EPWriter20

スクリプトとマクロの利用の手引

Release 1.1

Monday, August 24, 2009

株式会社ソリトンウェーブ

■履歴

平成 21 年 2 月 13 日	第 1 版
平成 21 年 8 月 24 日	Release 1.1 BlankCheck の追加、Read/Write/Verify の領域指定の追加

目次

概要

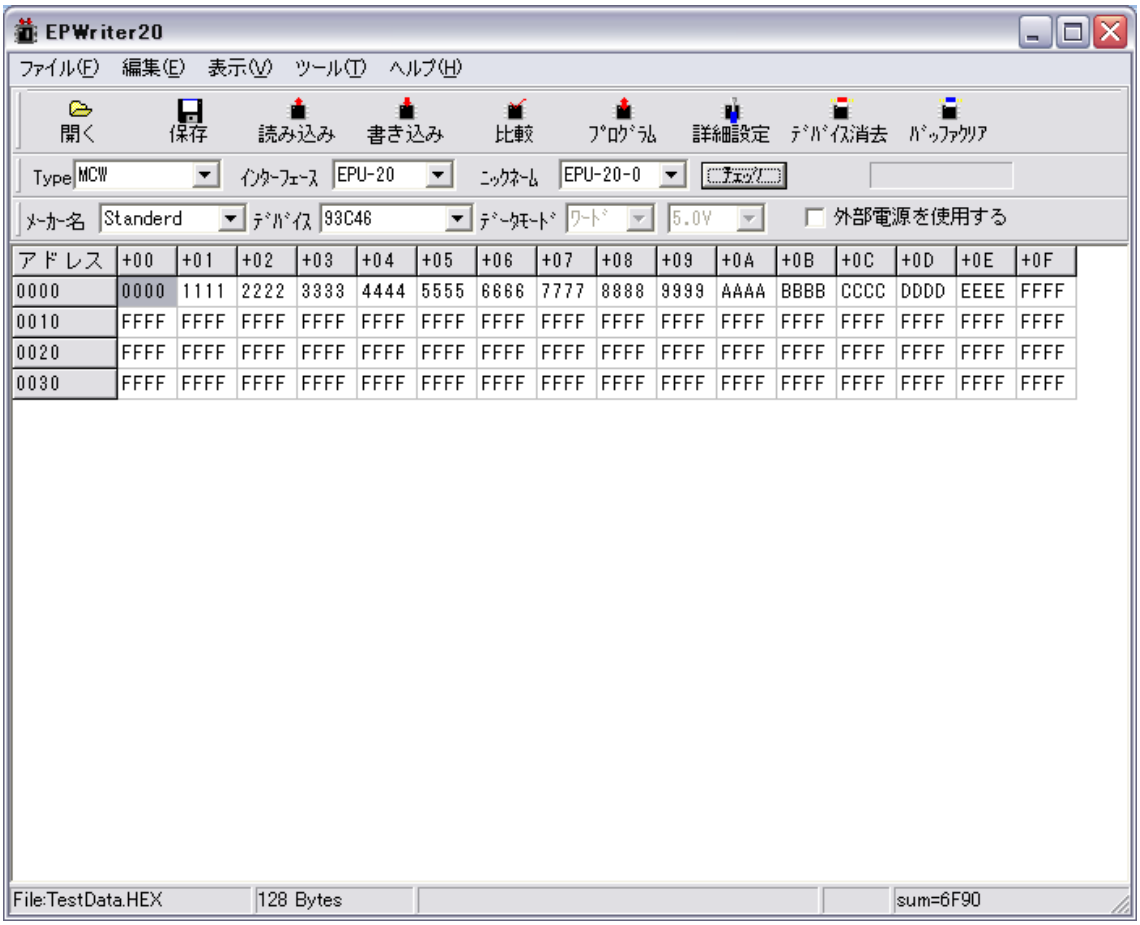
このドキュメントでは、EPWriter20 のスクリプトの使い方を、簡単に説明します。
EPWriter20 には、スクリプトとマクロの２つの機能があります。スクリプトでは、専用のスクリプト言語を使って、EPWriter20 の操作を自動化することができます。
スクリプトを使うと、複雑な手順を記憶させて、確実に実行させたり、繰り返し行う作業を、自動化することができます。EPWriter20 のスクリプト言語は、C 言語を非常にシンプルにしたような仕様となっていますので、C 言語に慣れている開発者の方はも緒論ですが、C 言語を使ったことのない人でも、簡単に使用することができます。また、EPWriter20 では、マクロ機能も搭載しています。マクロ機能では、実際に EPWriter20 を操作した手順を、記憶／再生することができます。マクロで記憶した手順は、スクリプトとして記憶されますので、この機能を使えば、プログラムを全く行わなくても、スクリプトを使用することができます。さらに、スクリプトを理解することで、マクロも十分に使いこなすことできるようになります。

簡単な例題

まずは、マクロを使って、通常の作業を自動化してみることにします。
使用するデバイスは、マイクロワイヤデバイスの 93C46 で、メーカは指定しないことにします。ある製品で、このデバイスを使用するため、製品の初期化データを、毎日 100 個書き込む必要があるとして、この手順を自動化することにします。
マクロの作成手順は、次のようになります。

1）データの準備

マクロを作成する前に、デバイスに書き込むデータを作成します。まず、EPWriter20 を起動し、デバイスを Standard の 93C46 に設定します。

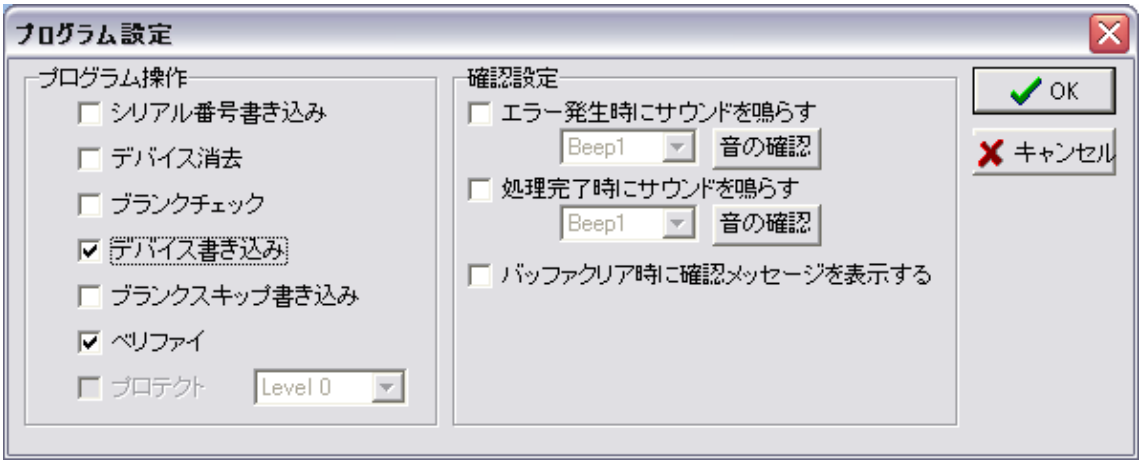


バイナリ編集画面で、デバイスに書き込むデータを作成し、ファイルメニューから、「ファイルの保存」を選択して、データを保存します。ここでは、データを上記のように作成し、ファイル名と保存先を、c:¥TestData.hex とします。

２）プログラム設定の変更

デバイスの書き込みには、EPWriter20 のプログラム機能を使用します。そこで、あらかじめプログラムの設定を行い、どのようなモードでデバイスに書き込むかを決めておきます。プログラムの設定は、「ツール」メニューの、「プログラム設定」で行います。

ここでは、図のように、標準的な、「書き込み」と「ベリファイ」を行って、プログラムを行うことにします。



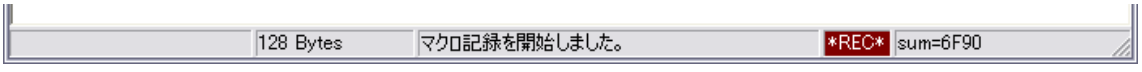
上記のように設定したら、OK を押して、ダイアログを閉じます。

３）マクロの作成

ここまでの設定で、マクロ作成の準備が整いましたので、マクロの作成を行います。

マクロは、実際にデバイスにデータを書き込む手順を記憶しますので、書き込み可能なデバイスをセットして、マクロを作成します。

デバイスの準備ができたら、マクロの作成を行います。マクロの記憶には、[CTRL]+[ALT]+R キーを押すか、「ツール」メニューから、「マクロ記録」を選択していください。マクロの記録が開始されると、図のように、ステータスバーに、マクロ記録中のメッセージが現れます。



