IEBus™ ダミーボード・マクロ言語対応

AP-ALDM2
ユーザーズ・マニュアル

株式会社アプリケーション

〒194-0013　町田市原町田 5-8-18 ｷｬｽﾃｨﾝｸﾞ町田ﾋﾞﾙ402号

TEL．042-732-1377　FAX．042-732-1378

http://www.apply.co.jp/

第1.0版　2003.03.25

目　次

第１章　はじめに

本装置は、パーソナルコンピュータ（以下パソコン）を利用して IEBus ユニットのダミー装置として、お客様が開発した装置の IEBus 通信部をデバックするための開発支援装置です。

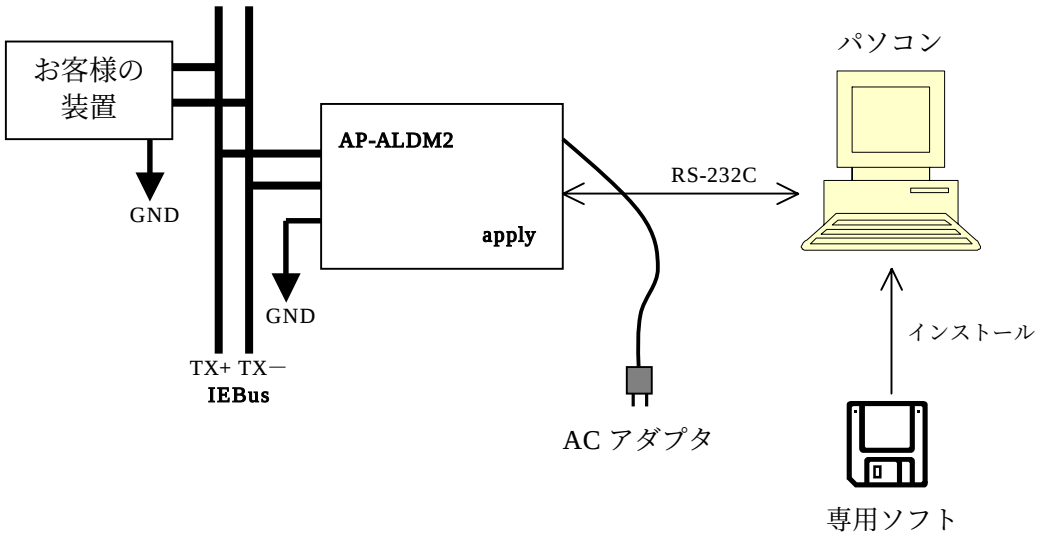
本装置によってIEBus通信をパソコンのRS-232Cポートを使用して送受信する事が出来ます。Windows 上で動作するマクロ言語（AP-Macro）によって、より複雑なプロトコルに対応する事が可能になります。

また、IEBus に特化したマクロ言語を新たに開発することによって、安易に通信プログラムが作成出来る様になりました。IEBus の通信先が開発中の場合や、動作テストなどにご使用下さい。

また、IEBus 上に流れる全てのデータをモニタする場合は弊社製品の「AP-IEB2」をご利用下さい。

第２章　構成

構成図



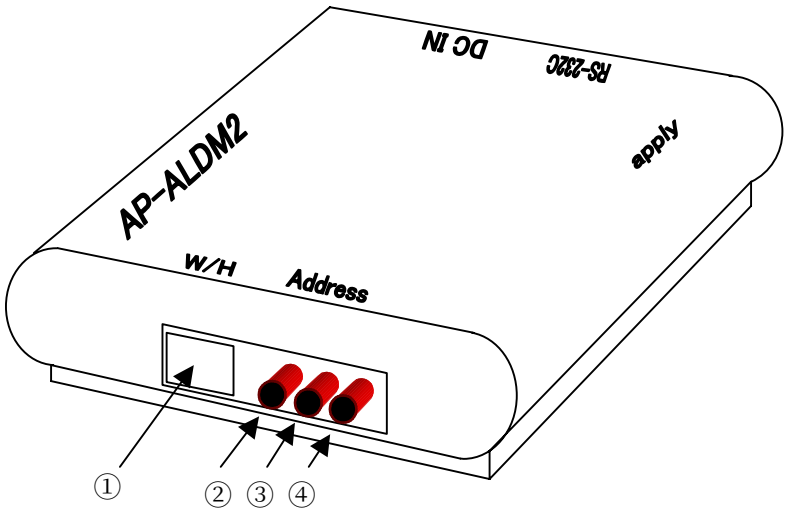
必要機器

パソコン	Windows95/98/Me/2000/XP が動作し RS-232C ポートが１チャンネル以上あること。
RS-232C ケーブル	9pinD-Sub メスコネクタ（本装置側）とパソコン側コネクタを持つクロスケーブル
お客様の装置	IEbus 接続して通信を行う装置

本製品に含まれる物

IEBus ダミーボード	IEBus の通信と RS-232C を変換する装置
専用ワイヤーハーネス	ミノムシクリップ付きと無しの２種類
AC アダプタ	+12V 出力
専用ソフト(FD)	本装置をコントロールする Windows95/98/Me/2000/XP 用アプリケーション・ソフト
マニュアル	

