

# **Windows カスタムライブラリ**

## **旧API 説明書**

### **C / C++ 用レギュラーDLL**

### **(AjrCst32.dll / AjrCst64.dll) 編**

( 1. 6. 7. 3 版 )

1.	概 要 .....	4
2.	ダイアログボックス項目／ウインド項目のアクセス .....	5
2.1.1.	ダイアログボックス／ウインド項目の関連付け (AjcSetDlgProp.../AjcSetCtrlProp...)	6
2.1.2.	プロファイルから関連付けられた全ダイアログ／ウインド項目へデータ読み出し (AjcLoadDlgProfiles/AjcLoadCtrlProfiles)	8
2.1.3.	関連付けられた全ダイアログ／ウインド項目のデータをプロファイルへ書き込む (AjcSaveDlgProfiles/AjcSaveCtrlProfiles)	8
2.1.4.	変数の内容を、関連付けられた全ダイアログ／ウインド項目へ設定 (AjcSetDlgValues/AjcSetCtrlValues)	9
2.1.5.	関連付けられた全ダイアログ／ウインド項目の内容を変数へ設定 (AjcGetDlgValues/AjcGetCtrlValues)	9
2.1.6.	関連付けられたダイアログ／ウインド項目のリフレッシュ (AjcRefreshDlgValues/AjcRefreshCtrlValues)	9
2.1.7.	関連付けに使用しているリソースの開放 (AjcReleaseDlgProps/AjcReleaseCtrlProps)	9
2.2.	サンプルプログラム .....	10
2.2.1.	サンプルプログラム 1 .....	10
3.	ウインドサポートAPI .....	13
3.1.	サポートAPI .....	13
3.1.1.	隠れたウインドをタイトルバーを掴める位置まで移動する (AjcMoveWindowToGripped)	13
3.1.2.	隠れたウインドを原点へ移動する - マルチディスプレイ対応 (AjcMoveWindowToGrippedEx)	13
4.	簡易ポップアップメニュー .....	14
4.1.1.	ポップアップメニューの表示 (AjcPopupMenu)	14
5.	C言語ライクなマクロ実行・インタプリタ (AjcCtrlCIP クラス) .....	15
5.1.	マクロテキストファイルの形式 .....	15
5.2.	言語仕様 .....	15
5.2.1.	変数名 .....	15
5.2.2.	数値型変数のタイプ .....	15
5.2.3.	変数の定義 .....	15
5.2.4.	変数のスコープ .....	16
5.2.5.	定義済定数 .....	16
5.2.6.	数式 .....	17
5.2.7.	数値定数 .....	17
5.2.8.	文字列 .....	17
5.2.9.	文字列の代入／比較 .....	18
5.2.10.	配列の代入 .....	18
5.2.11.	処理ブロック .....	19
5.2.12.	break / goto 文 .....	19
5.2.13.	関数の実行 .....	20
5.2.14.	書式文字列 .....	21
5.3.	エスケープシーケンス .....	21
5.3.1.	コメント .....	21
5.3.2.	イベント .....	22
5.3.3.	C言語との相違点 .....	23
5.3.4.	内部関数 .....	24
5.4.	デバッグ機能 .....	43
5.4.1.	トレース表示 .....	43
5.4.2.	ステップトレース .....	44
5.5.	コントロールのスタイル .....	45
5.6.	キャプション文字列によるプロパティの設定 .....	45
5.7.	構造体 .....	45
5.7.1.	引数情報 (AJCCIPAGV) .....	45
5.7.2.	関数呼び出し情報 (AJCCIPCALL) .....	46
5.7.3.	トークン情報 (AJCCIPTKN) .....	46
5.8.	APIの実行順序 .....	47
5.9.	サポートAPI .....	48
5.9.1.	オプションの設定 (AjcCipSetOption) .....	49
5.9.2.	ダブルクリック時の動作設定 (AjcCipSetDb1ClkAct) .....	49
5.9.3.	コード (マクロテキストファイル) の読み出し (AjcCipLoad) .....	49
5.9.4.	コード (マクロテキスト) の実行 (AjcCipExec) .....	50
5.9.5.	ユーザイベント関数の実行 (AjcCipEvent) .....	50
5.9.6.	コード (マクロテキスト) の実行中止 (AjcCipStop) .....	50
5.9.7.	ドロップされたファイル名取得 (AjcCipGetDroppedFile) .....	51
5.9.8.	ドロップされたディレクトリ名取得 (AjcCipGetDroppedDir[Ex]) .....	51

5.9.9.	ウインドへ書式文字列表示 (AjcCipPrintF) .....	51
5.9.10.	ユーザ関数からのエラー通知 (AjcCipNtcErrByUser) .....	52
5.9.11.	_Exit() マクロ関数により指定されたテキストの取得 (AjcCipGetExitText) .....	52
5.9.12.	関数呼び出し情報の初期化 (AjcCipCallInfoInit) .....	53
5.9.13.	関数呼び出し情報の破棄 (AjcCipCallInfoDelete) .....	53
5.9.14.	関数呼び出し情報へ引数 (整数) 追加 (AjcCipCallInfoInsInteger) .....	54
5.9.15.	関数呼び出し情報へ引数 (実数) 追加 (AjcCipCallInfoInsInteger) .....	54
5.9.16.	関数呼び出し情報へ引数 (文字列) 追加 (AjcCipCallInfoInsInteger) .....	54
5.9.17.	関数の戻り値取得 (整数) (AjcCipGetRetInt) .....	55
5.9.18.	関数の戻り値取得 (実数) (AjcCipGetRetReal) .....	55
5.9.19.	関数の戻り値取得 (文字列) (AjcCipGetRetReal) .....	55
5.9.20.	トレース表示 ウインド生成 (AjcCipCreateTraceWindow) .....	55
5.9.21.	トレース表示 ウインド破棄 (AjcCipDestroyTraceWindow) .....	56
5.9.22.	ステップトレース表示 ウインド生成 (AjcCipCreateStepWindow) .....	56
5.9.23.	ステップトレース表示 ウインド破棄 (AjcCipDestroyStepWindow) .....	56
5.9.24.	ステップ・トレースの禁止／許可 (AjcCipEnableStepTrace) .....	56
5.9.25.	VT100 エミュレーションウインドのハンドル取得 (AjcCipGet VT100Handle) .....	57
5.9.26.	ステータス情報取得 (AjcCipGetStatus) .....	57
5.10.	通知情報の取得 A P I .....	58
5.10.1.	外部関数呼び出し情報の取得 (AjcCipGetNtcEvpFun) .....	58
5.10.2.	通知テキストの取得 (AjcCipGetNtcText) .....	58
5.10.3.	ドロップしたディレクトリ数の取得 (AjcCipGetNtcDirs) .....	58
5.10.4.	ドロップしたファイル数の取得 (AjcCipGetNtcFiles) .....	58
5.11.	通知メッセージ .....	59
5.11.1.	外部関数呼び出し通知 (AJCCIPN_FUN) .....	59
5.11.2.	ステップ実行通知 (AJCCIPN_EXE) .....	59
5.11.3.	エラー通知 (AJCCIPN_ERR) .....	60
5.11.4.	ダブルクリック通知 (AJCCIPN_DBLCLK) .....	60
5.11.5.	ファイルドロップ通知 (AJCCIPN_DROPFILE) .....	60
5.11.6.	ディレクトリドロップ通知 (AJCCIPN_DROPDIR) .....	60
5.12.	サンプルプログラム .....	61
5.12.1.	サンプルプログラム 1 .....	61
5.12.2.	サンプルプログラム 2 .....	66

## 1. 概 要

Windows カスタムライブラリ C/C++用レギュラーD L L (AjrCst32.dll / AjrCst64.dll) の旧仕様A P I の説明書です。

これらのA P I は推奨されませんが、D L Lには実装されています。

## 2. ダイアログボックス項目／ウインド項目のアクセス

以下は、旧仕様におけるコントロール設定値の永続化に関するAPIです。

これらのAPIは非推奨としています。

新しく永続化に関する処理を行う場合は、`Ajc{Load/Save}AllControlSettings()` を使用して下さい。

#	関数名	内容
1	<code>AjcSetDlgItemProp ...</code> <code>AjcSetCtrlProp ...</code>	ダイアログボックス／ウインド項目と、変数／プロファイルの関連付け関数群
2	<code>AjcLoadDlgProfiles</code> <code>AjcLoadCtrlProfiles</code>	プロファイルから関連付けられたダイアログ／ウインド項目群へデータ読み出し
3	<code>AjcSaveDlgProfiles</code> <code>AjcSaveCtrlProfiles</code>	関連付けられた全ダイアログ／ウインド項目のデータをプロファイルへ書き込む
4	<code>AjcSetDlgValues</code> <code>AjcSetCtrlValues</code>	変数の内容を関連付けられたダイアログ／ウインド項目群へ設定
5	<code>AjcGetDlgValues</code> <code>AjcGetCtrlValues</code>	関連付けられた全ダイアログ／ウインド項目の内容を変数に設定
6	<code>AjcRefreshDlgValues</code> <code>AjcRefreshCtrlValues</code>	関連付けられた全ダイアログ／ウインド項目のリフレッシュ（数値再表示）
7	<code>AjcReleaseDlgProps</code> <code>AjcReleaseCtrlProps</code>	関連付けに使用しているリソースの開放

## 2.1.1. ダイアログボックス／ウインド項目の関連付け (AjcSetDlgProp.../AjcSetCtrlProp...)

形 式 : ダイアログボックス項目の関連付け

```

BOOL AjcSetDlgPropUInt (HWND hDlg, int id, C_UTP pKey, UI defValue, BOOL fComma, UI *pVar);
BOOL AjcSetDlgPropSInt (HWND hDlg, int id, C_UTP pKey, SI defValue, BOOL fComma, SI *pVar);
BOOL AjcSetDlgPropHex (HWND hDlg, int id, C_UTP pKey, UI defValue, int digits, UI *pVar);
BOOL AjcSetDlgPropReal (HWND hDlg, int id, C_UTP pKey, double defValue, BOOL fComma, int prec, double *pVar);
BOOL AjcSetDlgPropUI8 (HWND hDlg, int id, C_UTP pKey, UB defValue, BOOL fComma, UB *pVar);
BOOL AjcSetDlgPropSI8 (HWND hDlg, int id, C_UTP pKey, SB defValue, BOOL fComma, SB *pVar);
BOOL AjcSetDlgPropH8 (HWND hDlg, int id, C_UTP pKey, UB defValue, int digits, UB *pVar);
BOOL AjcSetDlgPropUI16 (HWND hDlg, int id, C_UTP pKey, UW defValue, BOOL fComma, UW *pVar);
BOOL AjcSetDlgPropSI16 (HWND hDlg, int id, C_UTP pKey, SW defValue, BOOL fComma, SW *pVar);
BOOL AjcSetDlgPropH16 (HWND hDlg, int id, C_UTP pKey, UW defValue, int digits, UW *pVar);
BOOL AjcSetDlgPropUI64 (HWND hDlg, int id, C_UTP pKey, ULL defValue, BOOL fComma, ULL *pVar);
BOOL AjcSetDlgPropSI64 (HWND hDlg, int id, C_UTP pKey, SLL defValue, BOOL fComma, SLL *pVar);
BOOL AjcSetDlgPropH64 (HWND hDlg, int id, C_UTP pKey, ULL defValue, int digits, ULL *pVar);
BOOL AjcSetDlgPropChk (HWND hDlg, int id, C_UTP pKey, UI defState, UI *pVar);
BOOL AjcSetDlgPropStr (HWND hDlg, int id, C_UTP pKey, C_UTP pDefStr, UTP pVar, int lVar);
BOOL AjcSetDlgPropCbo (HWND hDlg, int id, C_UTP pKey, C_UTP pDefLst, UI DefIx, UIP pVar);