この状態で、先ほど作成したデータファイルを読み込み、デバイスにデータを書きこみます。書き込み手順は、次のようになります。

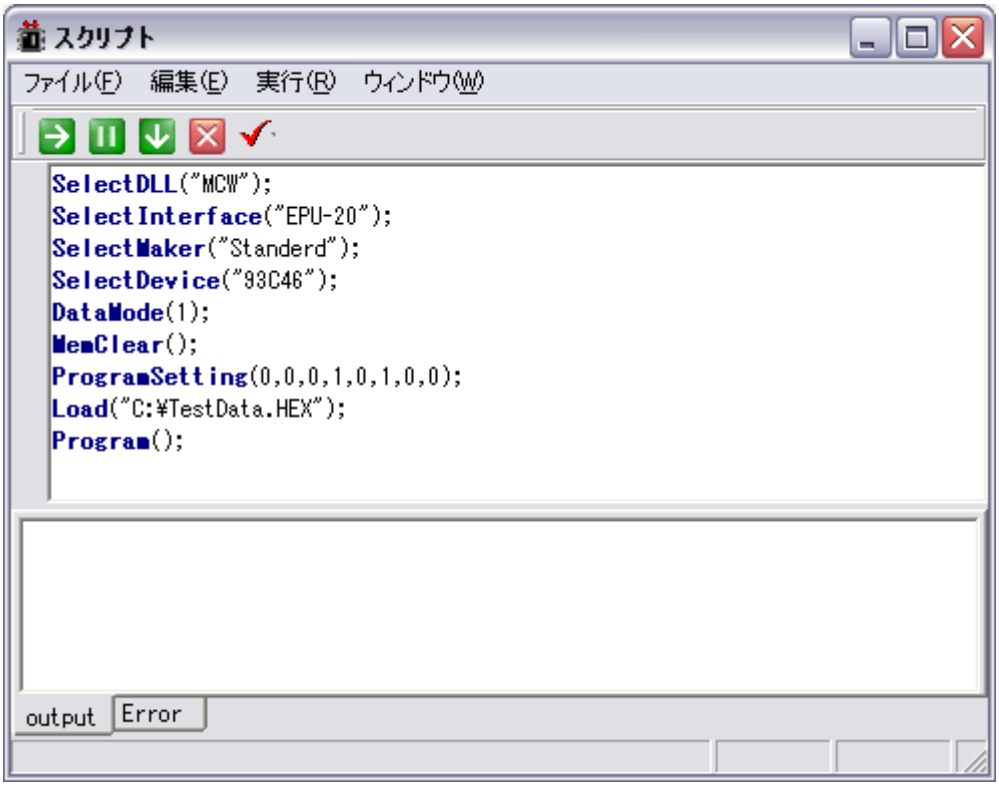
・　「ファイル」メニューから、「ファイルを開く」を選択して、c:¥TestData.hex を読み込む

・　「ツール」メニューから、「プログラム」を選択して、デバイスにデータを書き込む

書き込みが終了したら、再度[CTRL]+[ALT]+R キーを押すか、「ツール」メニューから、「マクロ記録」を選択してマクロの記録を終了します。

４）マクロの内容

マクロの記録が終了すると、図のようなウィンドウが起動して、作成されたマクロが表示されます。



この画面は、スクリプトエディタの画面で、作成されたマクロが、エディタに読み込まれた状態です。このマクロの内容については、後ほど詳しく説明するとして、とりあえず、このマクロをファイルに保存します。スクリプトエディタの、「ファイル」メニューから、「スクリプトファイルの保存」を選択して、マクロを保存します。ここでは、ファイルの保存先を、c:¥TestData.scp とします。ファイルを保存したら、「ファイル」メニューから、「終了」を選んで、スクリプトエディタを閉じてください。

５）マクロの実行

とりあえず、作成されたマクロが正しく動作するかどうか、確認してみることにします。データが読み込まれることを確認するため、「編集」メニューで、「バッファのクリア」を選択して、バッファをクリアしておきます。

「ツール」メニューで、「マクロの再生」を選択すると、記録した操作が再現され、書き込み用のデータがバッファにロードされ、デバイスへ書き込まれます。

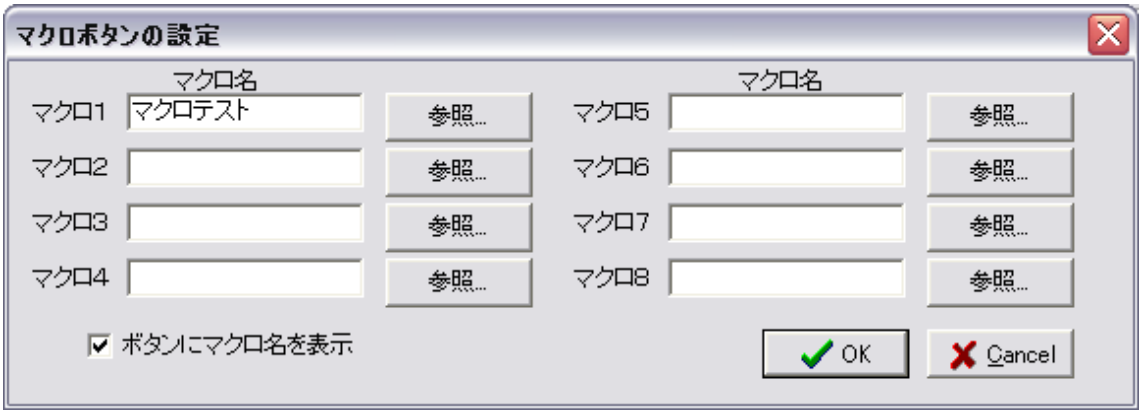
６）マクロのボタン化

これで、作成したマクロはいつでも使用可能な状態になります。現在は、作成したマクロ

がメモリにロードされている状態ですが、EPWriter20 を一旦終了すると、ロードされたマクロは消えてしまいますので、このマクロがいつでも使えるように、マクロをボタン化することにします。

マクロのボタン化は、次の手順で行います。

- 「ツール」メニューから、「マクロボタンの設定」を選択します。
- マクロ１のエディットボックスの右側の「参照...」ボタンを押して、先ほど作成したマクロ(c:¥TestData.scp)を選択します。
- マクロ１のマクロ名を、任意の名前（ここでは「マクロテスト」とする）にします。
- 「ボタンにマクロ名を表示」にチェックを入れ、OK を押します。



OK を押すと、EPWriter20 の画面に、図のようにマクロボタンが現れます。ボタンが現れない場合は、「表示」メニューで、「マクロボタン・バー」を選択してください。

このボタンは、EPWriter20 を次回起動した場合も、表示されますので、このボタンを押すことで、いつでもこのデータを書き込むことができます。

※注意

- マクロボタンに設定されたスクリプトファイルを移動したり、削除したりすると、マクロボタンを押しても、スクリプトファイルがないため、エラーとなります。
- マクロで使用しているデータファイル（TestData.hex）を移動したり、削除したりすると、マクロ実行中に、データファイルが見つけられず、エラーとなります。