第３章　各部の説明



① ワイヤーハーネス接続コネクタ
　付属のミノムシクリップ付き／無しのワイヤーハーネスを接続します。

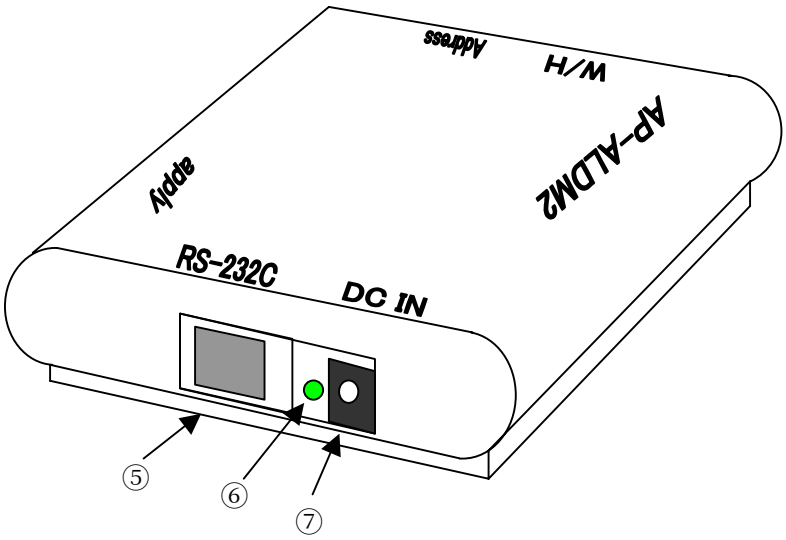
緑色ミノムシクリップ	TX＋	(1pin)	
青色ミノムシクリップ	TX－	(2pin)	
黒色ミノムシクリップ	GND	(3pin)	
ナシ	+12V	(4pin)	この+12V は[DC IN]に付属 AC アダプタより電源を供給できないときに、ここから電源を供給してください。(+12V と[DC IN]は導通しています)

② 自局アドレスH(bit 11～8)設定用ロータリスイッチ
③ 自局アドレスM(bit 7～4)設定用ロータリスイッチ
④ 自局アドレスL(bit 3～0)設定用ロータリスイッチ

自局アドレスを 123h に設定する場合は

自局アドレスH	1
自局アドレスM	2
自局アドレスL	3

に設定します。



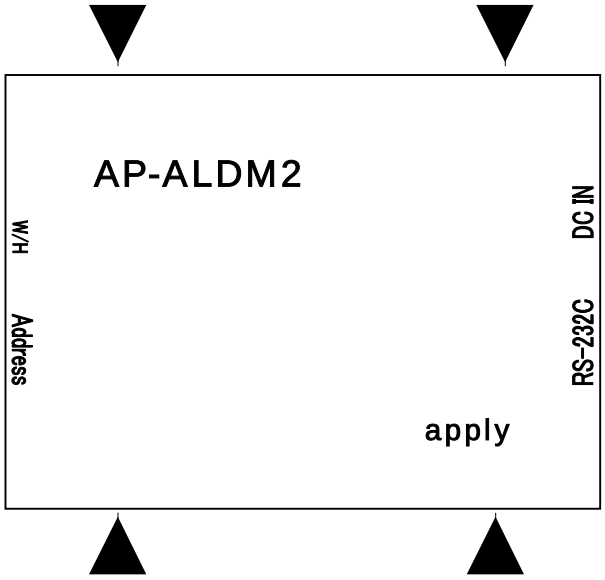
⑤ RS-232C コネクタ
　パソコン側の COM1 か COM2 へ RS-232C のクロスケーブルで接続してください。
⑥ POWER ランプ
　電源がＯＮの時に点灯します。
⑦ DC IN コネクタ
　付属の AC アダプタを接続します。

第４章　終端抵抗の設置

本装置内には 68Ω　1/8W の抵抗が実装されています。この抵抗を終端抵抗として使用する場合には、本体をあけてジャンパショートピン JP1 をショートしてください。