```

ウインド項目の関連付け

```

BOOL AjcSetCtrlPropUInt (HWND hwnd, C_UTP pKey, UI defValue, BOOL fComma, UI *pVar);
BOOL AjcSetCtrlPropSInt (HWND hwnd, C_UTP pKey, SI defValue, BOOL fComma, SI *pVar);
BOOL AjcSetCtrlPropHex (HWND hwnd, C_UTP pKey, UI defValue, int digits, UI *pVar);
BOOL AjcSetCtrlPropReal (HWND hwnd, C_UTP pKey, double defValue, BOOL fComma, int prec, double *pVar);
BOOL AjcSetCtrlPropUI8 (HWND hwnd, C_UTP pKey, UB defValue, BOOL fComma, UB *pVar);
BOOL AjcSetCtrlPropSI8 (HWND hwnd, C_UTP pKey, SB defValue, BOOL fComma, SB *pVar);
BOOL AjcSetCtrlPropH8 (HWND hwnd, C_UTP pKey, UB defValue, int digits, UB *pVar);
BOOL AjcSetCtrlPropUI16 (HWND hwnd, C_UTP pKey, UW defValue, BOOL fComma, UW *pVar);
BOOL AjcSetCtrlPropSI16 (HWND hwnd, C_UTP pKey, SW defValue, BOOL fComma, SW *pVar);
BOOL AjcSetCtrlPropH16 (HWND hwnd, C_UTP pKey, UW defValue, int digits, UW *pVar);
BOOL AjcSetCtrlPropUI64 (HWND hwnd, C_UTP pKey, ULL defValue, BOOL fComma, ULL *pVar);
BOOL AjcSetCtrlPropSI64 (HWND hwnd, C_UTP pKey, SLL defValue, BOOL fComma, SLL *pVar);
BOOL AjcSetCtrlPropH64 (HWND hwnd, C_UTP pKey, ULL defValue, int digits, ULL *pVar);
BOOL AjcSetCtrlPropChk (HWND hwnd, C_UTP pKey, UI defState, UI *pVar);
BOOL AjcSetCtrlPropStr (HWND hwnd, C_UTP pKey, C_UTP pDefStr, UTP pVar, int lVar);
BOOL AjcSetCtrlPropCbo (HWND hwnd, C_UTP pKey, C_UTP pDefLst, UI DefIx, UIP pVar);

```

マクロ : ダイアログボックス項目の関連付け

```

BOOL MAjcSetDlgPropUInt (HWND hDlg, int id, UI defValue, BOOL fComma, UI *pVar);
BOOL MAjcSetDlgPropSInt (HWND hDlg, int id, SI defValue, BOOL fComma, SI *pVar);
BOOL MAjcSetDlgPropHex (HWND hDlg, int id, UI defValue, int digits, UI *pVar);
BOOL MAjcSetDlgPropReal (HWND hDlg, int id, double defValue, BOOL fComma, int prec, double *pVar);
BOOL MAjcSetDlgPropUI8 (HWND hDlg, int id, UB defValue, BOOL fComma, UB *pVar);
BOOL MAjcSetDlgPropSI8 (HWND hDlg, int id, SB defValue, BOOL fComma, SB *pVar);
BOOL MAjcSetDlgPropH8 (HWND hDlg, int id, UB defValue, int digits, UB *pVar);
BOOL MAjcSetDlgPropUI16 (HWND hDlg, int id, UW defValue, BOOL fComma, UW *pVar);
BOOL MAjcSetDlgPropSI16 (HWND hDlg, int id, SW defValue, BOOL fComma, SW *pVar);
BOOL MAjcSetDlgPropH16 (HWND hDlg, int id, UW defValue, int digits, UW *pVar);
BOOL MAjcSetDlgPropUI64 (HWND hDlg, int id, ULL defValue, BOOL fComma, ULL *pVar);
BOOL MAjcSetDlgPropSI64 (HWND hDlg, int id, SLL defValue, BOOL fComma, SLL *pVar);
BOOL MAjcSetDlgPropH64 (HWND hDlg, int id, ULL defValue, int digits, ULL *pVar);
BOOL MAjcSetDlgPropChk (HWND hDlg, int id, UI defState, UI *pVar);
BOOL MAjcSetDlgPropStr (HWND hDlg, int id, C_UTP pDefStr, UTP pVar, int lVar);
BOOL MAjcSetDlgPropCbo (HWND hDlg, int id, C_UTP pDefLst, UI DefIx, UIP pVar);

```

引 数 : hDlg - ダイアログボックスのハンドル  
 id - コントロールの識別番号  
 hwnd - コントロールのウインドハンドル  
 pKey - プロファイルに保存する場合のキー名称文字列のアドレス  
 pVar - 関連付ける変数のアドレス (変数との相互変換を行わない場合は不要)  
 lVar - テキストに関連付ける変数の文字数 ( " " )  
 defValue - デフォルト数値 (プロファイルにキーが存在しない場合の仮定値)  
 defState - デフォルトのチェックボックス状態 (プロファイルにキーが存在しない場合の仮定値)  
 pDefStr - デフォルトの文字列のアドレス (プロファイルにキーが存在しない場合に仮定する文字列)  
 pDefLst - デフォルトのコンボボックス項目群のアドレス (多重文字列)  
 DefIx - デフォルトのコンボボックス選択項目のインデクス  
 fComma - 1 0 進数の整数部分を一定の桁数毎に (3 桁毎のカンマ (,) 等で) 区切るか否かのフラグ (TRUE:区切る)  
 digits - 1 6 進数の表示桁数 (0 の場合は桁そろえない)  
 prec - 実数を表示する場合の小数部の桁数

※ pKey/defValue/defState/pDefStr/pDefLst/DefIx は、プロファイルとの相互変換を行わない場合は、指定する必要はありません。(NULL / 0 を指定します)

※ pDefLst=NULL とした場合は、当該コンボボックスにおいて、項目群リスト (各項目のテキスト群) とプロファイルの相互変換は行なわれません。(選択項目インデクスのみ相互変換します)

説 明 : ダイアログボックスの各種コントロールと変数／プロファイルの関連付けを行います。  
 関連付けを行うことにより、ダイアログ項目群と変数／プロファイルの相互変換を一括して行うことができます。

マクロの場合は、pKey 引数 (プロファイルキー) の指定がありません。

マクロを使用した場合は、id 引数の名称がそのままプロファイルキーの名称となります。

例えば、「MAjSetDlgPropUInt(hDlg, IDC\_TXT\_COUNT, 123, TRUE, &nCount);」は、  
 「AjcSetDlgPropUInt(hDlg, IDC\_TXT\_COUNT, "IDC\_TXT\_COUNT", 123, TRUE, &nCount);」と展開されます。

ダイアログボックスをクローズする時には、「AjcReleaseDlgProps() / AjcReleaseCtrlProps()」により、関連付けに使用しているリソースを開放する必要があります。

各関数／マクロで関連付けるデータの種別は、以下のとおりです。

#	関数／マクロ名	データ種別	対象コントロール
1	[M]AjcSet...PropUInt	符号なし 1 0 進数	テキストを持つコントロール (STATIC や EDIT コントロール等)
2	[M]AjcSet...PropSInt	符号付き 1 0 進数	
3	[M]AjcSet...PropReal	実数	
4	[M]AjcSet...PropUI64	符号なし 1 0 進数 (64Bit)	
5	[M]AjcSet...PropSI64	符号付き 1 0 進数 (64Bit)	
6	[M]AjcSet...PropHex	1 6 進数	
7	[M]AjcSet...PropH64	1 6 進数 (64Bit)	
8	[M]AjcSet...PropStr	文字列	
9	[M]AjcSet...PropChk	ボタン状態 (0 : 未チェック, 1 : チェック, 2 : 不定)	チェックボックス ／ラジオボタン
10	[M]AjcSet...PropCbo	コンボボックスの選択項目のインデクス, 設定リスト内容, 設定データ値	コンボボックス

この関数群では、単に関連付けを行うだけであり、実際の相互変換は、以下の関数により実行されます。

#	関数名	内容
1	AjcLoad...Profiles	プロファイルから読み出したデータを関連付けられたダイアログ項目群へ表示
2	AjcSave...Profiles	関連付けられたダイアログ項目群をプロファイルへ記録する
3	AjcGet...Values	変数の内容を関連付けられたダイアログ項目群へ表示する
4	AjcSet...Values	関連付けられたダイアログ項目群の内容を変数へ設定する
5	AjcRefresh...Values	関連付けられたダイアログ項目群をリフレッシュする (数値項目のみ)

戻り値 : TRUE - 成功  
 FALSE - 失敗

**2.1.2. プロファイルから関連付けられた全ダイアログ／ウインド項目ヘデータ読み出し (AjcLoadDlgProfiles／AjcLoadCtrlProfiles)**

**形 式** : BOOL AjcLoadDlgProfiles (HWND hDlg, C\_UTP pSec);  
 BOOL AjcLoadCtrlProfiles (HWND hwnd, C\_UTP pSec);

**引 数** : hDlg - ダイアログボックスのハンドル  
 hwnd - コントロール群の親ウインドハンドル  
 pSec - プロファイルのセクション名文字列のアドレス

**説 明** : プロファイルからデータを読み出し、AjcSet…PropXXX() で関連付けられた全てのダイアログ項目へ設定します。  
 pSec でプロファイル・セクション名を指定し、キー名称は、各 AjcSet…PropXXX() の pKey で指定された名称となります。  
 プロファイル中に、指定されたセクション名／キー名が無い場合は、AjcSetPropXXX() で指定されたデフォルト値が設定されます。

AjcSet…PropCbo() で関連付けを行っている場合、プロファイルからリスト項目群のテキスト情報、各項目に関連付けられたデータ値と選択項目のインデクス値を読み出します。

**戻り値** : TRUE - 成功  
 FALSE - 失敗

**2.1.3. 関連付けられた全ダイアログ／ウインド項目のデータをプロファイルへ書き込む (AjcSaveDlgProfiles／AjcSaveCtrlProfiles)**

**形 式** : BOOL AjcSaveDlgProfiles (HWND hDlg, C\_UTP pSec);  
 BOOL AjcSaveCtrlProfiles (HWND hwnd, C\_UTP pSec);

**引 数** : hDlg - ダイアログボックスのハンドル  
 hwnd - コントロール群の親ウインドハンドル  
 pSec - プロファイルのセクション名文字列のアドレス

**説 明** : AjcSet…PropXXX() で関連付けられた、全てのダイアログ／ウインド項目の内容をプロファイルへ記録します。  
 pSec でプロファイル・セクション名を指定し、キー名称は、各 AjcSet…PropXXX() の pKey で指定された名称となります。

AjcSet…PropStr() で関連付けを行っている場合、以下のようにプロファイルへ記録します。

<i>KeyName=XXXXXX</i> ---- 文字列 <i>KeyName_Stl=nnn</i> ---- 文字列の長さ+1の値
--

※「*KeyName*」は、AjcSet…PropStr() の pKey 引数で指定した名称

AjcSet…PropCbo() で関連付けを行っている場合、プロファイルへリスト項目群のテキスト情報、各項目に関連付けられたデータ値と現在選択されている項目のインデクス値を記録します。

プロファイル上の記録形式は、「AjcSave \*\* ComboBox (コンボボックスの内容をプロファイルへ書き込む)」と同じです。

**戻り値** : TRUE - 成功  
 FALSE - 失敗



**2.1.4. 変数の内容を、関連付けられた全ダイアログ／ウインド項目へ設定 (AjcSetDlgValues／AjcSetCtrlValues)**

**形 式** : BOOL AjcSetDlgValues (HWND hDlg);  
 BOOL AjcSetCtrlValues (HWND hwnd);

**引 数** : hDlg - ダイアログボックスのハンドル  
 hwnd - コントロール群の親ウインドハンドル

**説 明** : AjcSetPropXXX() で関連付けられた全てのダイアログ／ウインド項目へ、関連付けられている変数の内容を設定します。  
 AjcSetPropCbo() で関連付けられたコンボボックスでは、関連付けられた変数の値 (インデクス) で項目を選択状態にします。