7）マクロのテスト

マクロをボタン化したところで、マクロのテストを行ってみることにします。

ために、デバイスタイプのドロップダウン・リスト(Type)で、I2C を選択してみてください。

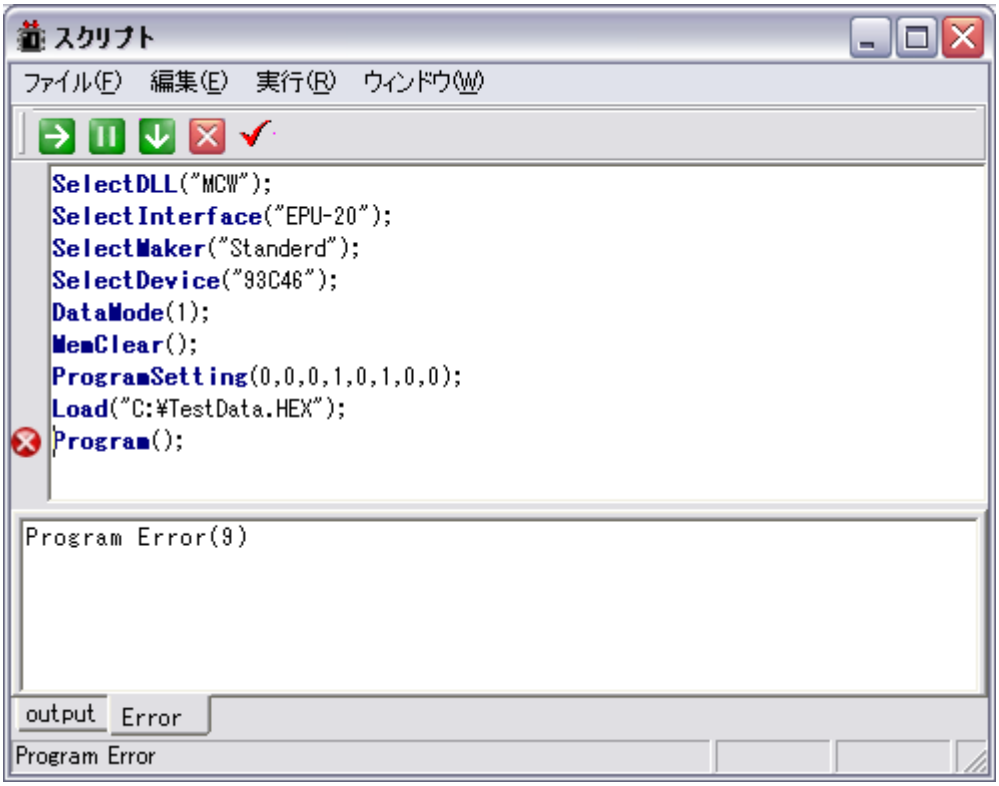
I2C を選択すると、使用デバイスが変更になり、バッファがクリアされます。

デバイスに、93C46 が挿入されているを確認して、「マクロテスト」ボタンを押してみます。

ボタンを押すと、デバイスが変更され、プログラムが実行されます。エラーがなければ、マクロはそのまま終了します。

こんどは、エラーが発生した場合の表示を確認してみます。

EPU-20 から、93C46 のデバイスを取り外し、再度「マクロテスト」ボタンを押してみてください。今度は、デバイスが挿入されていないため、エラーが発生し、次のようなウィンドウが表示されます。



これは、スクリプト実行中に、Program()行を実行した時に、エラーが発生したことを示しています。

エラーが確認できたら、このウィンドウを閉じてください。

作成されたスクリプトの詳細

簡単な動作を確認したところで、マクロで自動作成されたスクリプトについて説明します。
「ツール」メニューから、「スクリプトエディタ」を選択すると、再度スクリプトエディタのウィンドウが表示され、スクリプトの内容の確認や修正ができるようになっています。
自動作成されたスクリプトは、次のようなソースです。
[スクリプトのソース]

```
SelectDLL("MCW");
SelectInterface("EPU-20");
SelectMaker("Standerd");
SelectDevice("93C46");
DataMode(1);
MemClear();
ProgramSetting(0,0,0,1,0,1,0,0);
Load("C:¥TestData.HEX");
Program();
```

EPWriter20 のスクリプト言語は、C 言語と似たような書式となっていて、青字で表示されている部分が関数名で、続くカッコの中が、その関数の引数となります。
以下、それぞれについて、簡単に説明します。

- SelectDLL
  使用するDLLを選択します。DLLの名称は、Typeのコンボボックスの名前で、現在使用できるものは、"MCW","I2C","SPI"の何れかになります。
- SelectInterface
  インターフェース・ハードウェアを選択します。引数は、インターフェース機器名で、"EPU-10"もしくは"EPU-20"の何れかになります。
- SelectMaker
  メーカー名を指定します。メーカー名のリストボックスの名前が引数になります。"Standerd"は標準設定で、特にメーカーを指定せず、一般的な互換デバイスの場合の設定になります
- SelectDevice
  デバイスを選択します。引数には、デバイスのコンボボックスの名前がはいります。
- DataMode
  データモードを指定します。0ならバイトモード、1ならワードモードです。
- MemClear
  メモリバッファをクリアします。
- ProgramSetting
  プログラム設定を行います。引数の値は、プログラム設定のダイアログの設定をそのまま反映させて

います。ダイアログで希望する設定を行って、マクロ記録を行うと、対応するProgramSettingのスクリプトが作成されますので、その値をそのまま使ってください。

- Load
  引数で指定されたファイルを読み込みます。ここでは、"C:¥TestData.HEX"を読み込んでいます。
  なお、スクリプト関数の引数は、文字列の場合は、ダブルコーテーション(")で囲み、数値や変数の場合は、そのまま変数名や数値を記述します。
- Program
  プログラムを行います。

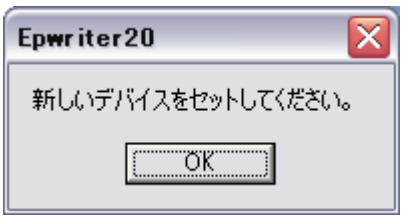
簡単な応用

スクリプトを変更して、100 個のデバイスに書き込むためのプログラムを作ってみましょう。

スクリプトエディタを開き、ソースを次のように変更します。

```
SelectDLL("MCW");
SelectInterface("EPU-20");
SelectMaker("Standerd");
SelectDevice("93C46");
DataMode(1);
MemClear();
ProgramSetting(0,0,0,1,0,1,0,0);
Load("C:¥TestData.HEX");
for(i=0;i<100;i++){
    ShowMsg("新しいデバイスをセットしてください。");
    Program();
}
ShowMsg("プログラムは終了しました。");
```

Progaram()の部分を、for ループで 100 回行うように変更しています。
変更が終わったらファイルの保存を行い、ファイルを同じ名前で上書きして、スクリプトエディタを終了します。
マクロボタンをおして、マクロを実行すると、次のようなメッセージが出ます。

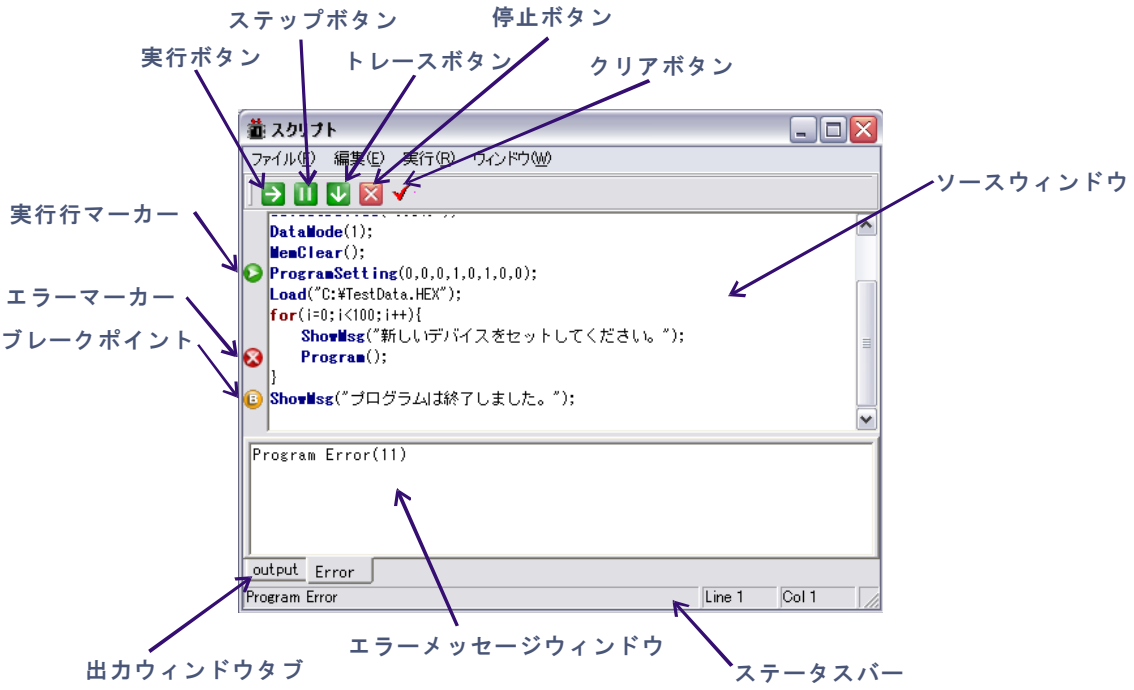


OK を押すとデバイスに書き込みが行われ、再度同じメッセージが表示されます。
この動作を 100 回繰り返して、マクロは終了します。
途中で書き込みエラーが発生すると、マクロは中断されて、エラーメッセージが表示されます。

このプログラムでは、単純に 100 回の書き込みを行っていますが、途中でエラーが発生すると、そこで終了してしまいます。もっと実用的なプログラムにするためには、エラー発生時には、メッセージを出して、デバイスの確認をさせてから、再度書き込みを行ったり、あるいは、正常に書き込めた数と、エラーが発生した数を表示するなどの工夫が必要になります。これらの拡張方法については、後の章を参照してください。