本体の開け方

下記のように本体下部の４ヶ所を押さえて上部を上げてください。



第５章　ソフトウェア

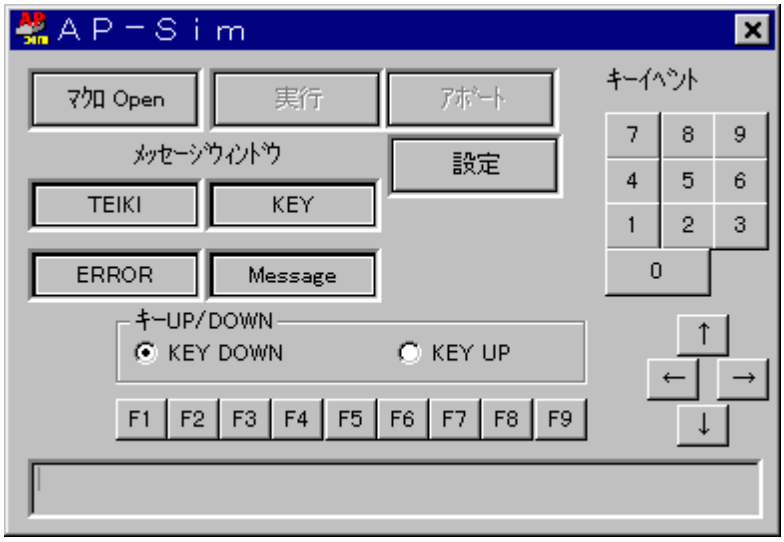
5.1.　インストール

添付 FD をパソコンに入れて setup.exe を実行してください。
表示されるメッセージに従い、インストールするホルダ等を指定してください。

5.2.　実行

正常にインストールされますと、ApSim のグループができますので、ここから apsim.exe を実行します。

起動時の画面

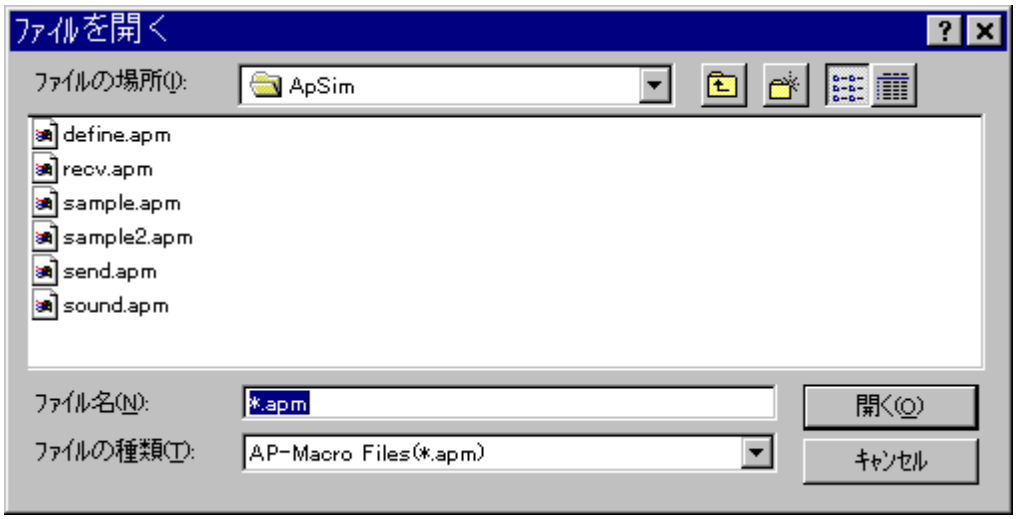


5.2.1　［設定］ボタン

AP-ALDM2 を接続している COM ボートを設定できます。

5.2.2　［マクロ Open］ボタン

実行するマクロファイルを指定します。



ロード後の画面

5.2.3　［実行］ボタン

ロードされたマクロを実行します。

5.2.4　［アボート］ボタン

実行中のマクロを強制終了します。

5.2.5　［TEIKI］ボタン

TEIKE メッセージウィンドウを表示します。

定期送信処理の実行内容を最初の 1 命令だけ表示します。

5.2.6 ［KEY］ボタン

KEY メッセージウィンドウを表示します。
マクロ中で定義されたキー入力が発生した場合に、実行内容を最初の 1 命令だけ表示します。

5.2.7 ［ERROR］ボタン

ERROR メッセージウィンドウを表示します。
マクロ実行中、エラーが発生するとウィンドウにエラーコード、
エラーラインを表示します。

5.2.8 ［Message］ボタン

MAIN メッセージウィンドウを表示します。
ECHO 文（マクロ言語仕様参照）を使用したメッセージ等を表示します。

5.2.9 キーUP/DOWN

ウィンドウ上に表示されているキーイベント用ボタン（0～9、矢印、F1～F9）をクリックした時のイベントを設定します。
[KEY DOWN]に設定すると、各ボタンのクリックを「キーを押した」として処理します。
[KEY UP]に設定すると、各ボタンのクリックを「キーを離した」として処理します。
実際にキーボードを操作する場合は、ここの設定には影響されません。ボタンをマウスでクリックした時だけの設定です。

5.3. アンインストール

コントロールパネルの「アプリケーションの追加と削除」から ApSim を選び、削除できます。

第６章　マクロ言語仕様

AP-Macroの言語仕様を説明します. AP-Macroでは１行に１命令を記述し, AP-ALDM2ではこれを１行ずつインタプリタ的に解釈し，実行していきます.

６．１．　コメント

AP-Macro では'#'文字以降はすべてコメントとなります．これは，行のはじめに'#'文字があった場合は行全体がコメントとして扱われ，行の途中から'#'文字があった場合はその部分からがコメントとして扱われ，実行時には無視されます.