**戻り値** : TRUE - 成功  
 FALSE - 失敗

**2.1.5. 関連付けられた全ダイアログ／ウインド項目の内容を変数へ設定(AjcGetDlgValues／AjcGetCtrlValues)**

**形 式** : BOOL AjcGetDlgValues (HWND hDlg);  
 BOOL AjcGetCtrlValues (HWND hwnd);

**引 数** : hDlg - ダイアログボックスのハンドル  
 hwnd - コントロール群の親ウインドハンドル

**説 明** : AjcSetPropXXX() で関連付けられた全てのダイアログ／ウインド項目の内容を、関連付けられている変数へ設定します。  
 AjcSetPropCbo() で関連付けられたコンボボックスでは、現在選択されている項目のインデクス値が変数に設定されます。

**戻り値** : TRUE - 成功  
 FALSE - 失敗

**2.1.6. 関連付けられたダイアログ／ウインド項目のリフレッシュ(AjcRefreshDlgValues／AjcRefreshCtrlValues)**

**形 式** : BOOL AjcRefreshDlgValues (HWND hDlg);  
 BOOL AjcRefreshCtrlValues (HWND hwnd);

**引 数** : hDlg - ダイアログボックスのハンドル  
 hwnd - コントロール群の親ウインドハンドル

**説 明** : AjcSetPropUInt/SInt/Hex/Real/UI64/SI64/H64 で関連付けられた全てのダイアログ／ウインド項目の内容を、リフレッシュします。つまり、各ダイアログボックス／ウインド項目の内容を、関連付けられたタイプの数値として読み出し、この数値を再度表示します。  
 これにより、各ダイアログボックス／ウインド項目では、関連付けられたタイプの数値として認識されない文字が排除されます。  
 例えば、AjcSetDlgPropUInt() により関連付けられたダイアログ項目に「123XYZ」というテキストが設定されている場合、このテキストは「123」に変更されます。

**戻り値** : TRUE - 成功  
 FALSE - 失敗

**2.1.7. 関連付けに使用しているリソースの開放 (AjcReleaseDlgProps／AjcReleaseCtrlProps)**

**形 式** : BOOL AjcReleaseDlgProps (HWND hDlg);  
 BOOL AjcReleaseCtrlProps (HWND hwnd);

**引 数** : hDlg - ダイアログボックスのハンドル  
 hwnd - コントロール群の親ウインドハンドル

**説 明** : AjcSetDlgPropXXX() / AjcSetCtrlPropXXX() で関連付の為に確保されたリソースを開放します。

**戻り値** : TRUE - 成功  
 FALSE - 失敗

## 2.2. サンプルプログラム

### 2.2.1. サンプルプログラム 1

以下のサンプルコードでは、ダイアログボックスの起動時に、プロファイルからデータを読み出して、ダイアログの各項目に設定します。(初回実行時等で) プロファイルにデータが無い場合は、MAJcSetDlgProp\_XXXで指定したデフォルト値を表示します。また、ダイアログの各項目の内容を変数にも設定します。

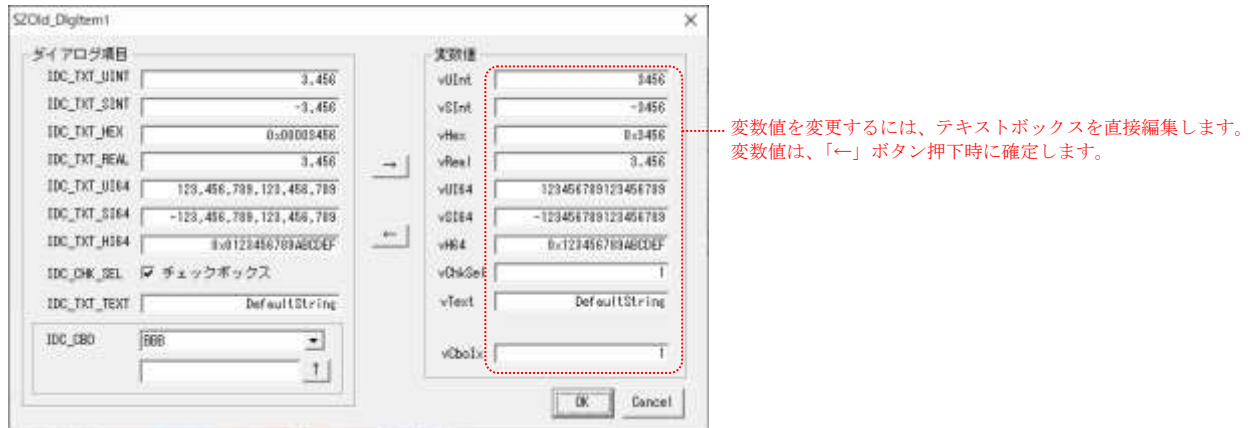
「→」 ボタンを押すと、ダイアログ項目の内容を変数に設定します。

「←」 ボタンを押すと、変数の内容をダイアログ項目に設定します。

「OK」 ボタンを押すと、ダイアログボックスの各項目の内容をプロファイルに記録して、ダイアログを終了します。

各ダイアログ項目の操作 (テキスト入力や、コンボボックス・リスト選択等) により変更された内容は、そのままプロファイルに記録され、次回起動時のダイアログボックスに再現されます。

「Cancel」 ボタンを押すと、ダイアログボックスの各項目の内容をプロファイルに記録しないでプログラムを終了します。



```

1 : //
2 : // SZOld_DlgItem1.c
3 : //
4 : #include <AjrCstXX.h>
5 : #include <math.h>
6 : #include <tchar.h>
7 : #include "resource.h"
8 :
9 : //-----//
10 : // ワーク
11 : //-----//
12 : HINSTANCE hInst;
13 : HWND hDlgMain;
14 :
15 : // ダイアログ項目と関連付ける変数
16 : UI vUInt, vHex, vChkSel;
17 : SI vSInt;
18 : double vReal;
19 : ULL vUI64, vH64;
20 : SLL vSI64;
21 : UT vText[512];
22 : UI vCboIx;
23 :
24 : //-----//
25 : // 内部サブ関数
26 : //-----//
27 : AJC_DLGPROC_DEF(Main);
28 : static VO ShowVariables(HWND hDlg);
29 :
30 : //=====//
31 : //
32 : // WinMain
33 : //
34 : //=====//
35 : int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int iCmdShow)
36 : {
37 :     MSG msg;
38 :
39 :     hInst = hInstance;
40 :

```

```

41 : //----- メイン・ダイアログオープン -----//
42 : hDlgMain = CreateDialog(hInst, MAKEINTRESOURCE(IDD_MAIN), NULL, AJC_DLGPROC_NAME(Main));
43 : ShowWindow(hDlgMain, SW_SHOW);
44 :
45 : //----- メッセージループ -----//
46 : while (GetMessage(&msg, NULL, 0, 0)) {
47 :     do {
48 :         if (IsDialogMessage(hDlgMain, &msg)) break;
49 :         TranslateMessage(&msg);
50 :         DispatchMessage (&msg);
51 :     } while (0);
52 : }
53 :
54 : return (int)msg.wParam ;
55 : }
56 : //=====//
57 : //
58 : // ダイアログ・プロシージャ
59 : //
60 : //=====//
61 : //----- ダイアログ初期化 -----//
62 : AJC_DLGPROC(Main, WM_INITDIALOG )
63 : {
64 :     hDlgMain = hDlg;
65 :
66 :     // チップテキスト設定
67 :     AjcTipTextAdd(GetDlgItem(hDlg, IDC_CMD_SETVAL), TEXT("ダイアログ項目を変数に設定"));
68 :     AjcTipTextAdd(GetDlgItem(hDlg, IDC_CMD_SETITEM), TEXT("変数値をダイアログ項目に設定"));
69 :     AjcTipTextAdd(GetDlgItem(hDlg, IDC_CMD_ADDCBO), TEXT("コンボボックスへ項目追加"));
70 :     AjcTipTextAdd(GetDlgItem(hDlg, IDC_TXT_CBOITEM), TEXT("コンボボックスへ追加するテキスト"));
71 :     AjcTipTextAdd(GetDlgItem(hDlg, IDOK), TEXT("ダイアログ項目をプロファイルに記録して終了"));
72 :     AjcTipTextAdd(GetDlgItem(hDlg, IDCANCEL), TEXT("ダイアログ項目をプロファイルに記録しないで終了"));
73 :     // ダイアログ項目と変数を関連付ける
74 :     MAjcSetDlgPropUInt(hDlg, IDC_TXT_UINT, 3456, TRUE, &vUInt);
75 :     MAjcSetDlgPropSInt(hDlg, IDC_TXT_SINT, -3456, TRUE, &vSInt);
76 :     MAjcSetDlgPropHex(hDlg, IDC_TXT_HEX, 0x3456, 8, &vHex);
77 :     MAjcSetDlgPropReal(hDlg, IDC_TXT_REAL, 3.456, TRUE, 3, &vReal);
78 :     MAjcSetDlgPropUI64(hDlg, IDC_TXT_UI64, 123456789123456789, TRUE, &vUI64);
79 :     MAjcSetDlgPropSI64(hDlg, IDC_TXT_SI64, -123456789123456789, TRUE, &vSI64);
80 :     MAjcSetDlgPropH64(hDlg, IDC_TXT_H64, 0x0123456789ABCDEF, 16, &vH64);
81 :     MAjcSetDlgPropChk(hDlg, IDC_CHK_SEL, TRUE, &vChkSel);
82 :     MAjcSetDlgPropStr(hDlg, IDC_TXT_TEXT, _T("DefaultString"), vText, 512);
83 :     MAjcSetDlgPropCbo(hDlg, IDC_CBO, _T("AAA¥0")_T("BBB¥0")_T("CCC¥0"), 1, &vCboIx);
84 :     // プロファイルから読み出した値をダイアログ項目群へ設定
85 :     AjcLoadDlgProfiles(hDlg, TEXT("MySect"));
86 :     // ダイアログ項目群の内容を変数へ設定
87 :     AjcGetDlgValues(hDlg);
88 :     // 変数値表示
89 :     ShowVariables(hDlg);
90 :
91 :     return TRUE;
92 : }
93 : //----- ウインド破棄 -----//
94 : AJC_DLGPROC(Main, WM_DESTROY )
95 : {
96 :     // ダイアログ項目関連付けリソース解放
97 :     AjcReleaseDlgProps(hDlg);
98 :
99 :     PostQuitMessage(0);
100 :    return TRUE;
101 : }
102 : //----- コンボボックス項目を変数へ設定 (「→」ボタン) -----//
103 : AJC_DLGPROC(Main, IDC_CMD_SETVAL )
104 : {
105 :     if (HIWORD(wParam) == BN_CLICKED) {
106 :         AjcGetDlgValues(hDlg);
107 :         ShowVariables(hDlg);
108 :     }
109 :     return TRUE;
110 : }
111 : //----- 変数値をコンボボックス項目へ設定 (「←」ボタン) -----//
112 : #define SETINT(ID, VAR, FUN) AjcSetDlgItem##FUN(hDlg, ID, VAR = AjcGetDlgItem##FUN(hDlg, ID))
113 : #define SETREL(ID, VAR, FUN) AjcSetDlgItem##FUN(hDlg, ID, VAR = AjcGetDlgItem##FUN(hDlg, ID), 3)
114 :
115 : AJC_DLGPROC(Main, IDC_CMD_SETITEM )
116 : {
117 :     if (HIWORD(wParam) == BN_CLICKED) {
118 :         // 変数値設定
119 :         SETINT(IDC_TXT_VUINT, vUInt, UInt);
120 :         SETINT(IDC_TXT_VSINT, vSInt, SInt);