スクリプトエディタの使い方

次の図は、スクリプトエディタの画面の各部の名称です。



各部の機能は、次のようになっています。

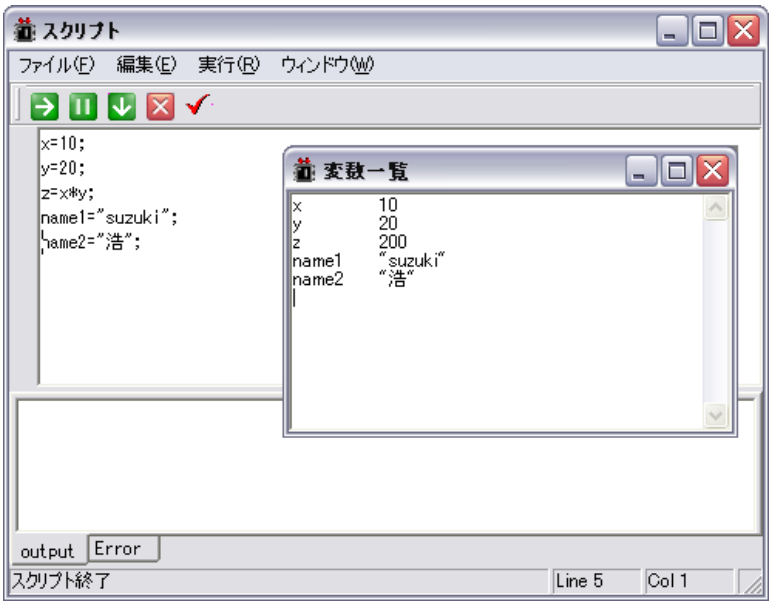
- 実行ボタン
  スクリプトを実行します。マクロボタンを押したときと同じ動作で、スクリプトを最後まで実行します。
- ステップボタン
  スクリプトの現在行を実行し、次の行で停止します。
- トレースボタン

スクリプトの現在行を実行し、次のステップで停止します。現在行がサブルーチンコールであれば、サブルーチンの最初の行で停止します。現在行がサブルーチンコールでない場合は、ステップボタンと同じ動作となります。

- 停止ボタン
  スクリプトの実行を停止させます。スクリプトを強制的に停止させたい場合に使用します。
- クリアボタン
  出力ウィンドウとエラーメッセージウィンドウの画面をクリアします。
- 実行行マーカー
  スクリプトの停止中に表示され、次に実行する行を表示します。
- エラーマーカー
  エラーの発生した行を表示します。
- ブレークポイント
  設定されたブレークポイントを表示します。ブレークポイントの設定は、ブレークポイントを設定したい行の、マーカーエリアの位置をマウスでクリックすることで、設定が行えます。設定したブレークポイントを解除したい場合は、解除したいブレークポイントのマーカーをクリックします。ブレークポイントは、複数設定することができます。
- ソースウィンドウ
  ソースの編集用ウィンドウです。スクリプトを読み込んで、ソースを編集することができます。
- エラーウィンドウ
  エラー発生時、またはエラーウィンドウタブをクリックしたときに表示されます。
  エラーメッセージは、「エラー内容(行番号)」の形式で表示され、エラーメッセージをダブルクリックすると、そのエラーの行が表示されます。
- 出力ウィンドウ
  エラーが発生していない場合、または出力ウィンドウタブをクリックすると表示されます。出力ウィンドウには、printf を使って出力した文字列が表示されます。
  また、このウィンドウの任意の行を編集して、リターンキーを押すと、その行を実行することができます。たとえば、出力ウィンドウに、次のような行を書いて、実行すると、変数 x の値を見ることができます。
  printf(x)
  この機能は、スクリプトのデバッグ時に便利です。

変数ウィンドウ

「ウィンドウ」メニューから、「変数ウィンドウ」を選択すると、スクリプト内で使用している変数の値が表示されます。



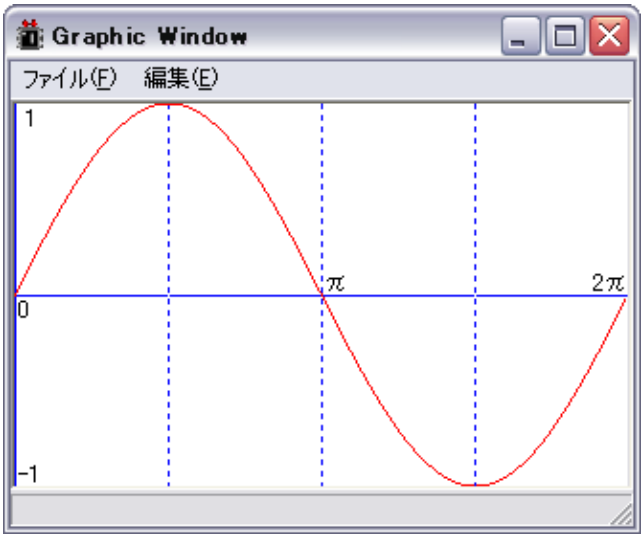
変数は、スクリプトを停止した時点で表示されるため、スクリプトをそのまま実行した場合は、表示されません。また、変数は、使用した時点で初めて確保されますので、未実行の変数は、表示されません。

グラフウィンドウ

スクリプトのグラフィック機能を使って、簡単なグラフを作ることができます。

「ウィンドウ」メニューから、「グラフウィンドウ」を選択すると、グラフウィンドウが表示されます。グラフィック機能の出力は、グラフィックウィンドウに行われます。

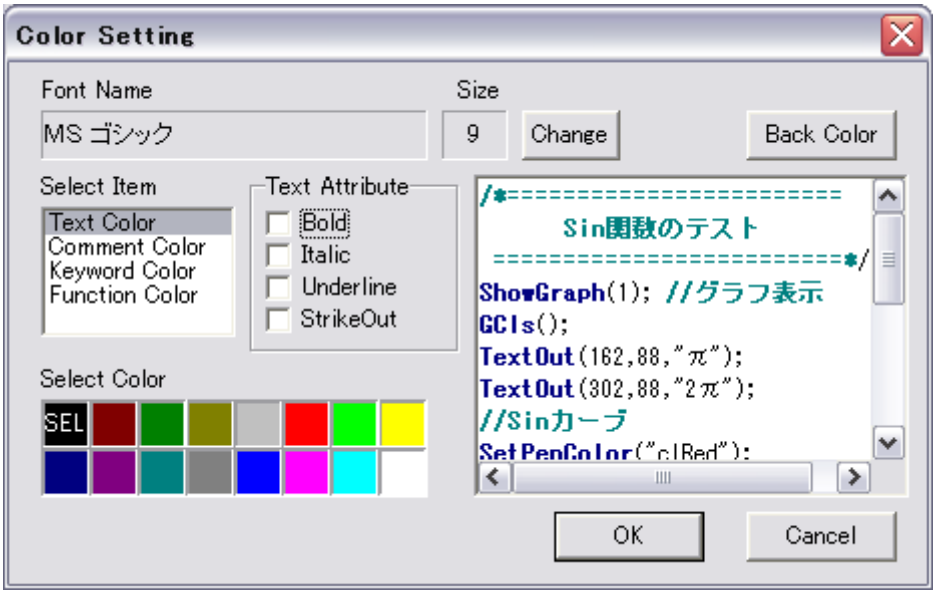
次の画面は、サンプルスクリプト「SinTest.scp」の実行画面です。



スクリプトのサンプルは、EPWriter20 をインストールしたフォルダの、ScpSample フォルダに格納されています。

色の設定

「ウィンドウ」メニューから、「色の設定」を選択すると、色の設定ダイアログが表示されます。このダイアログでは、スクリプトエディタで使用するフォントの種類やサイズ、画面の色の設定などが行えます。



スクリプトの言語仕様

EPWriter20 のスクリプトは、C 言語ライクなスクリプトエンジンを搭載しています。
ここでは、EPWriter20 のスクリプトの言語仕様を説明します。

スクリプトファイル

スクリプトファイルはテキストファイルで、拡張子が SCR となります。スクリプトファイルは、一旦メモリ上に読み込まれ、内部で展開されてから実行されます。スクリプトの実行は、先頭行から順に実行されます。スクリプトファイルは、C 言語と同様、フリーフォーマットですので、１つの文を複数行に分けて記述することが可能です。

コメント

コメントは、C 言語や C++言語で使用されるものと同じです。スクリプトテキスト内で、/*と*/で囲まれた部分は、コメントとして識別されます。
また、//から行末までもコメントとして扱われます。
コメントは、スクリプトソースの説明などを記録するためのもので、実行プログラムには、何の影響も与えません。

識別子

識別子は、英数字と記号の文字列で、必ず英字か記号で始まらなければなりません。また、大文字と小文字の区別もありません。識別子で使用できるのは、次の文字です。
英字：A-Z,a-z
数字：0-9
記号：_