例：
```
# フレームの送信
SEND frame0
RECV frame1   # 送信したフレームに対するリプライを受信する
```

６．２．　変数の型と宣言

AP-Macro では変数が使用できます. 型には **IDATA** 型, **INT** 型, **IFRAME** 型, **IFMASK** 型があります.
変数の１文字目は必ずアルファベットの小文字で記述しなければなりません．また，変数に使える文字はアルファベットとアンダーラインと数字のみで, 最大文字数は 255 文字です.
AP-Macro では，マクロ記述ファイル内で使用する変数の宣言をまとめて行い，プロシージャ（後述）の宣言がそれに続き，その後に処理の記述を行う，という順序で記述されていなければなりません.

＜IDATA 型＞

IDATA 型は，255 バイトという固定サイズのバイト値の配列で，IEBus フレームデータの配列を定義するのに使用します．各エントリは符号無しの８ビット値です．初期化されていない値はゼロ詰めされます.

例：
```
IDATA a={0x00,0x12,0x13}
a[10]=0x55
```

＜INT 型＞

INT 型は，整数の値を定義するのに使用します．内部的に符号付３２ビット(４バイト)のサイズです．値としては－２G(ギガ)~＋２G(ギガ)までの値を扱えます．
IDATA 型の各エントリや **IFRAME** および **IFMASK** の各メンバ(adr など)に **INT** 型の変数の値を代入することもできます．このときは，**INT** 型３２ビットの下位ビットだけが符号無し数として代入されます.

例：
```
IDATA a
INT y=1
a[2]=y
```

INT 型に限っては，１次元および２次元までの配列としても扱えます.

例：
```
INT y[2]
y[0]=50
y[1]=30

INT x[2][3]
x[0][2]=50
x[1][2]=30
```

＜**IFRAME** 型＞

IFRAME 型は，IEBus フレームを定義します．内部的には構造体となっており，各メンバは以下のとおりです．各メンバへのアクセスは C 言語と同様な方式で行います．すべてのメンバについて初期化されていない値はゼロ詰めされます.

```
IFRAME {
  INT bit          同報ビット
  INT adr          スレーブアドレス
  INT size         電文長
  IDATA data       フレームデータ配列
}
```

例：
```
IFRAME frame0
IDATA a
frame0.data[3]=5
frame0.data=a
IFRAME frame1={1,0x123,5,0x00,0x01,0x02}
```

補足)

上記で，1 が同報ビットです．0x123 がスレーブアドレスです．5 が電文長です．その後の 0x00, 0x01...はデータで, DATA 型同様 255 バイト固定のエリアを内部では確保しておき，初期化されていない部分は 0x00 詰めされます.

なお，同報ビットは，０で同報，１で個別を示します.

＜**IFMASK** 型＞

IFMASK 型は，IEBus フレームマスクを定義します．これは，後述の **CHECK** コマンドを使用して，受信したフレームの一部にマスキングをかけた値と規定値を比較チェックする場合などに使用します．内部構成は **IFRAME** 型と同じですので，アクセス方法などもまったく同じに行えます.

例：
```
IFRAME checkframe={1,0x123,5,0x00,0x01,0x02}
IFRAME recvframe
IFMASK fmask={0,0xffff,0x0,0xff,0xff,0xff}
RECV recvframe
CHECK checkframe recvframe fmask
```

補足)

上記の IFMASK 宣言では，同報ビットが非マスク（チェック対象外）で，アドレスがマスク（チェック対象），電文長が非マスク（チェック対象外），データのはじめの３バイト分がマスク（チェック対象）となっています.

RECV コマンドで受信したフレームと **checkframe** として宣言したフレームを **fmask** でアンドをとって比較するコマンドが **CHECK** コマンドです.

６．３．　定数

AP-Macro では定数が使用できます．定数値は C 言語と同様の仕様で，0x が頭に付く数値は 16 進数，0 が頭に付く数値は 8 進数，何も付かなければ 10 進数，となります．
定数は，変数の初期化値，値の代入，シフト値などに使用します．

６．４．　ラベル

AP-Macro ではラベルが使用できます．ラベルは，後述の **GOTO** コマンドや **IF～THEN** コマンドなどの行き先としてマクロファイルの位置を示す場合に使用します．
ラベルの有効範囲(スコープ)は，プロシージャ(後述)はその中のみとなり，また逆にメイン処理部からプロシージャ内部のラベルを参照することもできません．
ラベルの１文字目は必ずアルファベットの小文字で記述しなければなりません．また，ラベルに使える文字はアルファベットとアンダーラインと数字のみで，最大文字数は 255 文字です．

例：
label0:

上記のように**':'**文字で終わり，小文字のラベル名で始まる行です．ラベルを記述した行には他のコマンドなどを記述できません．

６．５．　プロシージャ

AP-Macro ではプロシージャというひとまとまりの処理が記述できます．プロシージャは，後述の定期処理用のプロシージャとして登録したり，キー処理用のプロシージャとして登録したり，GOSUB コマンドでサブルーチンとしてコールするためのものです．

例：
PROC proc0
・・・処理の記述
ENDPROC

このように必ず **PROC** ではじまり，**ENDPROC** で終わらなければなりません．
PROC 内からのラベルへの **GOTO** は同一の **PROC** 内に限られます(**IF** のラベルも同様)．また，**PROC** 内では定期送信やキー入力の設定，解除を行ってはいけません．