```

```

121 :         SETINT(IDC_TXT_VHEX , vHex , Hex );
122 :         SETREL(IDC_TXT_VREAL , vReal , Real);
123 :         SETINT(IDC_TXT_VUI64 , vUI64 , UI64);
124 :         SETINT(IDC_TXT_VSI64 , vSI64 , SI64);
125 :         SETINT(IDC_TXT_VH64 , vH64 , H64 );
126 :         SETINT(IDC_TXT_VCHKSEL, vChkSel, UInt);
127 :         AjcGetDlgItemStr(hDlg, IDC_TXT_VTEXT, vText, AJCTSIZE(vText));
128 :         SETINT(IDC_TXT_VCBOIX, vCboIx, UInt);
129 :         // 変数をコンボボックス項目へ設定
130 :         AjcSetDlgValues(hDlg);
131 :     }
132 :     return TRUE;
133 : }
134 : //----- コンボボックスへ項目追加 (「↑」ボタン) -----//
135 : AJC_DLGPROC(Main, IDC_CMD_ADDCBO )
136 : {
137 :     if (HIWORD(wParam) == BN_CLICKED) {
138 :         UT txt[256];
139 :         AjcGetDlgItemStr(hDlg, IDC_TXT_CBOITEM, txt, AJCTSIZE(txt));
140 :         AjcSetDlgItemCboAdd(hDlg, IDC_CBO, 0, txt, AJCCBF_SORT);
141 :     }
142 :     return TRUE;
143 : }
144 : //----- OK -----//
145 : AJC_DLGPROC(Main, IDOK )
146 : {
147 :     // ダイアログ項目群の値をプロファイルへ記録する
148 :     AjcSaveDlgProfiles(hDlg, TEXT("MySect"));
149 :
150 :     DestroyWindow(hDlg);
151 :     return TRUE;
152 : }
153 : //----- キャンセル -----//
154 : AJC_DLGPROC(Main, IDCANCEL )
155 : {
156 :     DestroyWindow(hDlg);
157 :     return TRUE;
158 : }
159 : //-----//
160 : AJC_DLGMAP_DEF(Main)
161 :     AJC_DLGMAP_MSG(Main, WM_INITDIALOG )
162 :     AJC_DLGMAP_MSG(Main, WM_DESTROY )
163 :
164 :     AJC_DLGMAP_CMD(Main, IDC_CMD_SETVAL )
165 :     AJC_DLGMAP_CMD(Main, IDC_CMD_SETITEM)
166 :     AJC_DLGMAP_CMD(Main, IDC_CMD_ADDCBO )
167 :     AJC_DLGMAP_CMD(Main, IDOK )
168 :     AJC_DLGMAP_CMD(Main, IDCANCEL )
169 : AJC_DLGMAP_END
170 :
171 : //-----//
172 : // 変数の値表示 //
173 : //-----//
174 : static VO ShowVariables(HWND hDlg)
175 : {
176 :     AjcSetDlgItemUInt(hDlg, IDC_TXT_VUINT , vUInt);
177 :     AjcSetDlgItemSInt(hDlg, IDC_TXT_VSINT , vSInt);
178 :     AjcSetDlgItemHex (hDlg, IDC_TXT_VHEX , vHex );
179 :     AjcSetDlgItemReal(hDlg, IDC_TXT_VREAL , vReal, 3);
180 :     AjcSetDlgItemUI64(hDlg, IDC_TXT_VUI64 , vUI64);
181 :     AjcSetDlgItemSI64(hDlg, IDC_TXT_VSI64 , vSI64);
182 :     AjcSetDlgItemH64 (hDlg, IDC_TXT_VH64 , vH64);
183 :     AjcSetDlgItemUInt(hDlg, IDC_TXT_VCHKSEL, vChkSel);
184 :     AjcSetDlgItemStr (hDlg, IDC_TXT_VTEXT , vText);
185 :     AjcSetDlgItemUInt(hDlg, IDC_TXT_VCBOIX , vCboIx);
186 : }

```

### 3. ウインドサポートAPI

ウインドのアクセスや操作をサポートするAPIの集合です。

#### 3.1. サポートAPI

ウインドサポートAPIの一覧を以下に示します。

#	関数名	内容
1	AjcMoveWindowToGripped	隠れたウインドをタイトルバーを掴める位置まで移動する
2	AjcMoveWindowToGrippedEx	隠れたウインドを原点へ移動する（マルチディスプレイ対応）

##### 3.1.1. 隠れたウインドをタイトルバーを掴める位置まで移動する (AjcMoveWindowToGripped)

形 式 : BOOL AjcMoveWindowToGripped(HWND hwnd);

引 数 : hwnd - ウインドハンドル

説 明 : ウインドのタイトルバーがプライマリディスプレイのデスクトップ作業域／親ウインド内クライアント領域上に見えない位置に存在する場合、当該ウインドをタイトルバーが見える位置まで移動します。  
つまり、タイトルバーを掴めなくなってしまったウインドを、タイトルバーを掴める位置まで移動します。

戻り値 : TRUE - 成功  
FALSE - 失敗

備 考 : このAPIはマルチウインドに対応していません。  
通常は、AjcMoveWindowToGrippedEx()を使用してください。

##### 3.1.2. 隠れたウインドを原点へ移動する - マルチディスプレイ対応 (AjcMoveWindowToGrippedEx)

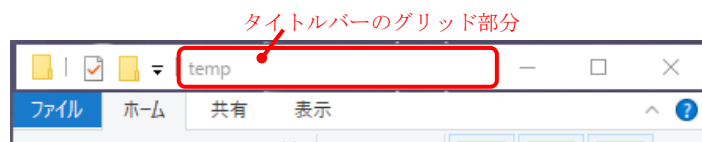
形 式 : BOOL AjcMoveWindowToGrippedEx(HWND hwnd, UI ratio);

引 数 : hwnd - ウインドハンドル  
ratio - ウインドのタイトルバー部分の割合（%）

説 明 : ウインドのタイトルバーのグリッド部分がマルチディスプレイの各作業域上に見えない（あるいは、僅かしか見えない）位置に存在する場合、当該ウインドを原点（0, 0）へ移動します。  
「見えない位置」とは、指定したウインドでタイトルバーのグリッド部分が、マルチディスプレイ上において、ratioで指定した割合以下しか見えないことを意味します。  
つまり、タイトルバーを掴めなくなってしまった（あるいは掴みずらくなった）ウインドを、原点へ移動します。

戻り値 : TRUE - 成功  
FALSE - 失敗

備 考 : 「作業領域」とは、ディスプレイ中のタスクバー等を除いた、アプリケーションが使用できる部分を意味します。  
「タイトルバーのグリッド部分」とは、ウインドを掴んで移動可能な、タイトルバー中の部分を意味します。（下図参照）



## 4. 簡易ポップアップメニュー

簡易的なポップアップメニュー機能です。

本ポップアップメニューは、処理がブロックされず、メニュー選択中でも処理を続行できるようになります。

メニューの選択結果は、メニューが選択された時に、指定されたウインドへメッセージの送信により通知します。

ポップアップメニューの外観は、Windows 標準のポップアップメニューとほぼ同じです。

### 4.1.1. ポップアップメニューの表示(AjcPopupMenu)

**形 式** : HWND AjcPopupMenu (HWND hWndOwner, UI msg, int x, int y, PAJCPPMTBLA pTbl, LPARAM lParam);

**引 数** :

- hWndOwner - オーナーウインドハンドル
- msg - メニュー選択メッセージコード
- x, y - ポップアップメニュー表示位置 (左上位置)
- pTbl - メニューテーブル
- lParam - メニュー選択メッセージ発行時のパラメタ

**説 明** : ポップアップメニューウインドを表示します。

ポップアップメニューウインドを表示したまま、メニューの選択を待たずに制御を返します。

後で、ポップアップメニューが選択された場合は、hWndOwner で指定されたウインドへ、msg で指定されたメッセージをポストします。

このメッセージのパラメタは以下のとおりです。

wParam - メニュー I D (メニューテーブルの「id」の値)  
lParam - 本 A P I の引数「lParam」と同じ値

ポップアップメニュー以外の部分をクリックした場合は、ポップアップメニューがキャンセルされます。

pTbl は、メニュー項目群を、以下の構造体の配列で指定します。

```
typedef struct {
    int     flag;           // フラグ情報 (AJCPPMF_XXXXX)
    int     id;             // メニュー I D
    int     key;            // ショートカットキー (0 固定, 内部ワーク)
    UTP     pText;          // メニューテキストへのポインタ
} AJCPPMTBL, *PAJCPPMTBL;
```

flag=AJCPPMF\_SEPARATE を設定した場合、当該メニュー項目と次のメニュー項目との間に境界線を引きます。

配列の最後は、flag=AJCPPMF\_EOT を設定します。(最後の項目にはメニューテキストを指定しません)

その他の項目 (メニューテキスト項目) は、flag=0 とします。

「id」は、メニュー項目の識別番号です。

pText はメニューテキストへのポインタを指定します。

ショートカットキーを指定する場合は、テキスト中の当該文字の前にアンパサンド(&)をおきます。

例えば、編集メニュー項目のショートカットキーを “E” とする場合は、“編集(&E)” というように指定します。

pTbl で指定したメニュー項目群の配列は、本関数から戻ったら破棄してもかまいません。(保持しておく必要はありません)

**戻り値** : ≠NULL - 正常 (ポップアップメニューウインドのハンドル)  
=NULL - 失敗

**使用例** :

```
AJCPPMTBL Tbl[] = {
    {0, 10, 0, TEXT("AAA")},
    {AJCPPMF_SEPARATE, 15, 0, TEXT("BBB")},
    {0, 20, 0, TEXT("CCC")},
    {AJCPPMF_EOT},
};

AjcPopupMenu(hDlg, WM_APP, 100, 100, Tbl, 0);
```



## 5. C 言語ライクなマクロ実行・インタプリタ (AjcCtrlCIP クラス)

C 言語ライクな言語仕様によるインタプリタ実行エンジンです。

プログラムファイル (マクロテキストファイル) を入力して、そのまま実行することができます。

### 5.1. マクロテキストファイルの形式

マクロテキストファイルは、シフト JIS コード・ファイルでなければなりません。

(他国のマルチバイトもロケールの設定に従うため、正常に動作するかとは思いますが、現状は未テストです)  
尚、UNICODE ファイルはサポートしていません。

### 5.2. 言語仕様

本インタプリタは、C 言語ライクな言語仕様となっていますが、C 言語との相違点もあります。

以下、本インタプリタの言語仕様を述べます。(C 言語と同じ部分については、詳細な説明は割愛しています)

#### 5.2.1. 変数名

変数名は、先頭が「半角英字, 全角文字, アンダバー(\_) or ドルマーク(\$)」で始まり、後続が「先頭文字の条件」+「半角数字」で構成します。

先頭が「\$」で始まる変数は文字変数、それ以外は数値変数となります。

C 言語と同様に変数名の英字は、大文字/小文字を区別します。

以下の変数名は、いずれも有効な変数名です。

```
(例)    define var = 10, VAR, _abc09, $str1;
        define 東京都港区赤坂_10_2_102;
        define $私は誰でしょう = “文字変数です”;
        define 3教科点数[3] = {82, 75, 96};
```

#### 5.2.2. 数値型変数のタイプ

先頭がドルマーク(\$)以外の変数は、数値型変数として扱われますが、そのタイプは「符号付き整数」「符号無し整数」あるいは、「実数」となります。

どの型にするかは、マクロテキストの先頭で「option」文により指定します。

```
(例)    option signed; ----- 数値型を符号付き整数として扱う (option 指定無し時のデフォルト)
        option unsigned; ----- 数値型を符号無し整数として扱う
        option double; ----- 数値型を実数として扱う
```

整数は C 言語の「signed long long」「unsigned long long」と同じです。(64 ビット精度)

実数は C 言語の「double」と同じです。(64 ビット精度)