予約語

次の識別子は、予約語ですので、変数名等には使用できません。

“while”,”loop”,”if”,”else”,”call”,”break”,”default”

変数

変数は、整数、実数、及び文字列を格納できます。変数には宣言文は無く、初期値はゼロになります。変数の型は、変数の初期化（代入）時に決定されます。整数を代入すれば、整数変数、文字列を代入すれば、文字列の変数となります。文字列の場合は、数値や変数名と区別するため、次の使用例ように、ダブルコーテーションで囲みます。

変数の使用例）

```
x=10;    //変数 x に整数の 10 を格納する
```

pi=3.14;　　//変数 pi に、実数 3.14 を格納する
name=”suzuki”;　//変数 name に、”suzuki”という文字列を格納する。

演算子

次の表は、演算子の一覧と優先順位です。演算子の機能は、C 言語と同様です。

番号	演算子	優先度	機能
1	(	12	開カッコ
2	)		閉カッコ
3	++	11	インクリメント
4	−−		デクリメント
5	!		論理反転
6	~		ビット反転
7	*	10	積
8	/		商
9	%		剰余
10	+	9	加算
11	−		減算
12	<<	8	左シフト
13	>>		右シフト
14	<	7	比較
15	<=		比較
16	>		比較
17	>=		比較
18	==	6	比較
19	!=		比較
20	&	5	AND
21	^	4	XOR
22	|	3	OR
23	&&	2	論理積
24	||	1	論理和
25	=	0	代入
26	+=		代入
27	−=		代入
28	*=		代入
29	/=		代入
30	%=		代入
31	&=		代入
32	^=		代入
33	|=		代入
34	<<=		代入
35	>==		代入

条件文

条件文は、次の２つの何れかの形式になります。

```
if(式){
        文
}
```

この形式では、式が真（0 ではない）場合に、文が実行されます。

```
if(式){
        文１
}else{
        文２
}
```

この形式では、式が真の時、文１が実行され、式が偽の時、文２が実行されます。

while 文

while 文は次のような形をしています。

```
while（式）{
        文
}
```

while 文では、式が真の間、文が繰り返し実行されます。

※C 言語で使用できる、do{}while(式);の書式は使用できませんので、注意してください。

loop 文

loop 文は次のような形をしています。

```
loop(式){
        文
}
```

loop 文では、式の値の回数だけ、文が実行されます。

※while 文では、繰り返し実行の度に式が評価されますが、loop 文では式の評価は最初の 1 回だけです。

次の２つのスクリプトは、同じ結果になります。

Ex1)

```
x=1;
```

```
loop(10){
        printf("x=%d¥n",x);
        x++;
}
```

Ex2)

```
x=1;
loop(x+9){
        printf("x=%d¥n",x);
        x++;
}
```

switch 文

switch 文は、C 言語の switch とよく似ていますが、C 言語の場合は、式に使用できる型が整数に限られているのに対し、本スクリプトの場合は、整数以外の型でも使用できる点が異なります。

switch 分の書式は次のとおりです。

書式)

```
        switch(式){
        case 値 1:
                //式＝値 1 の場合の処理
                break;
        case 値 2:
                //式＝値 2 の場合の処理
                break;
        default:
                //式がどの case にも一致しない場合の処理
        }
```

使用例)

```
name="suzuki";

switch(name){
case "yamamoto":
    printf("山本");
    break;
case "suzuki":
```

```
    printf("鈴木");
    break;
}
```

システム変数

@で始まる変数は、システム変数です。
システム変数には、次の 2 つがあります。
@ErrorCode　　：エラーコード番号が入ります
@ModalResult　：モーダルダイアログの戻り値が入ります

※システム変数は参照専用ですので、変数への値の代入はしないでください。

システム関数

システム関数は、システム組み込みの内部関数です。内部関数の一覧は、関数リファレンスの章を参照してください。

サブルーチン

本スクリプトでは、ユーザ定義の関数は使用できませんが、サブルーチンを作成することができます。サブルーチンの定義方法は、C 言語の関数の定義とよく似ていますが、戻り値の型指定がない点と、引数がない点が異なります。
次の例は、サブルーチンの記述例です。

サブルーチンの記述例)

```
subfunc(){
    printf(x);
}

main(){
    x="test Start...";
    call subfunc();
}
```

スクリプトは、C言語と同じように、mainサブルーチンから開始されます。サブルーチンの定義は、main より前にあっても、後ろにあっても構いません。
main サブルーチンがない場合は、スクリプトの最初の行から実行が開始されます。

サブルーチンは、関数と異なり、次のような書式となります。

書式）

・サブルーチンの定義

```
サブルーチン名()
{
    :
}
```

・サブルーチンの呼び出し

```
call サブルーチン名();
```

※サブルーチンには、戻り値や引数はありません。また、サブルーチンを使用する場合は、必ず main サブルーチンを用意してください。

内部関数リファレンス

システム関数

システム関数には、システムの標準的な関数が含まれます。

printf

書式１：

printf(式１,式２, 式３・・・) ;

式の値をデバッグウィンドウに出力します。

書式２：

printf(フォーマット文字列,式１,式２, 式３・・・) ;

式をフォーマット付で出力します。フォーマット文字列には、C 言語の printf のフォーマット文字列のうち、d,x,c のみが使用できます。

機能：

printf は、C 言語の printf と同じようなインターフェースを提供します。C 言語と異なり、フォーマットを指定しないで出力することも可能です。

printf の出力先は、出力ウィンドウとなります。

例)

```
printf(x);                    //x の値を、出力ウィンドウに表示します。
printf(x,y);                  //x と y の値を、出力ウィンドウに表示します。
printf("x=%d,y=%d",x,y); //x と y の値を、フォーマットを指定して、出力ウィンドウに表示します。
```

StatusMsg

書式：

StatusMsg(文字列);

戻り値：なし

機能：

ステータスバーに、指定した文字列を表示します。

InputBox

書式１：

str=InputBox(キャプション,プロンプト,デフォルト値)；

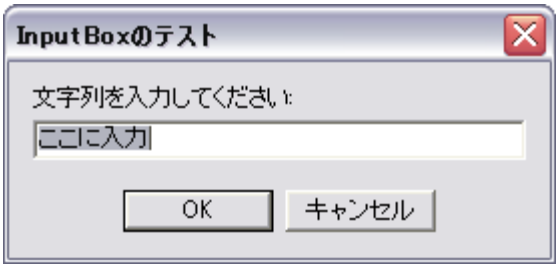
機能：

InputBox は、入力を行いたい場合に使用します。キャプションとプロンプト、デフォルト値が使用できます。入力された文字列を返します。

例)

```
str=InputBox("InputBoxのテスト","文字列を入力してください:","ここに入力");
ShowMsg(str);
```

実行結果)



キャンセルを押した場合は、デフォルト値が返されます。

MsgBox

書式１：

MsgBox (式１,式２, 式３・・・) ;

書式２：

MsgBox(フォーマット文字列,式１,式２, 式３・・・) ;

機能：

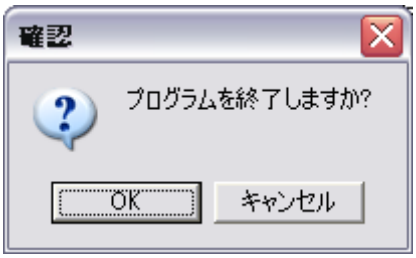
メッセージボックスを表示し、指定した文字列を表示します。形式は printf と同じです。メッセージボックスは、確認用に用いられ、Yes と No のいずれかのボタンを押すとメッセージボックスは終了します。

押されたボタンは、戻り値または@ModalResut で参照することができ、Yes の場合は１、No の場合は０になります。

例)

```
res=MsgBox("プログラムを終了しますか?");
if(res){
    ShowMsg("プログラムを終了します。");
    Exit();
}else{
    ShowMsg("プログラムを続行します。");
}
```

実行結果)



ShowMsg

書式１：

ShowMsg (式１,式２, 式３・・・) ;

書式２：

ShowMsg (フォーマット文字列,式１,式２, 式３・・・) ;

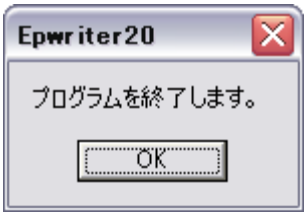
機能：

ShowMsg の機能は、MsgBox とほぼ同じですが、ボタンは OK のみとなります。また、戻り値はありません。

例)

```
ShowMsg("プログラムを終了します。");
```

実行結果)



StopOnError

書式：

StopOnError(式);