６．６．　演算

AP-Macro では変数への代入や演算が記述できます．式として１行に記述できる演算の種類を以下に示します．

a[3] += 5	定数値の加算
a[3] -= 5	定数値の減算
a[3] |= 5	定数値の論理和
a[3] &= 5	定数値の論理積
a[3] /= 5	定数値の除算
a[3] *= 5	定数値の積算
a[3] <<= 5	左ビットシフト
a[3] >>= 5	右ビットシフト
a[3] = 5	定数値の代入
a[3] += b	変数の加算
a[3] -= b	変数の減算
a[3] |= b	変数の論理和
a[3] &= b	変数の論理積
a[3] /= b	変数の除算
a[3] *= b	変数の積算
a[3] <<= b	左ビットシフト
a[3] >>= b	右ビットシフト
a[3] = b	変数の代入

ただし，左辺と右辺の変数型は同じものでなければなりません．
また，左辺，演算子，右辺の間には必ず空白(空白文字あるいは TAB)が入っていなくてはなりません．演算子の＋などと＝の間には空白があってはなりません．
論理和，論理積，ビットシフト演算に関しては，変数としては符号付きである **INT** 型などでも，符号無しの値として扱われます．

６．７．　フロー制御コマンド

AP-Macro でのフロー制御のコマンドとしては，以下があります．

＜SWITCH～CASE コマンド＞

SWITCH コマンドのパラメータとしては変数名を指定しなければなりません．変数の型は **INT** 型あるいは，**IDATA** 型の各エントリ，**IFRAME** 型および **IFMASK** 型の各メンバを指定することができます．
SWITCH～ENDSWITCH コマンドの中に **SWITCH～ENDSWITCH** コマンドを含めるという入れ子構造も記述できます．

例：

```
IDATA      abc
SWITCH abc[3]
 CASE 0
   ・・・処理の記述
 ENDCASE
 CASE 0x11
   ・・・処理の記述
 ENDCASE
 CASE 0x22
   ・・・処理の記述
```

```
  ENDCASE
  CASE DEFAULT
    ・・・処理の記述
  ENDCASE
ENDSWITCH
```

＜IF～THEN コマンド＞

IF の後に記述する式の評価結果に従って **THEN** の後に記述したラベルに飛びます．ここでは，符号付きの変数は符号付きの変数として，符号無しの変数は符号無しとして扱われます．

例：

```
IF a != 0 THEN label0
IF a == 0 THEN label0
IF a & 1 THEN label0
IF a < 1 THEN label0
IF a > 1 THEN label0
IF a <= 1 THEN label0
IF a >= 1 THEN label0
```

＜WHILE コマンド＞

WHILE の後に記述する式の評価結果が真である間, **ENDWHILE** までの処理を実行します．**WHILE～ENDWHILE** コマンドの中に **WHILE～ENDWHILE** コマンドを含めるという入れ子構造も記述できます．

例：

```
WHILE a!=0
  ・・・処理の記述
ENDWHILE
```

＜GOTO コマンド＞

GOTO の後に記述したラベルに飛びます．

例：

```
GOTO label0
```

＜EXIT コマンド＞

マクロ処理を強制的に終わらせます．パラメータなしで指定し，その場所でマクロから抜けてしまうものです．ただし，プロシージャ内で本コマンドを使用することはできません．

例：

```
EXIT
```

６．８．　その他のコマンド

AP-Macro ではそのほか，後述の CSV 形式ファイル用のコマンド以外に以下のコマンドがあります.

＜WAIT コマンド＞

指定したミリ秒分スリープします.

例：

```
WAIT 100
```

＜SEND コマンド＞

指定したフレームを送信します.

例：

```
SEND frame0
```

＜RECV コマンド＞

指定した変数にフレームを受信します．タイムアウト値（ミリ秒単位）を指定できます．タイムアウトは省略することも可能です(省略時はタイムアウトせず受信するまで待ちます).
タイムアウトが発生した場合はシステム定義変数 **errno** に-1 がセットされます.
正常受信した場合は 0 がセットされます.

例：

```
RECV frame0 100
```

＜CHECK コマンド＞

指定したフレームを，IFMASK 型の変数で指定したパターンで１の部分のみチェックします．比較した結果がイコールならば後述するシステム定義変数 **errno** には 0 がセットされ，イコールでなければ-1 がセットされます.

例：

```
IFRAME checkframe={1,0x123,5,0x00,0x01,0x02}
IFRAME recvframe
IFMASK fmask={0x0,0xffff,0x0,0xff,0xff,0xff}
RECV recvframe
CHECK checkframe recvframe fmask
IF errno != 0 THEN label_error
ECHO チェック OK
  ・・・処理の記述
label_error:
ECHO チェックエラー
```

補足)

上記の IFMASK 宣言では，同報ビットが非マスク（チェック対象外）で，アドレスがマスク（チェック対象），電文長が非マスク（チェック対象外），データのはじめの３バイト分がマスク（チェック対象）となっています.
RECV コマンドで受信したフレームと **checkframe** として宣言したフレームを **fmask** でアンドをとって比較します.