※マクロテキスト先頭の「option」文以外に、AjcCipSetOption() API により設定する方法もあります。

#### 5.2.3. 変数の定義

変数は、「define」文により、以下の構文で定義します。

#	構文	内容	
1	define 変数名;	単純変数 (変数名[1]; と同じ)	単純変数
2	define 変数名 = 値;	初期値付き単純変数 (変数名[1] = 値; と同じ)	
3	define 変数名[要素数];	配列変数	配列変数
4	define 変数名[要素数] = 値;	初期値付き配列変数 (先頭要素のみ初期化)	
5	define 変数名[要素数] = {値1, 値2, ..., 値n};	初期値付き配列変数 (複数の要素を初期化)	
6	define 変数名[] = 値;	初期値付き配列変数 (要素数 = 1)	
7	define 変数名[] = {値1, 値2, ..., 値n};	初期値付き配列変数 (要素数 = 初期値の数)	

※ カンマ(,)で区切って複数の変数を定義可能です。(ex. define a, b=21, c[2]= {10, 20};)

※ 初期値が指定されていない場合、数値変数はゼロ (0) で、文字変数は空文字列(“”) で初期化されます。

※ 「要素数」や「初期値」は、一般の数式で指定可能です。

(ex. define var[a \* 3 + b] = {a + b, a - b}; // a, b は先に定義されている必要があります)

※ 単純変数は、実際は要素数 = 1 の配列となります。(「define x;」 と 「define x[1];」 は同じです)

※ 変数参照時に添え字([n])を付けない場合、変数名[0]と同じです。(「var = 1;」と「var[0] = 1;」は同じです)

### 5.2.4. 変数のスコープ

C 言語と同様に、関数の外で定義された変数は、グローバルなスコープを持ちます。(ソース全体から参照可能)

関数の中で定義された変数は、C 言語の自動変数と同様に、当該関数内のスコープを持ちます。(当該関数内でのみ参照可能)

但し、関数内のブラケット ( { . . . } ) 内に定義した自動変数も、関数全体のスコープとなります。

グローバルな変数名と関数内の自動変数名が同じ場合、自動変数がアクセスされ、グローバルな変数はアクセスできません。

```
(例)  define gbl;           // 広域変数 (いずれの関数からも参照可)
      define mult = 10;    // 広域変数 (関数内で同名の自動変数が定義された場合は参照不可)
      function func(a, b)
      {
          define v1, v2;   // 自動変数
          define mult = 20; // 自動変数 (広域変数と同名のため、この自動変数のみ参照可)
          if ( . . . ) {
              define inb = 1; // 「inb」も関数内全体のスコープを持つ自動変数となる
              . . .
          }
          v1 = inb;         // 「inb」は上記の「if { . . . }」ブラケット内で定義されているが
          gbl = inb;        // 関数内全体のスコープを持つため、参照可
          v2 = mult;        // 自動変数の「mult」が参照され、v2=20 となります。
      }
```

関数内の「define」は、「自動変数を生成する実行命令」です。手続き上同じ「define」が2度実行されると、二重定義エラーとなります。以下のコードは「define a;」が2度実行されるため、実行時エラーとなります。

```
(例)  for (i = 0; i < 2; i++) {
      define a;    // 2回目の実行で二重定義エラーとなる
      . . .
  }
```

※ 二重定義エラーを防止するために、自動変数は関数の先頭でまとめて定義しておくことをお勧めします。

### 5.2.5. 定義済定数

定義済定数一覧を以下に示します。

#	名称	値	備考
1	_TRUE	1	真値
2	_FALSE	0	偽値
3	_CR	0x0D	復帰コード
4	_LF	0x0A	改行コード
5	_SLOT_PORT	256	メールスロットのポート番号
6	_MB_OK	0	_MessageBox() の BoxType 引数
7	_MB_OKCANCEL	1	
8	_MB_RETRYCANCEL	5	
9	_MB_YESNO	4	
10	_MB_YESNOCANCEL	3	
11	_MB_ICONERROR	16	_MessageBox() の戻り値
12	_IDCANCEL	2	
13	_IDNO	7	
14	_IDOK	1	
15	_IDYES	6	_CalcCrc() の knd 引数
16	_CRC_ITL	0	
17	_CRC_ITR	1	
18	_CRC_16L	2	
19	_CRC_16R	3	復帰・改行
20	\$_CR	"¥r"	
21	\$_LF	"¥n"	
22	\$_CRLF	"¥r¥n"	復帰・改行
23	\$_MyPath	マクロテキストのパス名部分	マクロのパス名部分 (「D:¥work¥Mac.txt」 → " D:¥work¥" )
24	\$_MyName	マクロテキストファイル名	マクロのファイル名部分 (「D:¥work¥Mac.txt」 → " Mac" )



### 5.2.6. 数式

数式の表現は、ほぼC言語と同じです。(但し、文字列の代入／比較を記述できる等、若干の違いがあります)  
演算子の種類と優先順位は、以下の通りです。

#	演算子	名称	備考	優先度
1	( . . . )	優先計算部	「 . . . 」は数式	<div>高</div> <div>↑</div> <div>↓</div> <div>低</div>
2	= ==, !=, <, >, <=, >=	文字列の代入 文字列の比較	文字列の代入／比較は高優先 ex. \$s[2]=" AB"    \$s < \$t	
3	++ -- +, -, ! ~	プリ・インクリメント, ポスト・インクリメント プリ・デクリメント , ポスト・デクリメント 単項演算子 (プラス, マイナス, 論理否定) 単項演算子 (ビット反転)	ex.    ++x    y++ ex.    --x    y-- ex,    !x    -10 実数の場合、使用不可	
4	*, /, %	乗算, 除算, 剰余算		
5	+, -	加算, 減算		
6	<<, >>	左シフト, 右シフト	実数の場合、2 <sup>n</sup> で乗算／除算	
7	<, >, <=, >=	大小比較	数値の比較	
8	==, !=	等値, 不等値比較		
9	&	ビット論理積	実数の場合、使用不可	
10	^	ビット排他論理和	実数の場合、使用不可	
11		ビット論理和	実数の場合、使用不可	
12	&&	論理積		
13		論理和		
14	? :	3項演算	ex,    a == 1 ? 0 : 1	
15	= +=, -= *=, /=, %= &=,  =, ^= <<=, >>=	代入 加減算 代入 乗除算, 剰余算 代入 ビット論理積, 論理和, 排他論理和 代入 左シフト, 右シフト 代入	数値型変数への(演算付)代入  実数の場合、使用不可 実数の場合、2 <sup>n</sup> で乗算／除算	

※ カンマ(,) 演算子は使用できません。(但し、「for(初期化; 条件; 更新)」の初期化と更新はカンマで区切って複数指定可)

※ 「# (項番)」が同じものは、同一優先度です

※ 「#2」の文字列の代入(=)の場合、空文字列(“)を代入した場合は「0」、それ以外は「1」となります。  
文字列の比較(==, !=, <, >, <=, >=)は、結果が真(成立)の場合「1」、偽(不成立)の場合は「0」となります。

```
(例)  var %= 100;           $str = "ABC" ;
      var <= (a + (b[2] = 3));  $str >= $func(1, 2);
      var = (a++ + ++b);
      var = a << 2;
      (var = a + func(1, 2)) || b == 2;
```

### ●プリ・インクリメント／デクリメントや、ポスト・インクリメント／デクリメントについて

プリインクリメント／デクリメントやポストインクリメント／デクリメントは、「++」や「--」を変数の直前または、直後に記述し、変数を括弧で囲んだりしないでください。

```
ex.    ++a; -----OK
      b--; -----OK
      (c)++; --- エラー
```

### 5.2.7. 数値定数

数値定数の表現は、C言語と同じです。(123, -789, 3.14, 0.7G3 . . . 等)

尚、文字列も数値として扱うことができます。(先頭の8バイトまで)

例えば、「a = "ABC"」は、「a = 0x414243」と同じになります。

但し、文字式の場合は、文字式の評価結果となります。(「a = "ABC" == "DEF"」は、「a = 0」となります)

### 5.2.8. 文字列

文字列は、連なる文字をダブルクォート(“)、あるいは、アポストロフィ(‘)で囲みます。

ダブルクォート(“)で囲んだ文字列は、C言語と同様の「¥」で始まるエスケープ文字が記述できます。

アポストロフィ(‘)で囲んだ文字列は、エスケープ文字を認識せずに、書いた内容がそのままの文字列となります。

```
ex.    "¥x1B[0mABC¥n" ----- 「¥x1B」は1バイトの「0x1B」に、「¥n」は「0x0A」になります。
      'c:¥work¥subdir¥sample.txt' --- 書いた内容「c:¥work¥subdir¥sample.txt」がそのまま文字列となる
```

### 5.2.9. 文字列の代入／比較

数式中に文字列の代入や比較を記述することができます。  
文字列の代入は、以下の構文で記述します。

```
$文字変数 = <" 文字列" / $文字変数 / $文字関数(・・・)>
```

文字列の代入結果は、空文字列(“”)を代入した場合 偽(=0)、 空文字列以外を代入した場合 真(=1)です。

```
(例)    $s[1] = "ABC"
        $t = $func(1, 2)
        $u = $s[1]
```

文字列の比較は、以下の構文で記述します。

```
<" 文字列" / $文字変数 / $文字関数(・・・)> <比較演算子> <" 文字列" / $文字変数 / $文字関数(・・・)>
比較演算子: 「==」, 「!=」, 「<」, 「>」, 「<=」 or 「>=」
```

文字列は、英字の大文字と小文字を区別します。(“ABC” と “abc” は異なる文字列と判断します)  
文字列の比較結果は、比較が成立する場合 真(=1)、成立しない場合 偽(=0)です。

```
(例)    $s[1] == "ABC"
        $t >= $func(1, 2)
        $u < $s[1]
```

文字列の代入／比較は、数式中で高優先度で計算されます。(文字列代入／比較全体を括弧で囲んだのと同様です)  
例えば、「if (1 - \$s == “ABC”)」は、「if (1 - (\$s == “ABC”))」と同じです。

### 5.2.10. 配列の代入

代入演算子「=」により、配列全体の代入が可能です。  
配列の代入は、以下の構文で記述します。

```
変数 1 [] = 変数 2 []; ----- 変数 1 に変数 2 の全内容をコピーします。変数 1 の要素数は変数 2 と同じになります。
変数 1 [] = 関数(・・・); --- 関数は return 文で配列を返す(ex. return var[]; )
```

**配列の代入は、通常の数式式中では記述できません。**上記構文により単独で記述してください。

```
(例)    a[] = b[]; ----- 数値型配列の代入 (配列全体をコピー)
        a[] = func(1, 2); ----- func は、return 文で配列を返す(ex. return n[]; ) .....※1
        $s[] = $t[]; ----- 文字列配列の代入 (配列全体をコピー)
        $s[] = $func(1, 2); --- $func は、return 文で配列を返す(ex. return $x[]; ) .....※1
```

※1: 配列全体を返す関数の例を以下に示します。(下記例は、数値型配列の例)

関数の戻り値に配列 (添え字なしで「**変数[]**」のように記述) を指定した場合、変数には配列全体がコピーされます。

```
(例)    function func()
        {
            define arr[5] = {10, 20, 30, 40, 50};
            return arr[];
        }
        .....
        var[] = func(); // 変数「var」には配列「arr[5]」の内容がコピーされ、
                        // var[0] = 10, var[2] = 20, ... var[4] = 50 となります。
        .....