式の値：

0:エラーによる停止を行わない

1: エラーによるスクリプトの停止を行なう（デフォルト）

機能：

エラーでスクリプトを停止するかどうかを決定します。

式が真の場合はエラーで停止、偽の場合停止しません。

MediaPlay

書式：

MediaPlay(ファイル名);

機能：

ファイル名で指定されたファイルを指定します。ファイル名で指定できるのは、MediaPlayer で演奏できるファイルです。演奏が終わると、自動で停止します。
エラーのサウンドを鳴らす場合などに使用します。

例）

```
MediaPlay("C:¥WINDOWS¥Media¥chimes.wav");
```

MediaStop

書式：

MediaStop()

機能：

演奏中のメディアファイルを停止します。

例)

```
MediaPlay("C:¥WINDOWS¥Media¥Windows XP Startup.wav");
if(MsgBox("演奏を停止しますか？")){
    MediaStop();
}
```

Exit

書式：

Exit()

機能：

スクリプトを終了します。スクリプトの途中で、スクリプトを強制的に終了したい場合に使用します。

Beep

書式：

Beep(メッセージ番号);

戻り値：なし

機能：

システムビープ音を鳴らします。

メッセージ番号によって、次のビープ音が鳴ります。

メッセージ番号	機能
0	MB_OK
1	MB_ICONHAND
2	MB_ICONEXCLAMATION
3	MB_ICONASTERISK

変数操作関数

Load

書式：
Load(ファイル名);

機能：
データファイルを編集バッファに読み込みます。読み込むファイルの種類は、ファイルの読み込みで使用できる形式（HEX または BIN）で、ファイルの形式は、拡張子で自動的に判断されます。

例）

```
Load("c:¥test.hex");
```

Save

書式：

Save(ファイル名);

機能：

編集バッファの内容を、指定したファイルに書き込みます。書き込むファイルの形式は、拡張子で自動的に判断されます。

例）

```
Save("c:¥test.hex");
```

Store

書式：

Store(変数１,変数２,・・・);

機能：

現在スクリプトで使用している変数を保存します。保存した変数は、内部ファイルに書き込まれますので、PC の電源を切っても、復元することができます。

例）

```
name="suzuki";
Address="Tokyo";
Store(name,address);        //変数nameとaddressを保存
```

Restore

書式：

Restore(変数１,変数２,・・・)；

機能：

Store で保存した変数を読み込みます。

例)

```
Restore(name,address);  //変数nameとaddressを復元
printf(name, address);
```

ClearData

書式：

ClearData();

機能：

Store で保存している変数を、すべてクリアします。

Store で保存したデータは、ClearData()を行わない限り、保存されていますので、不要になった時点で、ClearData()で破棄するようにしてください。

例)

```
Restore(name,address);   //変数nameとaddressを復元
ClearData();             //不要になった保存データを破棄する
printf(name,address);
```

ClearVariable

書式：

ClearVariable();

機能：

変数を、すべてクリアします。

例)

```
Restore(name,address);   //変数nameとaddressを復元
printf(name, address);
ClearVariable();         //変数をすべてクリア
```

文字列操作関数

Left

書式：

Left(文字列,文字数);

戻り値：切り出した文字列

機能：

文字列の先頭から、指定された文字数の文字列を返す。

例)

```
str=Left("abcdefg",4);
printf(str);
```

Right

書式：

Right(文字列,文字数);

戻り値：切り出した文字列

機能：

文字列の右から、指定された文字数の文字列を返す。

例)

```
str=Right("abcdefg",4);
printf(str);
```

Mid

書式：

Mid(文字列,開始位置,文字数);

戻り値：切り出した文字列

機能：

文字列の開始位置から、指定された文字数の文字列を返す。

例)

```
str=Mid("abcdefg",2,4);
printf(str);
```

Len

書式：

Len(文字列);

戻り値：文字列の長さ

機能：

文字列の文字数を返します。

例)

```
printf(Len("abcd"));
```

Trim

書式：

Trim(文字列);

戻り値：前後の空白を削除した文字列

機能：

文字列の前後の空白を削除します。

例)

```
str="  abcd  ";
printf(str+":");
printf(Trim(str)+":");
```

実行結果)

```
  abcd  :
abcd:
```

※最初のprintfでは、スペースがあり、2番目のprintfではスペースが削除されている。

sprintf

書式：

sprintf(フォーマット文字列,式１,式２, 式３・・・)；

戻り値：フォーマット後の文字列

機能：

フォーマット文字列で指定した形式で、式 1 以降の値を文字列にセットして返します。

printf は、出力ウィンドウに文字列を表示しますが、sprintf は、戻り値で文字列を返します。

例)

```
str=sprintf("x=%d",110);
printf(str);
```

Value

書式：

Value(文字列);

戻り値：文字列を数値に変換した値

機能：

文字列を数値として読んだ値を返します。InputBox で入力した文字列を、数値として使用したい場合などに利用します。

例)

```
str="123";
printf(str);
printf(Value(str)*2);
```

Ascii

書式：

Ascii(文字列);

戻り値：最初の文字のアスキーコード

機能：

文字列の先頭文字のアスキーコードを返します。

例)

```
printf(Ascii("A"));
```

Chr

書式：

Chr(ASCII コード);

戻り値：アスキーコードに対応する文字

機能：

アスキーコードに対応する文字を返します。

例)

```
printf(Ascii(65));
```

ファイル操作関数

SelectFile

書式：

SelectFile(ファイル名,ファイル選択文字列);

戻り値：選択ファイル名

機能：

ファイルのオープンダイアログを開き、ファイルを選択します。

オープンされたファイル名が戻り値となります。キャンセルされた場合は、戻り値は、ヌル文字列(“”)となります。ファイル選択文字列は、”ファイル形式名|マスク”の形となります。

例)

```
file=SelectFile("Input File::","Text File(*.txt)|*.txt");
if(file==""){
    Printf("Select File Error!");
    Exit();
}
```

ReadOpen

書式：

ReadOpen(ファイル名);

戻り値：ファイルハンドル

機能：

ファイル名で指定されたファイルを開き、ハンドルを返します。ファイルの入力は、ハンドルを使って行います。

例)

```
//テキストファイルを選択して、出力ウィンドウに表示する
file=SelectFile("Input File::","Text File(*.txt)|*.txt");
if(file==""){
    Printf("Select File Error!");
    Exit();
}

//ファイルのオープン
hd=ReadOpen(file);
if(hd==0){
    Printf("ファイルが開けませんでした。");
    Exit();
}

err=0;
while(err==0){
    err=gets(hd,str);
    if(err==0){
        printf(str);
    }
}

close(hd);
```

WriteOpen

書式：

WriteOpen(ファイル名);

戻り値：ファイルハンドル

機能：

ファイル名で指定されたファイルを開き、ハンドルを返します。ファイルの入力は、ハンドルを使って行います。

例)

```
//ファイルのオープン
hd=WriteOpen("c:¥LogFile.txt");
if(hd==0){
    Printf("ファイルが開けませんでした。");
    Exit();
}
Puts(hd,"-- Log File Start ---");
close(hd);
```

Close

書式：

Close(ファイルハンドル);

機能：

ファイルハンドルで指定したファイルをクローズします。

例）

※ReadOpen の例を参照

Gets

書式：

Gets(ファイルハンドル,取得文字列);

戻り値：

０：正常終了

０以外：エラー

機能：

ファイルハンドルで指定したファイルから 1 行取得します。取得した文字列は、取得文字列の変数に格納されます。エラー発生時には、戻り値が 0 以外の値となります。

例）

※ReadOpen の例を参照

Getc

書式：

Getc(ファイルハンドル,取得文字);

戻り値：

０：正常終了

０以外：エラー

機能：

ファイルハンドルで指定したファイルから 1 文字取得します。取得した文字列、取得文字の変数に格納されます。エラー発生時には、戻り値が 0 以外の値となります。

Puts

書式：

Puts (ファイルハンドル,文字列);

戻り値：

０：正常終了

０以外：エラー

機能：

ファイルハンドルで指定したファイルに、文字列を書き込みます。エラー発生時には、戻り値が 0 以外の値となります。

例）

※WriteOpen の例を参照。

Putc

書式：

Putc (ファイルハンドル,文字);

戻り値：

０：正常終了

０以外：エラー

機能：

ファイルハンドルで指定したファイルに、文字を書き込みます。エラー発生時には、戻り値が 0 以外の値となります。

その他のシステム関数

GetDateTime

書式：

GetDateTime ();

戻り値：現在の日付と時間の文字列

機能：

現在の日付と時間を取得します。日付のみ、あるいは時間のみを取得したい場合は、取得した文字列を、文字列操作関数で切り出します。

例)

```
printf(GetDateTime());
```

実行結果)