＜KEYGOTO コマンド＞

指定したキーを押されたときまたは離されたときにジャンプする飛び先ラベルを定義します．キーは 0 から 9 の数字，f1～f9 のファクションキー，up/down/left/right の矢印キー，a～z の英字(大文字・小文字の区別はありません)です．

例：
KEYGOTO 0 label0 (u)

ラベルの後の u はオプションで省略可です．指定するとキーを離した場合の定義となります．

＜KEYGOSUB コマンド＞

指定したキーを押されたときにまたは離されたときに実行するキー処理プロシージャを定義します．キーは 0 から 9 の数字，f1～f9 のファクションキー，up/down/left/right の矢印キー，a～z の英字(大文字・小文字の区別はありません)です．

例：
KEYGOSUB 0 proc0 (u)

プロシージャ名の後の u はオプションで省略可です．指定するとキーを離した場合の定義となります．

＜BEEP コマンド＞

システムで定義されたビープ音・サウンドを鳴らします．パラメータの minfo(メッセージ情報)，mwarn(メッセージ警告)，syserr(システムエラー)，mques(問い合わせ)，ok(一般の警告音)は[コントロールパネル]―[サウンド]で割り当てられた各サウンドです．パソコンにサウンドカードが無い場合，ボリュームが絞られている場合は鳴りません．パラメータの pc は内蔵スピーカのビープ音です．これらのいずれかを指定できます．

例：
BEEP pc/minfo/mwarn/syserr/mques/ok

＜TEIKI コマンド＞

定期処理プロシージャを登録します．登録できる番号は 0 から 9 までです．最後のパラメータはミリ秒単位の時間間隔です．

例：
TEIKI 0 proc0 100

＜TSTOP コマンド＞

定期処理プロシージャを登録破棄します．

例：
TSTOP 0

＜ECHO コマンド＞

デバッグ文字列表示ウィンドウに指定文字列を表示します.

例：

```
ECHO これはデバッグプログラムです
```

＜GOSUB コマンド＞

プロシージャを呼び出します．呼び出し元は内部スタックに保存されますので, 呼び出されたプロシージャから更にプロシージャを呼び出すことも可能です.

例：

```
GOSUB proc0
```

＜XOR コマンド＞

左辺に指定した変数と右辺に指定した変数／定数を排他的論理和演算し, 左辺に結果を代入します．指定した変数／定数は符号無しの値として扱われます．指定した変数が配列変数の場合は，1 要素のみが指定できます.

例：

```
INT a=0xffffffff
INT b=0xCCCCCCCC
INT c[2][3]
XOR a 0x30303030
XOR a b
c[1][0]=0xddddddddd
XOR c[1][0] b
```

＜INV コマンド＞

指定した変数をビット反転して結果を代入します．指定した変数／定数は符号無しの値として扱われます．指定した変数が配列変数の場合は，1 要素のみが指定できます.

例：

```
INT a=0xffffffff
INT c[2][3]
INV a
c[1][0]=0xddddddddd
 INV c[1][0]
```

＜INCLUDE コマンド＞

ファイルをインクルードし，その場所に展開します．インクルードファイルのファイル名や拡張子は何でも構いません．””で囲んでファイルのフルパスを指定します．パスを省略した場合はカレントディレクトリになります.

例：

```
INCLUDE “file.inc”
INCLUDE “c:¥user¥file.inc”
```

＜DEFINE コマンド＞

定数や変数の置き換え文字列を定義します．置き換え文字列は変数同様，英小文字ではじまる文字列でなくてはなりません．また，変数名やラベル名，PROC 名，コマンド名と重複してはいけません．定義は２５６個までできます．
変数のメンバのみを定義することはできません．また，””で囲まれた文字列内を定義することはできません．

例：
DEFINE abcd 1
DEFINE aa55 frame.data[1]

＜TIMEGOSUB コマンド＞

指定時間(ミリ秒)後に実行されるプロシージャを定義します．登録できる番号は 0 から 9 までです．タイマ実行はワンタイムで，タイムアウト処理などに利用できます．周期タイマ処理は **TEIKI** コマンドを使用してください．

例：
TIMEGOSUB 0 proc0 800

＜TIMEGOTO コマンド＞

指定時間(ミリ秒)後に実行位置を変更するラベルを定義します．登録できる番号は 0 から 9 までです．タイマ実行はワンタイムで，タイムアウト処理などに利用できます．周期タイマ処理は **TEIKI** コマンドを使用してください．ラベルの制限事項(スコープ)は他の場合と同じで，同一 PROC 内，またはメイン処理内に限られます．

例：
TIMEGOTO 0 label0 800

＜TIMESTOP コマンド＞

TIMEGOSUB コマンド，**TIMEGOTO** コマンドで定義したプロシージャ，ラベル分岐を登録破棄します．番号は 0 から 9 までです．

例：
TIMESTOP 0

６．９．　ＣＳＶ形式ファイル用コマンド

AP-Macro では，CSV 形式のファイル（カンマで各フィールドを区切った１行１レコードのテキストファイル）が利用できます．CSV 形式のファイルを利用するためのコマンドは以下のとおりです．