```

但し、関数の戻り値を受ける変数を通常の変数で記述した場合は、当該変数に配列の先頭の値 (上記 [arr[0]] ) が設定されます。  
上記例で、「var[] = func();」を「var[2] = func();」とした場合、var[2]には、arr[0]の内容(=10)が設定されます。  
逆に、「var[] = func();」で、func() が配列を返さない (単一の値を返す) 場合は戻り値が var[0]に設定されます。  
関数の戻り値を受ける変数を通常の変数で記述した「var[2] = func()」のような記述は、通常の数式内でも記述できます。

### 5.2.11. 処理ブロック

「do { ... }」（後置き while() なし）で処理ブロックを記述できます。  
 この処理ブロックは、繰り返し処理は行わず、単に「break」により抜けるブロックとなります。  
 以下の 2 つの構文は、同じ動作となります。

do ブロック（後置き while なし）

```
do {
    if (a == 10) break; // →①へジャンプ
    . . . . .
}
①→
```

do ブロック（後置き while あり）

```
do {
    if (a == 10) break; // →①へジャンプ
    . . . . .
} while(0);
①→
```

※ do ブロックの場合、ブラケット（“{” と “}”）は必須です。

その他の処理ブロック（ほとんど C 言語と同じですが、**switch-case 文は使用できません**）

while ブロック（前置き while, ブラケットあり）

```
while (条件式) {
    [expression;]
    . . . . .
}
```

while ブロック（前置き while, ブラケットなし）

```
while (条件式) [expression];
```

for ブロック（ブラケットあり）

```
for ([初期化式 1][, 初期化式 2] ...; [条件式]; [更新式 1][, 更新式 2] ...) {
    [expression;]
    . . . . .
}
```

※ for ブロックの場合のみ、カンマ演算子(,)で区切って複数の「初期化式」「更新式」を記述できます

for ブロック（ブラケットなし）

```
for ([初期化式 1][, 初期化式 2] ...; [条件式]; [更新式 1][, 更新式 2] ...) [expression];
```

if / else ブロック（ブラケットあり）

```
if (条件式) {
    [expression;]
    . . . . .
}

[ else if (条件式) {
    [expression;]
    . . . . .
}

    ○
    ○
    ○

[ else {
    [expression;]
    . . . . .
} ]
```

if / else ブロック（ブラケットなし）

```
if (条件式) [expression] ;
[ else if (条件式) [expression] ; ]

    ○
    ○
    ○

[ else [expression] ; ]
```

※ C 言語と同様に、「ブラケットあり」と「ブラケットなし」が混在しても OK です。

もちろん、各処理ブロックはネストすることができます。

### 5.2.12. break / goto 文

C 言語と同様に break, goto 文を使用できます。

ex.

```
while (!fEnd) {
    for (i = 0; i < 10; i++) {
        . . . . .
        if (fIllegalCondition) goto label;
        . . . . .
    }
    if (fExitCondition) break;
}
label;;
```

### 5.2.13. 関数の実行

C 言語と異なり、関数が後方に記述されていてもプロトタイプを定義する必要はありません。(プロトタイプは記述できません)

マクロテキスト内で定義した関数を呼び出す場合、関数の引数の個数と、引数の型(数値型/文字型)は完全に一致していなければなりません。

```
(例)      . . .
          add(1, 2); ----- OK   (引数の個数と型が完全に一致, func() は後方に定義されていても参照可)
          add(1, 2, 3); ----- エラー (引数の個数が一致しない)
          add(1, "XYZ"); ----- エラー (第2引数の型が一致しない)
          . . .
          function add(a, b)
          {
              return a + b;
          }
```

引数に 'var[]' のようにインデックスを指定しない配列の場合は、配列全体を渡します。

この場合、引数を受け取る関数でも、引数を 'arg[]' のようにインデックスを指定しない配列としなければなりません。

```
(例)      define var[3] = {10, 20, 30};
          sub(var[]);
          . . .
          function sub(arg[]) {
              // arg[0]=10, arg[1]=20, arg[2]=30 が設定されています。
          }
```

関数に渡す引数は、(配列全体を渡す場合も含めて) 仮引数としてコピーされたデータとなります。

従って、関数内で引数の値を変更しても、呼び出し元の変数値は変更されません。

```
(例)      define var = 30;
          sub(var);
          . . .
          function sub(arg) {
              arg = 999; // 呼び出し元の引数 (var) の値は変更されません (30 のままです)
          }
```

<注> **関数の引数に 文字式を含む数式**を指定する場合は、数式全体を括弧で囲むか、数式の先頭を数値項目としてください。

- func(**\$a** == "ABC" + 1);    --- ×
- func(**(**\$a == "ABC" + 1)**)**;    --- ○ (\$a が "ABC" の場合「func(2)」、\$a が "ABC" 以外の場合「func(1)」となる)
- func(**1 +** \$a == "ABC" **)**;    --- ○ (\$a が "ABC" の場合「func(2)」、\$a が "ABC" 以外の場合「func(1)」となる)



### 5.3.2. イベント

マクロテキストの実行中に以下のイベントを処理することができます。

いずれのイベントも多重には発生しません。

イベントの実行中にさらにイベントが発生した場合や、イベントが禁止状態である場合にイベントが発生した場合の動作については、下表の「多重発生時」の欄を参照。

標準イベント処理関数

#	イベント 処理関数名	イベント	イベント 禁止／許可	多重 発生時	備考
1	<code>_OnKeyIn</code>	キー入力	<code>_DisStdEvt</code> <code>_EnaStdEvt</code>	全保留	
2	<code>_OnRClick</code>	右クリック	〃	1 回の み保留	Shift や Ctrl キーを押下しない場合は <code>_EnableStdRClick(FALSE)</code> 実行要
3	<code>_OnDb1C1k</code>	ダブルクリック	〃	1 回の み保留	<code>AjcCipSetDb1C1kAct()</code> API で <code>fNtcDb1C1k=FALSE</code> 指定要
4	<code>_OnFileSearch</code>	ファイル検索フォルダ通知	〃	破棄	
5	<code>_OnTmcClose</code>	タイムチャートグラフ クローズ	〃	全保留	
6	<code>_OnButton</code>	ボタン押下	〃	全保留	
7	<code>_OnChkBox</code>	チェックボックス設定	〃	全保留	
8	<code>_OnTimer</code>	タイマ周期通知	〃	1 回の み保留	但し、タイマ停止時は、残留イベントを破棄
9	<code>_OnMenu</code>	ポップアップメニュー選択	〃	全保留	実際の関数名は「 <code>_PopupMenu()</code> 」で指定

COMポート通信イベント処理関数

#	イベント 処理関数名	イベント	イベント 禁止／許可	多重 発生時	備考
1	<code>_OnRxText</code>	テキスト受信通知	<code>_DisComEvt</code> <code>_EnaComEvt</code>	全保留	
2	<code>_OnRxCtrl</code>	制御コード受信通知	〃	全保留	
3	<code>_OnRxEsc</code>	E S C シーケンス受信通知	〃	全保留	
4	<code>_OnRxPacket</code>	パケット受信通知	〃	全保留	

※ `main()` 等、プログラム開始からの延長線で実行されるコードをメインコードといいます。

これに対し、上記イベントで実行されるコードをイベントコードといい、イベントコードは非同期に実行されます。

イベントは、メインコードの各ステップに間に割り込んで実行されます。

ステップとは、`if` 等の条件式や実行コードの式が単位となりますが、途中に関数の呼び出しがある場合は関数呼び出しの直前までとなります。また、`while/for` 等の制御構造の末尾 ( `‘;’` や `’}` ) も 1 つの実行ステップとなります。

### 5.3.3. C 言語との相違点

C 言語との、およその相違点を以下にまとめます。

- 1) 「#」で始まるプリプロセス文 (#include, #if や#define 等) は使用できません。
- 2) switch-case 文は使用できません。
- 3) ポインタは使用できません。
- 4) 変数は define 文で定義します。
- 5) 変数のタイプは、数値型と文字型で、数値型は「符号付き整数」「符号無し整数」or「実数」に統一されます。  
数値演算式の評価は、統一された型（「符号付き整数」「符号無し整数」or「実数」）で行われます。
- 6) 関数は function 文で定義し、引数は変数名だけを列記します。
- 7) 関数のプロトタイプを定義する必要はありません。
- 8) 構造体(struct)や共用体(union)は使用できません。
- 9) プログラム (マクロテキスト) は、1つのテキストファイルにしなければなりません。
- 10) 配列は、1次元配列のみ使用可能です。
- 11) 配列の要素数には、変数を含め一般の数式が使用できます。
- 12) 関数内で定義された自動変数は、(ブラケット「{・・・}」でネストしても) すべて関数内全体のスコープとなります。
- 13) 後置き while 「do {・・・} while(式);」の while は省略できます。  
この場合「do {・・・}」は、単に break で抜けるための処理ブロックとなります。
- 14) マクロテキスト内で定義する関数では、可変個引数「...」は使用できません。
- 15) 文字式 (文字列の代入や比較) が使用できます。


## 5.3.4. 内部関数

本マクロ実行コントロールには、以下の関数が暗黙的に内蔵されています。

関数定義の記述形式は、以下の通り。

- 1) [XXXX]で囲まれた引数は省略可能。但し、後続のパラメタを指定する場合は省略できない。  
ex. \_Func( a [,b] [,c]) ----- 「a」は必ず指定しなければならない。  
「b」「c」は省略可能だが、「c」を指定する場合は「b」も指定しなければならない。
- 2) 「・・・」は、直前のパラメタを繰り返して指定可能  
ex. \_Func( a [,b]・・・) ----- 「b」は(何回でも)繰り返して指定可能
- 3) {XXXX | YYYY | ZZZZ}は、いずれかの項目(XXXX, YYYY or ZZZZ)を指定することを意味します。

## オプション設定

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>option {signed   unsigned   double}</b> 数値型変数のタイプを指定します。 この命令は、マクロテキストの先頭に記述しなければなりません。 signed - マクロテキスト中の数値型変数を「符号付き整数」として扱います unsigned - マクロテキスト中の数値型変数を「符号なし整数」として扱います double - マクロテキスト中の数値型変数を「実数」として扱います	— ※AjcCipSetOption()の引数で fEnableOption=FALSEとした場合はoption文を記述できません。	M010
2	<b>_EnableStdRClk(fEnable)</b> 右クリックによる標準のポップアップメニューを許可／禁止します。 許可した場合は、右クリックにより標準のポップアップメニューが表示されます。 禁止した場合は、右クリックイベントが有効となり、右クリックにより_OnRClick()に が実行されます。 fEnable - _TRUE : 標準のポップアップメニュー許可 (デフォルト) _FALSE : 標準のポップアップメニュー禁止 (_OnRClick()実行可) ※Shift/Ctrl キーを押さないで右クリックしたときの挙動を設定します。 Shift/Ctrl キーを押しながら右クリックした場合は、本設定とは関係なく常に右ク リックイベントが発生します。	— ※標準のポップアップメニュー 	M010 M070
3	<b>_EnableAutoScroll(fEnable)</b> オートスクロール (描画時に画面最下行へスクロールする) の許可／禁止 fEnable - _TRUE : オートスクロール許可 (デフォルト) _FALSE : オートスクロール禁止	—	M020

## スクリーン情報

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>_SetVramSize(width, height)</b> VRAMのサイズを設定します。表示内容はクリアされます。 width : VRAM の幅 (半角文字数, デフォルト=256 文字) height : VRAM の高さ (行数, デフォルト=64 行) ※本コントロールを使用するアプリケーションで設定する VRAM サイズに依存します。 (デフォルトでは最大、2 5 6 桁, 6 4 行)	— ※_locate()により VRAM 内の 任意の位置へカーソルを移動 できます。	M020 M030
2	<b>_SetBufLines(lines)</b> スクロールバッファの行数を設定します。表示内容はクリアされます。 lines : スクロールバッファの行数 (VRAM の高さを含む) ※本コントロールを使用するアプリケーションで設定する値に依存します。 (デフォルトでは1 0 0 0 0行)	— ※何行までバッファリングし、 スクロール可能とするかを設 定します。	M030

つづく



3	<b>_GetVRamWidth()</b> VRAM の幅 (半角文字数) を取得します	VRAM の幅 (半角文字数)	M030
4	<b>_GetVRamHeight()</b> VRAM の高さ (行数) を取得します	VRAM の高さ (行数)	M030
5	<b>_GetBufLines()</b> スクロールバッファの行数 (VRAM 行数を含む) を取得します	スクロールバッファの行数	M030