2009/02/18 16:28:32

Rand

書式：

Rand();

戻り値：0-32767 までの乱数

機能：

0-32767 までの乱数を発生させます。任意の範囲の乱数を取得したい場合は、下記の例のように、剰余を使ってください。

例）

```
printf(Rand()%6+1);          //1-6 までの乱数を発生させる
```

Delay_ms

書式：

Delay_ms(ミリ秒数);

戻り値：なし

機能：

ミリ秒数で指定した時間のディレイを発生させる。

Delay_s

書式：

Delay_ms(秒数);

戻り値：なし

機能：

秒数で指定した時間のディレイを発生させる。

ライタ制御関数

SelectDLL

書式：
SelectDLL（DLL ファイル名）；
戻り値：なし

機能：
デバイス書き込み用の DLL ファイルを選択します。DLL ファイル名は、Type で指定される”MCW”,”SPI”,”I2C”といった名前です。

例）

```
//"C:¥TestData.HEX"ファイルを読み込んで、デバイスのプログラムを行う
SelectDLL("MCW");
SelectInterface("EPU-20");
SelectMaker("Standerd");
SelectDevice("93C46");
DataMode(1);
MemClear();
ProgramSetting(0,0,0,1,0,1,0,0);
Load("C:¥TestData.HEX");
Program();
```

SelectInterface

書式：

SelectInterface(インターフェース番号);

戻り値：なし

機能：

使用するインターフェースを選択します。

例）

※SelectDLL のサンプルを参照

SelectMaker

書式：
SelectMaker(メーカー名)；
戻り値：なし

機能：
デバイスのメーカーを選択します。

SelectDevice

書式：

SelectDevice(デバイス名)；

戻り値：なし

機能：

デバイスを選択します。

例）

※SelectDLL のサンプルを参照

DeviceSetting

書式：

DeviceSetting(設定文字列);

戻り値：なし

機能：

DLL の SetStting を呼び出します。

例）

※SelectDLL のサンプルを参照

PowerOn

書式：

PowerOn();

戻り値：なし

機能：

現在接続されているライタのターゲットデバイスに電源を入れます。

PowerOff

書式：

PowerOff();

戻り値：なし

機能：

現在接続されているライタのターゲットデバイスの電源を切ります。

SetLED

書式：

SetLED(表示モード);

戻り値：なし

機能：

現在接続されているライタの LED を ON/OFF します。表示モードが 1 で ON、0 の時 OFF になります。

DataMode

書式：

DataMode(モード);

戻り値：なし

機能：

バッファ編集の編集モードを変更します。モードは０の時バイトモード、１の時ワードモードです。

デバイス操作関数

デバイス操作関数は、デバイスの読み書きなど、デバイスに関する操作を行う関数です。

WriteControl

書式：

WriteControl(Cnt);

機能：

DLL の WriteControl を呼び出します。Cnt は WriteControl の引数となります。
詳細は、DLL の仕様書をご確認ください。

Read

書式：

Read(); (1)

Read(adr,size); (2)

戻り値：なし

機能：

デバイスのデータをバッファに読み込みます。

(1)の形式場合は、デバイスのすべてのアドレスを読み込みます。

(2)の形式の場合は、指定されたアドレス(adr)から、指定されたサイズ(size)分を読み込みます。

Write

書式：

Write(); (1)

Write(adr,size); (2)

戻り値：なし

機能：

デバイスにバッファのデータを書き込みます。

(1)の形式場合は、デバイスのすべてのアドレスを読み込みます。

(2)の形式の場合は、指定されたアドレス(adr)から、指定されたサイズ(size)分を書き込みます。

Verify

書式：

Verify(); (1)

Verify(adr,size); (2)

戻り値：なし

機能：

デバイスのデータとバッファのデータを比較します。

(1)の形式場合は、デバイスのすべてのアドレスを比較します。

(2)の形式の場合は、指定されたアドレス(adr)から、指定されたサイズ(size)分を比較します。

Program

書式：

Program();

戻り値：なし

機能：

デバイスのプログラムを実行します。プログラムの処理内容は、アプリケーションの設定に従います。

BlankCheck

書式：

BlankCheck();

戻り値：なし

機能：

デバイスがブランクかどうかをちぇっくします。

GangWrite

書式：

GangWrite();

戻り値：なし

機能：

ギャングプログラマで書き込みを行います。

GangVerify

書式：

GangVerify();

戻り値：なし

機能：

ギャングプログラマでベリファイをおこないます。

GangProgram

書式：

GangProgram();

戻り値：なし

機能：

ギャングプログラマでプログラムをおこないます。

ReadByte

書式：

ReadByte(アドレス,変数)；

戻り値：なし

機能：

デバイスの指定したアドレスのデータを、変数に読み込みます。バイトモードが使用できないデバイスの場合はエラーとなります。

WriteByte

書式：

WriteByte(アドレス,データ)；

戻り値：なし

機能：

デバイスの指定したアドレスに、データを書き込みます。バイトモードが使用できないデバイスの場合はエラーとなります。

ReadWord

書式：

ReadWrod(アドレス,変数)；

戻り値：なし

機能：

デバイスの指定したアドレスのデータを、変数に読み込みます。ワードモードが使用できないデバイスの場合はエラーとなります。

WriteWord

書式：

WriteWord(アドレス,データ)；

戻り値：なし

機能：

デバイスの指定したアドレスに、データを書き込みます。ワードモードが使用できないデバイスの場合はエラーとなります。

消去コマンド

ChipErase

書式：
ChipErase()；
戻り値：なし

機能：
DLL の ChipErase を実行します。

SectorErase

書式：

SectorErase(アドレス)；

戻り値：なし

機能：

DLL の SectorErase を実行します。アドレスは、そのまま DLL の SectorErase の引数となります。

ReadId

書式：

ReadId(ID 格納データ)；

戻り値：なし

機能：

ID が取得可能なデバイスの場合、ID を取得します。ID をサポートしていないデバイスの場合は、リードエラーになります。

例)

```
ReadId(id);
printf(id);
```

GangErase

書式：

GangErase()；

戻り値：なし

機能：

ギャングライタで、消去を実行します。

バッファメモリ操作関数

バッファメモリ操作関数は、編集バッファのメモリを操作します。

Fill

書式：

Fill(アドレス,サイズ、データ);

戻り値：なし

機能：

バッファの指定したアドレス範囲を、指定したデータで埋めます。

MemClear

書式：

MemClear();

戻り値：なし

機能：

編集バッファをクリアして、すべて FFh とします。

MemReadByte

書式：

MemReadByte(アドレス、変数)；

戻り値：なし

機能：

バッファの指定したアドレスの 1 バイトのデータを、変数に読み込みます。

MemWriteByte

書式：

MemWriteByte(アドレス、データ)；

戻り値：なし

機能：

バッファの指定したアドレスに、1 バイトのデータを書き込みます。

MemReadWord

書式：

MemReadWord(アドレス、変数)；

戻り値：なし

機能：

バッファの指定したアドレスの 1 ワードのデータを、変数に読み込みます。

MemWriteWord

書式：

MemWriteWord(アドレス、データ)；

戻り値：なし

機能：

バッファの指定したアドレスに、１ワードのデータを書き込みます。

メニューとフォームの操作関数

PrintPreview

書式：

PrintPreview();

戻り値：なし

機能：

印刷プレビューを表示します。

PrintData

書式：

PrintData();

戻り値：なし

機能：

データを印刷します。

PrintSetting

書式：

PrintSetting();

戻り値：なし

機能：

印刷設定ダイアログを開きます。

About

書式：

About();

戻り値：なし

機能：

情報ダイアログを開きます。

Setting

書式：

Setting ();

戻り値：なし

機能：

設定ダイアログを開きます。

OpenGangProgram

書式：

OpenGangProgram();

戻り値：なし

機能：

ギャングプログラマの画面を開きます。

SerialSetting

書式：

SerialSetting();

戻り値：なし

機能：

シリアル番号設定ダイアログを開きます。

ProgramSetting

書式１：

ProgramSetting();

戻り値：なし

書式２：

ProgramSetting(SerialNo,Erase,BlankCheck,Write,DataSkip,erify,Protect,ProtectLevel);

戻り値：なし

機能：

書式１：プログラム設定ダイアログを開きます。

書式２：プログラム設定の、それぞれの項目の値を設定します。各パラメータが 0 の時は、対応する項目は False, 1 の時は True となります。また ProtectLevel には、プロテクトレベルの値を入れます。

通信制御関数

通信制御関数は、シリアル通信の制御を行います。

SetSpeed

書式：

SetSpeed(Speed);

戻り値：なし

Speed:通信速度

機能：

通信速度の設定を行います。

SetDataBits

書式：

SetDataBit(DBit);