＜COPENR コマンド＞

CSV 形式ファイルを読み込みモードでオープンします．ファイルのファイル名や拡張子は何でも構いません．””で囲んでファイルのフルパスを指定します．パスを省略した場合はカレントディレクトリになります．これ以降のファイルの読み込みはすべてこのファイルに対して行われます．

例：
COPENR “c:¥user¥frame.csv”

＜COPENW コマンド＞

CSV 形式ファイルを書き出しモードでオープンします．ファイルのファイル名や拡張子は何でも構いません．””で囲んでファイルのフルパスを指定します．パスを省略した場合はカレントディレクトリになります．これ以降のファイルの書き出しはすべてこのファイルに対して行われます．

例：

```
COPENW “c:¥user¥frame.csv”
```

＜CCLOSE コマンド＞

COPENR または COPENW でオープンされたファイルをクローズします．'R'文字または'W'文字で区別します．

例：

```
CCLOSE   R
CCLOSE   W
```

＜CLOAD コマンド＞

読み込みモードでオープンしたファイルから指定した変数に１レコード分(１行分)読み込みます．”data0[3]”や”frame.bit”のように１配列要素や１メンバの指定もできます．１レコード分で，余ったフィールドのデータ(変数に入りきらなかったデータ)は捨てられます．

IDATA 型や INT 型の 1 次元配列の変数，2 次元配列の変数では，1 要素の指定だけでなく，配列全体の指定や 2 次元配列の列全体といった指定もできます．

例：

```
INT int0
INT int1[2]
INT int2[2][3]
CLOAD frame0
CLOAD frame0.bit
CLOAD data0
CLOAD data0[2]
CLOAD int0
CLOAD int1
CLOAD int1[1]
CLOAD int2
CLOAD int2[1]
CLOAD int2[1][1]
```

＜CSAVE コマンド＞

書き出しモードでオープンしたファイルに指定した変数の内容を１レコード分(１行分)書き出します．”data0[3]”や”frame.bit”のように１配列要素や１メンバの指定もできます.

IDATA 型や INT 型の 1 次元配列の変数，2 次元配列の変数では，1 要素の指定だけでなく，配列全体の指定や 2 次元配列の列全体といった指定もできます.

例：

```
INT int0
INT int1[2]
INT int2[2][3]
CSAVE frame0
CSAVE frame0.bit
CSAVE data0
CSAVE data0[2]
CSAVE int0
CSAVE int1
CSAVE int1[1]
CSAVE int2
CSAVE int2[1]
CSAVE int2[1][1]
```

６．１０．システム定義の変数

AP-Macro では, errno というシステム定義の整数値の変数があります. これは, CHECK コマンド等のコマンド行を実行した際に発生したエラー値を保存しています．CHECK コマンドと IF～THEN コマンドを組み合わせて使用する際等に使用します.

６．１１．サンプルマクロ

AP-Macro 言語で記述したマクロファイルのサンプルを下記に示します．マクロファイル本文中，左端の“行番号：”は説明のための便宜上のもので，実際のマクロファイルには記述してはいけません．

```
 1：#
 2：# サンプルマクロ
 3：#
 4：
 5：# 変数宣言
 6：IFRAME   fSend0 = {1, 0x123, 5, 0x00, 0x11, 0x22, 0x33, 0x44}
 7：IFRAME   fSend1 = {1, 0x123, 3, 0x56, 0x78, 0x9a}
 8：IFRAME   fSend2 = {1, 0x123, 1, 0xff}
 9：IFRAME   fSend3 = {1, 0x123, 1, 0x55}
10：IFRAME   fSend4 = {1, 0x123, 1, 0xee}
11：IFRAME   fCmp =   {0,      0x123,   0,      0x00, 0x42}
12：IFMASK   fMask = {0x00, 0xFFFF, 0x00, 0x00, 0xFF, 0x00, 0x00, 0x00}
13：IFRAME   fRecv
14：INT      cnt
15：
16：# 定期送信プロシージャ
17：PROC     teikiProc0
18：  cnt = 3
19：  WHILE cnt!=0          # ３回繰り返し
20：  SEND   fSend1
21：  cnt -= 1
22：  ENDWHILE
23：ENDPROC
24：
25：# 5 キー押下時プロシージャ
26：PROC     key5Proc
27：  SEND   fSend2
28：ENDPROC
29：
30：# 処理
31：
32：KEYGOSUB 5 key5Proc          # 5 キー押下時の処理を定義
33：
34：label0:
35：RECV     fRecv          # 条件に合ったフレームの受信を待つ
36：IF errno!=0 THEN error
37：CHECK    fRecv fCmp fMask
38：IF errno==0 THEN label1
39：GOTO label0
40：
41：label1:
42：WAIT 500
43：SEND fSend2
44：IF errno!=0 THEN error
45：label2:
46：RECV fRecv                   # 受信データの１バイト目によって処理を変更
47：IF errno!=0 THEN error
48：SWITCH fRecv.data[0]
```

```
４９：   CASE 0x11
５０：           TEIKI 0 teikiProc0 5000
５１：           GOTO label3
５２：   ENDCASE
５３：   CASE 0x22
５４：           SEND fSend3
５５：           IF errno!=0 THEN error
５６：   ENDCASE
５７：   CASE DEFAULT
５８：           SEND  fSend3
５９：           IF errno!=0 THEN error
６０：   ENDCASE
６１：ENDSWITCH
６２：GOTO label2
６３：
６４：label3:
６５：RECV      fRecv           # 受信データの１バイト目が 0x88 まで待つ
６６：IF errno!=0 THEN error
６７：IF fRecv.data[0]==0x88 success
６８：SEND fSend3
６９：IF errno!=0 THEN error
７０：goto label3
７１：
７２：error:
７３：   EXIT
７４：success:
７５：   EXIT
```

