr.sin_addr.s_addr = (int)inet_addr( kcxtcp->svrhostname );
if( kcxtcp->sockaddr.sin_addr.s_addr != (-1) ){
   kcxtcp->sockaddr.sin_family = AF_INET;
}else{
   hentp = gethostbyname( kcxtcp->svrhostname );
   kcxtcp->sockaddr.sin_family = hentp->h_addrtype;
   memcpy( &kcxtcp->sockaddr.sin_addr,hentp->h_addr,hentp->h_length );
}
kcxtcp->sockaddr.sin_port = htons( kcxtcp->port );
```

使用例:

```
KcxTcpInfoSX  kcxtcp1;
KcxTcpInfoSX  kcxtcp2;

  /*サーバの受信待ちポート番号 10000*/
  /*サーバを URL(例 : test.karacrix.co.jp)で指定する場合*/
  kcxtcp_smpl_client_init( &kcxtcp1,"test.karacrix.co.jp", 10000 );

  /*サーバを IP アドレス(例 : 192.168.0.100)で指定する場合*/
  /*(自分ホストを指定する場合の IP アドレスは、"127.0.0.1")*/
  kcxtcp_smpl_client_init( &kcxtcp2, "192.168.0.100", 10000 );
```

kcxtcp_smpl_socket_connect 関数

機能: TCP コネクト(接続手続)をする。(TCP クライアント使用)

書式: status = kcxtcp_smpl_socket_connect(kcxtcp)

引数:	KcxTcpInfoSX	*kcxtcp	通信情報格納構造体
返値:	int	status	正常動作: 0 (エラー時:負値)

解説: TCP サーバへ接続します。一度接続が確立するとクローズするまで接続を保持します。
関数の内部処理(エラー処理等除く):

```
kcxtcp->sockid = socket( AF_INET,SOCK_STREAM,0 );
connect( kcxtcp->sockid,&kcxtcp->sockaddr,sizeof(struct sockaddr) );
```

使用例:

```
KcxTcpInfoSX  kcxtcp;
int           sockid;
int           rcvlen,sndlen;
char          rcvbuff[BUFSIZ];
char          sndbuff[BUFSIZ];

  kcxtcp_smpl_client_init( &kcxtcp, "?.?.co.jp", 20000 );

  for(;;){
    if( kcxtcp_smpl_socket_connect( &kcxtcp ) < 0 ){
      sleep(1);
      continue;
    }
    strcpy( sndbuff, "how do you do." );
    send( kcxtcp.sockid, sndbuff, strlen(sndbuff), 0 );
    rcvlen = kcxtcp_smpl_tmout_recv( &kcxtcp, rcvbuff, BUFSIZ, 5 );
    close( kcxtcp.sockid );
  }
```

kcxtcp_smpl_tmout_recv 関数

機能: TCP データをタイムアウト付き受信をする。(TCP サーバクライアント両使用)

書式: len = kcxtcp_smpl_tmout_recv(kcxtcp,rcvbuff,rcvlen,timeout)

引数:	KcxTcpInfoSX	*kcxtcp	通信情報格納構造体
	void	*rcvbuff	受信データバッファ
	int	rcvlen	受信データバッファ長(byte)
	int	timeout	タイムアウト時間(秒)
返値:	int	len	受信データ数(byte) (エラー時:負値)

解説: 受信データ数が、0 の場合には、相手(サーバ或はクライアント)のプロセスがデータを送って来なかったか、終了している可能性があります。

関数の内部処理(エラー処理等除く):

```
select( fds,&rfds,NULL,NULL,(struct timeval *)&tm );
recv( kcxtcp->sockid,rcvbuff,rcvlen,0 );
```

使用例:

```
KcxTcpInfoSX  kcxtcp;
int           len;
char          rcvbuff[BUFSIZ];

  /*受信タイムアウトを 5[sec]とした場合*/
  len = kcxtcp_smpl_tmout_recv( &kcxtcp, rcvbuff, BUFSIZ, 5 );
```

kcxsnd_email_text 関数

機能: 1 行のメッセージを E メールで送信する。

書式: kcxsnd_email_text(toaddr, ccaddr, bccaddr, subject, mailtext)

引数:

char *toaddr	送り先	(アスキー(1byte)文字最大 95 文字)
char *ccaddr	同報送り先[CC]	(アスキー(1byte)文字最大 95 文字)
char *bccaddr	同報送り先[BCC]	(アスキー(1byte)文字最大 95 文字)
char *subject	メールタイトル	(アスキー(1byte)文字換算最大 127 文字)
char *mailtext	メール本文	(アスキー(1byte)文字換算最大 255 文字)

返値: 未定義

解説: 指定した E メールアドレスに、アスキー(1byte)文字換算最大 255 文字以内のメッセージ(メール本文)を送信します。

メール本文1行に使用できる漢字文字数:

漢字1字のアスキー文字換算長は、EUCコードの場合2文字(バイト)ですが、UTF8コードの場合 3～4 文字(バイト)です。UTF8 を使用する場合 255 文字(バイト)以内で扱える漢字文字数は、255/4=63 以上 255/3=85 以下で安全側を取ると 63 文字までとなります。

※メールタイトルの場合は同様に 31 文字まで。

使用例:

```
char  toaddr[96];    /*送り先*/
char  ccaddr[96];    /*同報送り先[CC]*/
char  bccaddr[96];   /*同報送り先[BCC]*/
char  subject[128];   /*メールタイトル*/
char  mailtext[256];  /*メール本文*/

  strcpy( toaddr,   "nippon@xxx.jp"                 );
  strcpy( ccaddr,   "honami@xxx.jp,syuusei@xxx.jp" );
  strcpy( bccaddr,  "tarou@xxx.jp,hanako@xxx.jp"   );
  strcpy( subject,  "ハロー"                        );
  strcpy( mailtext, "こんにちは！"                  );

  kcxsnd_email_text( toaddr, ccaddr, bccaddr, subject, mailtext );
```

kcxsnd_email_texts 関数

機能: 複数行のメッセージをＥメールで送信する。

書式: kcxsnd_email_texts(toaddr, ccaddr, bccaddr, subject, mtext, mtexts)

引数:				
	char	*toaddr	送り先	(アスキー(1byte)文字最大 95 文字)
	char	*ccaddr	同報送り先[CC]	(アスキー(1byte)文字最大 95 文字)
	char	*bccaddr	同報送り先[BCC]	(アスキー(1byte)文字最大 95 文字)
	char	*subject	メールタイトル	(アスキー(1byte)文字換算最大 127 文字)
	char	*mtext[]	メール本文	(1 行アスキー(1byte)文字換算最大 255 文字)
	int	mtexts	メール本文の行数	(最大 400)

返値: 未定義

解説: 指定したＥメールアドレスに、1 行当たりアスキー(1byte)文字換算最大 255 文字で 400 行以内のメッセージ(メール本文)を送信します。 (kcxsnd_email_text 関数解説参照)

使用例:

```
char  toaddr[96];   /*送り先*/
char  subject[128]; /*メールタイトル*/
char *mtext[400];   /*メール本文(作業メモリは malloc により確保する)*/
int   i,mtexts;     /*メール本文の行数*/

  strcpy( toaddr,  "nippon@xxx.jp" );
  strcpy( subject, "Hello!"        );

  mtexts = 2; /*mtexts 値は 400 以下*/
  for(i=0;i<mtexts;i++){
    mtext[i] = (char *)malloc( 256 );
  }
  strcpy( mtext[0], "こんにちわ！"  );
  strcpy( mtext[1], "お元気ですか！" );

  kcxsnd_email_texts( toaddr, "", "", subject, mtext, mtexts );
```

kcxsnd_email_text_append 関数

機能: 1 行のメッセージに添付情報を添えて E メールで送信する。

書式: kcxsnd_email_text_append(toaddr, ccaddr, bccaddr, subject, mailtext, appendtext)

引数:	char	*toaddr	送り先	(アスキー(1byte)文字最大 95 文字)
	char	*ccaddr	同報送り先[CC]	(アスキー(1byte)文字最大 95 文字)
	char	*bccaddr	同報送り先[BCC]	(アスキー(1byte)文字最大 95 文字)
	char	*subject	メールタイトル	(アスキー(1byte)文字換算最大 127 文字)
	char	*mailtext	メール本文	(アスキー(1byte)文字換算最大 255 文字)
	char	*appendtext	添付文	(アスキー(1byte)文字最大 95 文字)

返値: 未定義

解説: 指定した E メールアドレスに、アスキー(1byte)文字換算最大 255 文字以内のメッセージ(メール本文)を送信します。 (kcxsnd_email_text 関数解説参照)
また、監視画面あるいは計測トレンド画像を1つ添付することができます。
添付文は、以下の通りです。 (5 章「5.16 E メールによる監視・操作」参照)
mon <no> [-pansize <x><y><w><h>] [-quality <value>] [-png] [-jpg]
mtre <no> [-winmenusz <no>] [-quality <value>] [-png] [-jpg]

使用例:

```
char   toaddr[96];      /*送り先*/
char   subject[128];    /*メールタイトル*/
char   mailtext[256];   /*メール本文 */
char   appendtext[96];  /*添付文*/

  strcpy( toaddr,   "nippon@xxx.jp" );
  strcpy( subject,  "Hello!" );
  strcpy( mailtext, "こんにちは！" );

  strcpy( appendtext, "mon 1" );                /*1 番のモニタ画面を添付*/
  kcxsnd_email_text_append( toaddr,"","",subject,mailtext,appendtext );

  strcpy( appendtext, "mon 1 -quality 90" );    /*上記に、画質指定も追加*/
  kcxsnd_email_text_append( toaddr,"","",subject,mailtext,appendtext );

  strcpy( appendtext, "mtre 1" );               /*1 番のトレンドグラフ画像を添付*/
  kcxsnd_email_text_append( toaddr,"","",subject,mailtext,appendtext );
```

kcxsnd_email_texts_append 関数

機能: 複数行のメッセージに添付情報を添えてＥメールで送信する。

書式: kcxsnd_email_texts_append(toaddr, ccaddr, bccaddr, subject, mtext, mtexts, appendtext)

引数:

char *toaddr	送り先	(アスキー(1byte)文字最大 95 文字)
char *ccaddr	同報送り先[CC]	(アスキー(1byte)文字最大 95 文字)
char *bccaddr	同報送り先[BCC]	(アスキー(1byte)文字最大 95 文字)
char *subject	メールタイトル	(アスキー(1byte)文字換算最大 127 文字)
char *mtext[]	メール本文	(アスキー(1byte)文字換算最大 255 文字/1 行)
int mtexts	メール本文の行数	(最大 400)
char *appendtext	添付文	(アスキー(1byte)文字最大 95 文字)

返値: 未定義

解説: 指定したＥメールアドレスに、1 行当たりアスキー(1byte)文字換算最大 255 文字で 400 行以内のメッセージ(メール本文)を送信します。 (kcxsnd_email_text 関数解説参照)
また、監視画面あるいは計測トレンド画像を1つ添付することができます。
添付文は、以下の通りです。 (5 章「5.16 Ｅメールによる監視・操作」参照)
mon <no> [-pansize <x><y><w><h>] [-quality <value>] [-png] [-jpg]
mtre <no> [-winmenusz <no>] [-quality <value>] [-png] [-jpg]

使用例:

```
char   toaddr[96];       /*送り先*/
char   subject[128];     /*メールタイトル*/
char *mtext[400];        /*メール本文(作業メモリは malloc により確保する)*/
int    i,mtexts;         /*メール本文の行数*/
char   appendtext[96];   /*添付文*/

  strcpy( toaddr,  "nippon@xxx.jp" );
  strcpy( subject, "Hello!"        );

  mtexts = 2; /*mtexts 値は 400 以下*/
  for(i=0;i<mtexts;i++){
    mtext[i] = (char *)malloc( 256 );
  }
  strcpy( mtext[0], "こんにちわ！"   );
  strcpy( mtext[1], "お元気ですか！" );

  strcpy( appendtext, "mon 1" );              /*1 番のモニタ画面を添付*/
  kcxsnd_email_texts_append( toaddr, "", "", subject, mtext, mtexts, appendtext );

  strcpy( appendtext, "mon 1 -quality 90" );   /*上記に、画質指定も追加*/
  kcxsnd_email_texts_append( toaddr, "", "", subject, mtext, mtexts, appendtext );

  strcpy( appendtext, "mtre 1" );             /*1 番のトレンドグラフ画像を添付*/
  kcxsnd_email_texts_append( toaddr, "", "", subject, mtext, mtexts, appendtext );
```

kcxsnd_email_texts_appends 関数

機能: 複数行のメッセージに複数の添付情報を添えてＥメールで送信する。

書式: kcxsnd_email_texts_appends(toaddr, ccaddr, bccaddr,
subject, mtext, mtexts, appendtext, appendtexts)

引数:				
	char	*toaddr	送り先	(アスキー(1byte)文字最大 95 文字)
	char	*ccaddr	同報送り先[CC]	(アスキー(1byte)文字最大 95 文字)
	char	*bccaddr	同報送り先[BCC]	(アスキー(1byte)文字最大 95 文字)
	char	*subject	メールタイトル	(アスキー(1byte)文字換算最大 127 文字)
	char	*mtext[]	メール本文	(アスキー(1byte)文字換算最大 255 文字/1 行)
	int	mtexts	メール本文の行数	(最大 400)
	char	*appendtext[]	添付文	(アスキー(1byte)文字最大 95 文字/1 行)
	int	appendtexts	添付文の数	(最大 4)

返値: 未定義

解説: 指定したＥメールアドレスに、1 行当たりアスキー(1byte)文字換算最大 255 文字で 400 行以内のメッセージ(メール本文)を送信します。 (kcxsnd_email_text 関数解説参照)
また、監視画面あるいは計測トレンド画像を複数添付することができます。
添付文は、以下の通りです。 (5 章「5.16 Ｅメールによる監視・操作」参照)
mon <no> [-pansize <x><y><w><h>] [-quality <value>] [-png] [-jpg]
mtre <no> [-winmenusz <no>] [-quality <value>] [-png] [-jpg]

使用例:

```
char  toaddr[96];       /*送り先*/
char  subject[128];     /*メールタイトル*/
char *mtext[400];       /*メール本文(作業メモリは malloc により確保する)*/
int   i,mtexts;         /*メール本文の行数*/
char *apppendtext[4];   /*添付文(作業メモリは malloc により確保する)*/
int   apppendtexts;     /*添付文の数*/

  strcpy( toaddr,  "nippon@xxx.jp" );
  strcpy( subject, "Hello!"        );

  mtexts = 2; /*mtexts 値は 400 以下*/
  for(i=0;i<mtexts;i++)  mtext[i] = (char *)malloc( 256 );
  strcpy( mtext[0], "こんにちわ！"   );
  strcpy( mtext[1], "お元気ですか！" );

  apppendtexts = 3;
  for(i=0;i<apppendtexts;i++) apppendtext[i] = (char *)malloc( 96 );
  strcpy( apppendtext[0], "mon 1"             ); /*1 番のモニタ画面を添付*/
  strcpy( apppendtext[1], "mon 1 -quality 90" ); /*上記に、画質指定も追加*/
  strcpy( apppendtext[2], "mtre 1"            ); /*1 番のトレンドグラフ画像を添付*/

  kcxsnd_email_texts_appends( toaddr, "", "",
                              subject, mtext, mtexts, apppendtext, apppendtexts );
```

○メール添付文の補足説明 (V3.51 以降)

メールの添付文は、KaracrixBuilder 上で作成した監視パネルやトレンドグラフをメール本文に添えて送信する時に使用するものです。またメール本文でオブジェクトの状態や属性を表す記述を作り込まなくても良いよう、システムで作成されるオブジェクトの状態や属性を添付文を使用して送れます。添付文に書き込む書式は、5 章「5.16　E メールによる監視・操作」に準じており @kcxget を取り除いた部分が送信する書式となります。書式に関しては「5.16　E メールによる監視・操作」を参照下さい。

送信の添付文で記述できるコマンドは、以下のものとなります。

1. 監視パネル　　(mon <no> ...)
2. 計測トレンド　　(mtre <no> ...)
3. 記録トレンド　　(ftre <no> ...)
4. ファイル　　(file <file_name> ...)
5. オブジェクト選択状態 (obj <objid>)
6. オブジェクト選択属性 (etc <objid>)

例) コマンド記述例

```
strcpy( apppendtext[0～7], "mon 1" );                    //監視パネル送信例
strcpy( apppendtext[0～7], "mtre 2" );                   //計測トレンドグラフ送信例
strcpy( apppendtext[0～7], "ftre 3" );                   //記録トレンドグラフ送信例
strcpy( apppendtext[0～7], "mon 1 -quality 90" );        //監視パネル送信例
strcpy( apppendtext[0～7], "file  abc.dat  /etc/hosts" ); //ファイルの送信例
strcpy( apppendtext[0～7], "obj  di001" );               //di001 の状態送信例
strcpy( apppendtext[0～7], "etc  di001" );               //di001 の属性送信例
strcpy( apppendtext[0～7], "obj  ai001" );               //ai001 アナログの状態送信例
strcpy( apppendtext[0～7], "obj  img001" );              //img001 カメラ画像の送信例
```

例) KaracrixBuilder によって内部作成され、添付ファイルを説明した本文例

```
##CpuTime 2015/11/01 07:00:04
#[Object Status]
# di001 (DI) stat=[OFF] alm=0 Nm=[接点入力 1]
# do001 (DO) stat=[ ON] alm=0 ctl=[リモート] Nm=[リレー出力１]
# ai001 (AI) stat=[   12.79 (-)] alm=0 Nm=[アナログ入力１]
# img001 (IMG) image=[t2015110307000409_img001.jpg] Nm=[イメージ入力１]
#[Attached Files Info]
# 1. mon 1) title= ハウス filename= t2015110307000611_mon1.png
# 2. mtre 7) title= 温度  filename= t2015110307000510_mtre7.png
# 3. ftre 9) title= 外気  filename= t2015110307000611_ftre9.png
```