戻り値：なし

DBit:データビット数

機能：

通信パラメータのデータビット幅の設定を行います。

SetStopBit

書式：

SetStopBit(StopBit);

戻り値：なし

StopBit：ストップビット

機能：

通信パラメータのストップビットの設定を行います。

SetParity

書式：

SetParity (Parity);

戻り値：なし

Parity：パリティ

機能：

通信パラメータのパリティの設定を行います。

Parity が 0 なら、パリティなし、1 ならパリティありとなります。

SetFlowType

書式：

SetFlowType (FlowControl);

戻り値：なし

FlowControl：フロー制御

機能：

通信パラメータのフロー制御の設定を行います。

FlowControl の値は、次の表のようになります。

FlowControl の値	機能
0	フロー制御なし
1	RTS/CTS
2	DTR/DSR
3	XON/XOFF

GetDSR

書式：

GetDSR ();

戻り値：

0:DSR は OFF

1:DSR は ON

機能：

DSR の状態を取得します。

GetCTS

書式：

GetCTS ();

戻り値：

0:CTS は OFF

1:CTS は ON

機能：

CTS の状態を取得します。

SetDTR

書式：

SetDTR(設定値);

戻り値：なし

設定値：

0:DTR を OFF にします。

1:DTR を ON にします。

機能：

DTR を設定します。

SetRTS

書式：

SetRTS(設定値);

戻り値：なし

設定値：

0:RTS を OFF にします。

1:RTS を ON にします。

機能：

RTS を設定します。

SetBreak

書式：

SetBreak (設定値);

戻り値：なし

設定値：

0:ブレーク信号を OFF にします。

1: ブレーク信号を ON にします。

機能：

ブレーク信号の ON/OFF を制御します。

SioOpen

書式：

SioOpen(ポート番号);

戻り値：

0:失敗

1:成功

機能：

シリアルポートをオープンします。通信パラメータは、SetLine であらかじめ設定しておきます。SetLine で設定していない場合は、デフォルトの値（9600bps、8 ビット、パリティなし、ストップビット 1 、フロー制御なし）が適用されます。

SioClose

書式：

SioClose();

戻り値：なし

機能：

シリアルポートをクローズします。

SioPutc

書式：

SioPutc(文字);

戻り値：

0:失敗

1:成功

機能：

シリアルポートに、1 文字送信します。

SioPuts

書式：

SioPuts(文字列);

戻り値：

0:失敗

1:成功

機能：

シリアルポートに、文字列を送信します。

SetTimeOut

書式：

SetTimeOut(秒数);

戻り値：

0:失敗

1:成功

機能：

シリアルポートのタイムアウト時間を、指定した秒数に変更します。

SioIsOpen

書式：

SioIsOpen ();

戻り値：

0:オープンされていない

1:オープンされている

機能：

シリアルポートが既にオープンされているかどうかを調べます。

SioWait

書式：

SioWait(str1,str2,...);

戻り値：受信した文字列

機能：

シリアルポートから、指定した文字列を受信するまで待ちます。

特定の文字列が現れると、その文字列を返します。タイムアウトで指定した時間を待っても、指定した文字列が現れない場合は、ヌル文字列(“”)が返ります。

グラフ表示関数

グラフ表示関数は、グラフィックの表示機能を提供します。

ShowGraph

書式:

ShowGraph();

戻り値：なし

機能：

グラフウィンドウを表示します。

Resize

書式:

Resize(width, height) ;

戻り値：なし

width:グラフの描画領域の幅

height:グラフの描画領域の高さ

機能：

描画サイズを変更します。

デフォルトのグラフサイズは、width=320, height=200 です。

GCls

書式:

GCls();

戻り値：なし

機能：

現在のグラフウィンドウを消去します。

GetColor

書式:

GetColor(色名)；

戻り値：色コード

機能：

色の名前から、色コードを取得します。

色コードの名称は、次のものが使用できます。

clAqua, clBlack, clBlue, clCream, clDkGray, clFuchsia, clGray,
clGreen, clLime, clLtGray, clMaroon, clMedGray, clMoneyGreen, clNavy,
clOlive, clPurple, clRed, clSilver, clSkyBlue, clTeal, clWhite, clYellow

GetRGB

書式:

GetRGB(R,G,B)；

戻り値：色コード

機能：

RGB の値(各 0～255)を引数にとり、色コードを取得します。

SetPenColor

書式:

SetPenColor(color)；

戻り値：なし

color:色コードまたは色名

機能：

ペンの描画色を設定します。color には、色コードまたは色名を指定します。色コードは、GetColor や GetRGB で取得することができます。また、引数に色名を直接指定することも可能です。この場合は、以下の色名の文字列を、直接指定します。

指定可能な色（ダブルクォテーションで囲む）：

clAqua, clBlack, clBlue, clCream, clDkGray, clFuchsia, clGray, clGreen,

clLime, clLtGray, clMaroon, clMedGray, clMoneyGreen, clNavy, clOlive,

clPurple, clRed, clSilver, clSkyBlue, clTeal, clWhite, clYellow

SetBrushColor

書式: SetBrushColor(color)；
戻り値：なし
color:色コードまたは色名

機能：
塗りつぶし色を設定します。color には、色コードまたは色名を指定します。色コードは、GetColor や GetRGB で取得することができます。また、引数に色名を直接指定することも可能です。この場合は、以下の色名の文字列を、直接指定します。

指定可能な色（ダブルクォテーションで囲む）：
clAqua, clBlack, clBlue, clCream, clDkGray, clFuchsia, clGray, clGreen,
clLime, clLtGray, clMaroon, clMedGray, clMoneyGreen, clNavy, clOlive,
clPurple, clRed, clSilver, clSkyBlue, clTeal, clWhite, clYellow

MoveTo

書式:

MoveTo(x, y) ;

戻り値：なし

機能：

描画ポインタを、現在の位置から x, y の位置へ移動します。

LineTo

書式:

LineTo(x, y) ;

戻り値：なし

機能：

現在の描画ポインタの位置から、x, y の位置までへ線を引きます。

線の色は、SetPenColor で指定された色になります。また、描画ポインタは、x,y の位置へ移動します。

Circle

書式: Circle(x , y, r,Fill) ;
戻り値：なし

機能：
中心が x,y で、半径 r の円を描きます。Fil1 が 1 なら円内を塗りつぶします。

Box

書式:

Box(x, y, width, height, Fill) ;

戻り値：なし

機能：

x,y の位置に、幅 width, 高さ height の四角形を描きます。Fill が 1 ならば、四角形の内部を塗りつぶします。

SetLineStyle

書式:

SetLineStyle(style)；

戻り値：なし

機能：

線のスタイルを設定します。スタイルは、以下のとおりです。

Style の値	線の種類
0	実線
1	破線
2	点線
3	一点鎖線
4	二点鎖線
5	線は表示されない
6	実線（中間色あり）

※線幅を、１以外の値にした場合は、点線や破線は使用できません。

SetLineWidth

書式:

SetLineWidth(width);

戻り値：なし

width:線の幅

機能：

線の太さをピクセル単位で設定します。線幅を、１以外の値にした場合は、点線や破線は使用できません。

TextOut

書式:

TextOut(x, y, str)；

戻り値：なし

x,y:文字列を表示する座標

str:表示する文字列

機能：

x,y で指定した座標に、文字列を描画します。

Math 関数

Math 関数は数学的な数値処理を行うことができます。

PI

書式:

PI();

機能：

π の値、3.14...を返します。

Exp

書式:

Exp(x);

機能：

定数 e（自然対数）の x 乗を計算して返します。

Log

書式:

Log(x);

機能：

x の自然対数を計算してその値を返します。Exp の逆です。

Log10

書式:

Log10(x);

機能：

x の常用対数(10 を底とした対数)を計算して返します。

Squrt

書式:

Sqrt(x);

機能：

x の平方根を計算して返します。

Pow

書式:

Pow(e, x);

機能：

e の x 乗を計算して返します。

Sin

書式:

Sin(rad);

機能：

ラジアン値(0～２π)を入力し、サイン値を求め、その値を返します。

Cos

書式:

Cos(rad);

機能：

ラジアン値(0～２π)を入力し、コサイン値を求め、その値を返します。

Tan

書式:

Tan(rad);

機能：

ラジアン値(0～２π)を入力し、タンジェント値を求め、その値を返します。

Abs

書式:

Abs(x);

機能：

x の絶対値を求め、その値を返します。

Int

書式:

Int(x);

機能：

x を int 型にした値を返します。

Mod

書式:

Mod(x, y);

機能：

x を y で割った剰余を計算し、その値を返します。