各行の説明

1 行目～5 行目：

コメント行および空白行です．実行時には無視されます．

６行目～１４行目：

本マクロ内で使用される変数を宣言しています．AP-Macro では，変数宣言はまとめて，処理記述の前に記述しなければなりません．

初期化データが省略された分は，ゼロ詰めして初期化されます．**IFRAME** 型の変数の初期化データは，左から順に同報ビット，アドレス，電文長，そしてデータ部分へと詰めてセットされます．省略したデータ部分はやはりゼロ詰めされます．

また，AP-Macro では変数はすべてグローバル変数となります．変数に関してスコープの概念はありません．これは，プロシージャ内も同じです．区別なくすべての変数は共通に参照されます．

１７行目～２３行目：

後で定期処理として使用するためのプロシージャです．ここでは，**IFRAME** 型の **fSend1** 変数を 3 回送信しています．

２５行目～２８行目：

後でキー処理として使用するためのプロシージャです．ここでは，**IFRAME** 型の **fSend2** 変数を送信しています．

３０行目～：

ここからメインの処理として実行されます．

３２行目：
キー処理の定義を行っています．これ以降，数字の’５’のキーが押下されると **key5Proc** が起動されるようになります．

３３行目：
ラベルを定義しています．

３５，３６行目：
fRecv 変数に受信します．次の行では前行の **RECV** コマンドでエラーが起こったかどうかをチェックし，もしエラーが起こっていたら７２行目のラベル **error** に飛びます．

３７～３９行目：
fRecv 変数に受信したフレームを **fMask** 変数とアンドをとり，**fCmp** 変数の内容と比較チェックします．次の行では前行の **CHECK** コマンドの結果がイコールかそうでなかったかをチェックし，イコールならば４１行目の **label1** に飛びます．そうでなければ次行の **GOTO** コマンドで３４行目の **label0** に戻り，再び受信～チェックを繰り返します．

４２～４４行目：
WAIT コマンドで 500 ミリ秒待って，**fSend2** 変数の内容を送信します．送信エラーが起これば７２行目のラベル **error** に飛びます．

４５～６２行目：
ここでは，**SWITCH～CASE** コマンドを使って，受信したフレームのデータ部の１バイト目によって処理を分けています．データ部の１バイト目が 0x11 だった場合は **TEIKI** コマンドで 5 秒ごとに **teikiProc** プロシージャを起動するようセットして６４行目の **label3** に進み，データ部の１バイト目が 0x22 だった場合は **fSend3** 変数の内容を送信してから，**GOTO** で **label2** に戻って再び受信を待ちます．**CASE DEFAULT** はデータ部の１バイト目が 0x11 でも 0x22 でもなかった場合に実行される部分で，内容としては 0x22 の場合と同じことをしています．

６４～７０行目：
ここでは，**IF～THEN** コマンドを使って，受信したフレームのデータ部の１バイト目によって処理を分けています．データ部の１バイト目が 0x88 だった場合は７４行目の **success** に飛んでマクロの処理を終了します．そうでない場合は **fSend3** 変数の内容を送信してから，再び６４行目の **label3** に戻ります．

７２行目：
本マクロ中で受信エラー，送信エラーが起こった場合に飛んでくるラベル **error** です．

７４行目：
本マクロ中で最終的にすべての処理が正常に進んだ場合に飛んでくるラベル **success** です．

第７章　通信仕様

以下に AP-ALDM2 とパソコン間の RS-232C 通信仕様を示します。データは全てバイナリー形式です。

AP-ALDM２ → パソコン

同報 bit	1byte
マスタアドレス	2byte
電文長	1byte
データ	1～32byte(電文長による)
ステータス	1byte(00h で正常、01h で通信エラー発生)

パソコン → AP-ALDM２

同報 bit	1byte
スレーブアドレス	2byte
電文長	1byte
データ	1～32byte(電文長による)

通信パラメータ

ボーレート	115200bps
パリティ	なし
データビット	８bit
ストップビット	１bit
フロー制御	CTS/RTS フロー制御