## 表示出力

#	関数定義 内容	戻り値/※備考	サン プル
1	<b>_Printf(\$fmt [,arg]・・・)</b> 書式化した文字列を表示します。 \$fmt - 書式文字列 (C 言語の printf() とほぼ同じ) arg - 書式(%・・)に対応した引数	— ※書式文字列(\$fmt)については、 冒頭の説明中の「書式文字列」 を参照	M010 M020 M050
2	<b>_Print({item1   \$item1} [, {item2   \$item2}・・・)</b> カーソル位置へ多項目表示出力 item / \$item - 数値/文字列 (引数で指定された項目を、数値は 10 進で、文字列はそのまま出力)	—	M020
3	<b>_Pd({item   \$item} [,len] [,lup] )</b> カーソル位置へ 1 項目表示出力 (引数で指定された項目を、数値は 10 進で、文字列はそのまま出力) item / \$item - 数値/文字列 len - 全表示桁数 (負数の場合は左詰め、省略時は有効文字長と同じ) lup - 小 数点以下の表示桁数 (実数モード時のみ有効)	— ※実数モード時、「lup」を省略 した場合、小数部は常に 6 桁で 表示	M061 M070 M101
4	<b>_Ph({item   \$item} [,len] )</b> カーソル位置へ 1 項目表示出力 item / \$item - 数値/文字列 len - 全表示桁数 (負数の場合は左詰め) (引数で指定された項目を、数値は 16 進で、文字列はそのまま出力)	—	M051 M052 M070
5	<b>_Locate(x, y)</b> 描画位置設定 x: 桁位置 (0～) y: 行位置 (0～) ※引数(x, y)は、本コントロールを使用するアプリケーションで設定するコントロール の VRAM サイズに依存します。(デフォルトでは最大、x=255, y=63) また、_SetVramSize()でも VRAM のサイズを設定可能です。	— ※位置を設定して描画する場合 通常_EnableAutoScroll(FALSE) によりオートスクロールを 禁止します。	M020
6	<b>_Cls( )</b> 画面クリアと描画色リセット (描画色=黒, 背景色=白)		M020
7	<b>_SetTextColor(color)</b> テキストの描画色を設定します。 color: テキスト描画色 (0:黒 1:赤 2:緑 3:黄 4:青 5:紫 6:水色 7:白)		M020
8	<b>_SetBackColor(color)</b> テキストの背景色を設定します。 color: テキスト背景色 (0:黒 1:赤 2:緑 3:黄 4:青 5:紫 6:水色 7:白)		M020
9	<b>_ResetColor()</b> テキストの描画色=黒, 背景色=白を設定します。		M020

配列操作			
#	関数定義 内容	戻り値／※備考	サンプ プル
1	<b>_Elements(arr[])</b> 配列変数の要素数を得る	配列の要素数	M040 M063
2	<b>_MakeArray(n)</b> n で指定された要素数の数値型配列を返します。(ex. arr[] = _MakeArray(10); )  ※ ex. n[] = _MakeArray(10) --- n[] を要素数 10 の配列に変更する	※戻り値を受ける変数を要素数 n の数値型配列に変更する	M040
3	<b>\$_MakeArray(n)</b> n で指定された要素数の文字型配列を返します。(ex. \$arr[] = \$_MakeArray(10); )  ※ ex. \$s[] = \$_MakeArray(10); -- \$s[] を要素数= 10 の配列に変更する	※戻り値を受ける変数を要素数 n の文字型配列に変更する	M040

整数演算			
#	関数定義 内容	戻り値／※備考	サンプ プル
1	<b>_CalcCrc(knd, bno, loc, len [, ini])</b> 16ビットCRCの算出 knd - 演算種別 _CRC_ITL : CCITT X.25 左送り _CRC_ITR : CCITT X.25 右送り _CRC_16L : ANSI-CRC 左送り _CRC_16R : ANSI-CRC 右送り bno - メモリブロック番号 loc - メモリブロックの先頭ロケーション len - バイト数 ini - 初期値 (0x0000 or 0xFFFF)	CRC 値  ※CCITT X.25 の生成多項式 $X^{16} + X^{12} + X^5 + 1$  ※ANSI-CRC の生成多項式 $X^{16} + X^{15} + X^2 + 1$	M021
2	<b>_SRand(n)</b> 擬似乱数の乱数系列初期化 n = 1: 乱数ジェネレータの再初期化 n ≠ 1: 乱数系列の初期化	—	M020
3	<b>_Rand()</b> 擬似乱数値取得	擬似乱数値 (0~32767)	M020

算術演算			
#	関数定義 内容	戻り値／※備考	サンプ プル
1	<b>_Sqrt(n)</b> n の平方根取得	n の平方根	M100
2	<b>_Sin(degree)</b> 正弦値取得 (degree は度単位の角度)	正弦値	M100 M120
3	<b>_Cos(degree)</b> 余弦値取得 (degree は度単位の角度)	余弦値	M100 M120
4	<b>_Tan(degree)</b> 正接値取得 (degree は度単位の角度)	正接値	M100 M120

つづく

5	<b>_ASin(n)</b> n の逆正弦値取得	度単位の角度 (-90~+90)	M100
6	<b>_ACos(n)</b> n の逆余弦値取得	度単位の角度 (0~180)	M100
7	<b>_ATan(n)</b> n の逆正接値取得	度単位の角度 (-90~+90)	M100
8	<b>_Atan2(y, x)</b> y/x の逆正接値取得	度単位の角度 (-180~+180)	M100
9	<b>_Abs(n)</b> n の絶対値取得	絶対値	M100 M120
10	<b>_Min(a, b)</b> a, b の小さい方の値を返します	小さいほうの値	M120
11	<b>_Max(a, b)</b> a, b の大きい方の値を返します	大きいほうの値	M120

## 時間／ウェイト

#	関数定義 内容	戻り値／※備考	サン プル
1	<b>_CurTime()</b> 現在時刻 (ローカルタイム) を 1970/1/1 00:00:00 からの通算秒数で取得	通算秒数	M020 M090
2	<b>_GetTime(year, month, day, hour, minute, second)</b> 指定日時を 1970/1/1 00:00:00 からの通算秒数で取得	1970/1/1 00:00:00 からの通算秒数	M090
3	<b>_GetDateAndTime([seconds])</b> 指定日時(1970/1/1 00:00:00 からの通算秒数) がら日付と時刻を得る seconds を省略した場合は、現在の日時を得る (ex. Today[] = _DateAndTime(); )	日時を格納した配列 [0]: 西暦年     [3]: 時 [1]: 月         [4]: 分 [2]: 日         [5]: 秒 [6]: ミリ秒	M020
4	<b>\$_Date([seconds])</b> 指定日時(1970/1/1 00:00:00 からの通算秒数) がら日付の文字列を得る seconds を省略した場合は、現在の日付を得る	日付の文字列 ( "yyyy/mm/dd" )	M020 M090
5	<b>\$_Time([seconds])</b> 指定日時(1970/1/1 00:00:00 からの通算秒数) がら時刻の文字列を得る seconds を省略した場合は、現在の時刻を得る	時刻の文字列 ( "hh:mm:ss" )	M090
6	<b>\$_DateAndTime([seconds])</b> 指定日時(1970/1/1 00:00:00 からの通算秒数) がら日付と時刻の文字列を得る seconds を省略した場合は、現在の日時を得る	日付と時刻の文字列 ( "yyyy/mm/dd hh:mm:ss" )	M020 M090
7	<b>_msWait(msTime)</b> ミリ秒単位でウェイト (Sleep(1) のループで時間経過をチェック。解像度は Windows タイマ解像度 (10~16ms 程度) に依存)	—	M020 M120
8	<b>_usWait(usTime)</b> マイクロ秒単位でウェイト (ウェイト中は Sleep() せずにループし、CPU を使い続けます)	—	M090

## タイマ起動／停止

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>_TimerStart(tid, period)</b> 一定周期でタイマイベント「_OnTimer()」を発生します。 tid - タイマ識別 ID (1～31) period - タイマイベント起動周期[ms]	—	M062
2	<b>_TimerStop([tid])</b> タイマイベントの発生を停止します。 tid - タイマ識別 ID (1～31) ※「tid」を省略した場合は、全てのタイマを停止します。	—	M062

## 文字列

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>_StrLen(\$str)</b> 文字列の長さを取得します。	文字列の長さ	M061
2	<b>_StrICmp(\$str1, \$str2 [, nEqual])</b> ; 文字列の比較 (英大文字, 小文字を区別しないで比較 ※1) nEqual: 文字列が一致した時の戻り値 (nEqual 未指定時は 0 を仮定)	= nEqual: 等しい > nEqual: \$str1 > \$str2 < nEqual: \$str1 < \$str2	M130
3	<b>_StrStr(\$str1, \$str2)</b> 文字列 1 (\$str1) 中の部分文字列 (\$str2) 検索 (大文字／小文字を区別して比較)	≠-1: 部分文字列の位置 (0～) =-1: 文字列が見つからない	M061
4	<b>_StrIStr(\$str1, \$str2)</b> 文字列 1 (\$str1) 中の部分文字列 (\$str2) 検索 (大文字／小文字を区別しないで比較)	≠-1: 部分文字列の位置 (0～) =-1: 文字列が見つからない	M130
5	<b>_ToValue(\$str)</b> 数値表現の文字列を数値に変換する (先頭が "0x" の場合は 16 進整数とします) \$str: 数値表現の文字列 (ex. "123", "-3.14", "0xFF")	変換した数値	M130
6	<b>\$_StrCat([\$str1] [, \$str2] ...)</b> 指定した文字列を全て連結した文字列を取得	連結した文字列	M060 M070
7	<b>\$_PathCat(\$str1, \$str2)</b> 2つの文字列の間に、必ず 1つの「¥」が挿入された文字列を作成する	連結した文字列	M130
8	<b>\$_StrPart(\$str, loc, len)</b> 文字列中の部分文字列を取得 loc: \$str 中の部分文字列のバイト位置 (負数も可) len: \$str 中の部分文字列の長さ (負数の場合は左方向への長さ ex. \$_StrMid(\$s, 3, -3); 先頭 3 文字)	部分文字列 ※loc, len が文字列の範囲外の場合は、マッピングされる文字列の右端／左端を仮定する。但し、文字列外をマッピングしている場合は空文字列	M070
9	<b>\$_StrRep(\$str1, loc, len, \$Str2)</b> 文字列 (\$str1) 中の部分文字列を、\$Str2 で置換 loc: \$str1 中の部分文字列のバイト位置 (0～\$str1 の長さ) len: \$str1 中の len: 部分文字列の長さ (0 の場合は文字列 (\$Str2) を挿入)	部分文字列を置換した文字列 ※loc が負数の場合は 0 に、\$str を超える場合は \$str1 の長さに補正します	M072

つづく

※1：英大文字、小文字を区別する場合は、文字列の比較式（ex. \$txt == “ABC”）で記述可能

問い合わせ			
#	関数定義 内容	戻り値／※備考	サ ン プ ル
1	<b>_MessageBox(\$Message, \$Title, BoxType)</b> 問い合わせのメッセージボックスを表示します。 \$Message - 問い合わせメッセージ文字列 \$Title - メッセージボックスのタイトルバーに表示する文字列 BoxType - メッセージボックスのタイプ _MB_OK - [OK] プッシュボタンだけを表示 _MB_OKCANCEL - [OK]、[キャンセル] の各プッシュボタンを表示 _MB_RETRYCANCEL - [再試行]、[キャンセル] の各プッシュボタンを表示 _MB_YESNO - [はい]、[いいえ] の各プッシュボタンを表示 _MB_YESNOCANCEL - [はい]、[いいえ]、[キャンセル] の各プッシュボタンを表示 _MB_ICONERROR - 停止マークアイコンを表示します。	_IDOK - 「OK」を選択 _IDCANCEL - 「CANCEL」を選択 _IDNO - 「NO」を選択 _IDYEAS - 「YES」を選択	M070
2	<b>_PopupMenu(\$menu[], x, y [, \$Func])</b> ポップアップメニューを表示し、選択した項目を返します。 \$menu[] - メニュー項目 (文字型配列) x, y - メニューの表示位置 (マクロウインドの左上が原点(0, 0)) \$Func - メニューが選択された時に実行するイベントファンクション	・ \$Func 未指定時 ≥0: 選択されたメニュー項目 (項目のインデクス値) -1: メニューはキャンセルされた ・ \$Func 指定時は返り値なし	M010 M070 M071 M072

メモリブロック			
#	関数定義 内容	戻り値／※備考	サ ン プ ル
1	<b>_MemAlloc(BlockSize)</b> メモリブロックを割り当てます BlockSize - 割り当てるメモリブロックのバイト数(1～0xFFFFFFFF, 最大 4GB-1)	メモリブロック番号	M050
2	<b>_MemFree(MemBlkNo)</b> メモリブロックを解放します。 MemBlkNo - メモリブロック番号	—	M050 M060 M070
3	<b>_GetMemBlk(MemBlkNo, loc, len [, fChangeEndian])</b> メモリブロック内のデータを読み出します。 (指定位置から指定バイト数の整数値(8～64Bit, リトルエンディアン)を読み出します) MemBlkNo - メモリブロック番号 loc - バイト位置 (0～) len - バイト数 fChangeEndian - エンディアン形式(BigEndian/LittleEndian)の変更指定フラグ _FALSE : エンディアン形式を変更しない _TRUE : エンディアン形式を変更する	メモリブロック内から読み出したデータ値 (実数モードの場合は、読み出した整数値(8～64Bit)を実数値に変換します。)	M050 M051 M052 M070
4	<b>_PutMemBlk(MemBlkNo, loc, len, value [, fChangeEndian])</b> メモリブロック内へデータを書き込みます (データの整数値(下位 8～64Bit)を指定バイト位置, バイト数へ書き込みます) MemBlkNo - メモリブロック番号 loc - バイト位置 (0～) len - バイト数 (1～8) value - メモリブロック内へ書き込むデータ値 実数モードの場合は、実数値を整数値(8～64 Bit)に変換して書き込みます fChangeEndian - エンディアン形式(BigEndian/LittleEndian)の変更指定フラグ _FALSE : エンディアン形式を変更しない _TRUE : エンディアン形式を変更する	—	M050 M051 M052

つづく

5	<b>_ClrMemBlk(MemBlkNo, ByteValue)</b> メモリブロックをバイト値(Value)でクリアーします。 MemBlkNo - メモリブロック番号 Value - バイト値(0~255)	—	M050 M051 M052
6	<b>_DumpMemBlk(MemBlkNo [, loc] [, len])</b> メモリブロックの内容を、16進ダンプ表示します。 MemBlkNo - メモリブロック番号 loc - バイト位置 (省略時は0を仮定) len - バイト数 (省略時はメモリブロック末尾までのバイト数を仮定)	—	M050 M051 M052

## ファイル/フォルダ

#	関数定義 内容	戻り値/※備考	サンプル
1	<b>_FOpen(\$FilePath, \$Access)</b> ファイルをオープンします。 \$FilePath - ファイルパス名 \$Access - ファイルアクセス指定(“rt”, “wt”, “at”, “rb”, “wb” or “ab”) “rt”: テキストファイル読出 “rb”: バイナリファイル読出 “wt”: テキストファイル書込 “wb”: バイナリファイル書込 “at”: テキストファイル追記 “ab”: バイナリファイル追記	ファイル番号 (1 ~ 31)	M060 M061 M062
2	<b>_FClose(fno)</b> ファイルをクローズします。 fno - ファイル番号(_FOpen の戻り値)	— ※マクロ終了時、クローズされていないファイルは全て自動的にクローズします。	M060 M061 M062
3	<b>_FPutS(fno, \$str)</b> ファイルに文字列を書き込みます。 fno - ファイル番号(_FOpen の戻り値) \$str - ファイルに書き込む文字列 ※ _FOpen のファイルアクセス指定は “wt” or “at” でなければなりません。 ※ ファイルから文字列の読み出しは、文字型関数(\$FGetS) 参照	—	M060
4	<b>_FRead(fno, bno, loc, len)</b> ファイルからバイナリデータを読み出します。 fno - ファイル番号(_FOpen の戻り値) bno - メモリブロック番号(_MemAlloc の戻り値) loc - 読みだしたデータを格納するバイト位置 (メモリブロック内ロケーション) len - 読み出すバイト数 ※ _FOpen のファイルアクセス指定は “rb” でなければなりません。	読み出したバイト数	M060
5	<b>_FWrite(fno, bno, loc, len)</b> ファイルへバイナリデータを書き込みます。 fno - ファイル番号(_FOpen の戻り値) bno - メモリブロック番号(_MemAlloc の戻り値) loc - 書き込むデータのバイト位置 (メモリブロック内ロケーション) len - 書き込むバイト数 ※ _FOpen のファイルアクセス指定は “wb” or “ab” でなければなりません。	書き込んだバイト数	M060
6	<b>_FGetSeek(fno)</b> 現在のファイルポインタ (ファイルのアクセスバイト位置) を取得します。 fno - ファイル番号(_FOpen の戻り値)	現在のファイルポインタ	M060
7	<b>_FSetSeek(fno, loc)</b> ファイルポインタ (ファイルのアクセスバイト位置) を設定します。 fno - ファイル番号(_FOpen の戻り値) loc - ファイルポインタ	—	M060

8	<b>\$_FGetS(fno)</b> ファイルから文字列を読み出します。(1行分, 最大 1023 バイト) fno - ファイル番号(_FOpen の戻り値) ※ _FOpen のファイルアクセス指定は”rt” でなければなりません。	読み出した文字列	M060 M061 M062
9	<b>_FDelete(\$FilePath)</b> 指定されたファイルを削除します。 \$FilePath - 削除するファイルのパス名(ex. ‘d:¥work¥sub¥temp¥Sample.txt’);	—	M060
10	<b>_MakeFolder(\$FolderPath)</b> 絶対パス名で指定されたフォルダを作成します。(パス名の末尾は「¥」であること) \$FolderPath - 作成するフォルダの絶対パス名(ex. ‘d:¥work¥sub¥temp¥’);	—	M140
11	<b>_PathExists(\$Path)</b> パスが存在するかチェックします \$Path - 存在をチェックするパス名	_TRUE : 存在する _FALSE : 存在しない	M140
12	<b>_PathIsFolder(\$Path)</b> 指定したパスが、実在するフォルダかチェックします \$Path - 実在するフォルダがをチェックするパス名	_TRUE : 実在するフォルダ _FALSE : 当該フォルダは存在しない	M140

## ファイル名／フォルダ名取得

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>\$_GetOpenFile([\$Title] [, \$Filter[ [, \$DefExt] [, \$DefFile]])</b> ダイアログにより、オープンするファイルを選択します。 \$Title - ダイアログタイトル (未指定時は空文字列) \$Filter - ファイル選択フィルタ(未指定時は “All(*.*)/*.*”) --- ※1 \$DefExt - デフォルト拡張子 (未指定時は “txt”) \$DefFile - デフォルトのファイル・パス名(未指定時は空文字列)	選択したオープンファイルのパス名 (キャンセルした場合は空文字列(“”))	M070
2	<b>\$_GetSaveFile([\$Title] [, \$Filter] [, \$DefExt] [, \$DefFile])</b> ダイアログにより、セーブするするファイルを選択／指定します。 \$Title - ダイアログタイトル (未指定時は空文字列) \$Filter - ファイル選択フィルタ(未指定時は “All(*.*)/*.*”) --- ※1 \$DefExt - デフォルト拡張子 (未指定時は “txt”) \$DefFile - デフォルトのファイル・パス名(未指定時は空文字列)	選択/指定したセーブファイルのパス名 (キャンセルした場合は空文字列(“”))	M061 M062 M071 M072
3	<b>\$_GetFolder([\$WndTitle] [, \$BoxTitle] [, DefFolder])</b> ダイアログにより、フォルダを選択します。 \$WndTitle - ウインド・タイトルバーに表示するタイトル(未指定時は空文字列) \$BoxTitle - ダイアログボックス内に表示するタイトル(未指定時は空文字列) \$DefFolder - デフォルトのフォルダ・パス名(未指定時は空文字列)	選択したフォルダのパス名 (キャンセルした場合は空文字列(“”))	M063 M072
4	<b>\$_GetMacFilePath()</b> 現在実行中のマクロファイル・パス名を取得します。	マクロファイルのパス名	M064
5	<b>\$_SearchFiles(\$Folder, fSubDir [{, \$Wild   \$Wild[] }])...</b> ファイルを探索し、見つかったファイルパス名の配列を返します。 \$Folder - 探索するフォルダパス名 fSubDir - サブフォルダ探索フラグ (_TRUE:サブフォルダも探索する、_FALSE:探索しない) \$Wild - 探索するワイルドカード \$Wild[] - 探索するワイルドカード (配列による複数指定)  ex. \$files[] = _SearchFiles( ‘d:¥work’ , _TRUE, “*.c” , “*.h” );	見つかったファイルパス名の配列  ※マクロテキスト中に「_OnFileSearch()」関数があれば、これを呼び出して、探索中のフォルダ名を通知します。 (→イベント処理関数を参照)	M063



6	<b>\$_SplitPath(\$PathName)</b> パス名分解 (絶対パス名を分解し、要素数 = 4 の文字型配列を返します)  ex.    \$s[] = \$_SplitPath( 'c:\\$sub_dir\$sample.txt' );	以下の内容の文字型配列 [0]: ドライブ (ex. 'c:') [1]: フォルダバ (ex. \$sub_dir\$) [2]: ファイル名 (ex. sample) [3]: 拡張子 (ex. '.txt')	M064
7	<b>\$_MakePath(\$drive [, \$dir] [, \$fname] [, \$ext])</b> <b>\$_MakePath(\$parts[])</b> パス名作成 (ドライブ, フォルダ名, ファイル名, 拡張子を連結したパス名の作成) \$drive: ドライブ (ex. 'c:') \$dir : フォルダ名 (ex. '\$sub_dir\$') \$fname: ファイル名 (ex. 'sample') \$ext : 拡張子 (ex. '.txt')  \$parts[] - 4 項目の文字列配列で、[0] = ドライブ, [1] = フォルダ名, [2] = ファイル名, [3] = 拡張子	連結した絶対パス名  ex. 'c:\\$sub_dir\$sample.txt'	M064

※ 1 : 「タイトル」と「ワイルドカード」のペアをスラッシュ (/) で区切って指定します。

例えば、テキストファイルと全ファイルを指定する場合は、” TextFile(\*.txt)/\*.txt/All Files(\*.\*)/\*.\*” のように指定します

## プロファイル・アクセス

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>\$_SetProfileSect(\$sect)</b> アクセスするプロファイルのセクションを設定します \$sect - セクション名 本関数を実行しない場合のデフォルトセクション名は「DefSect」です。	—	M061
2	<b>\$_SetProfileVal(\$key, value)</b> プロファイル (プロファイルセクション) へ、値を記録します \$key - キー名称 value - プロファイルへ記録する値	—	M150
3	<b>\$_SetProfileStr(\$key, \$string)</b> プロファイル (プロファイルセクション) へ、文字列を記録します \$key - キー名称 \$string - プロファイルへ記録する文字列	—	M061 M063
4	<b>\$_GetProfileVal(\$key, DefVal)</b> プロファイル (プロファイルセクション) から値を読み出します \$key - キー名称 DefVal - デフォルト値	プロファイルから読み出した値 当該キーが存在しない場合は DefVal を返します。	M150
5	<b>\$_GetProfileStr(\$key, \$DefStr)</b> プロファイル (プロファイルセクション) から文字列を読み出します \$key - キー名称 \$DefStr - デフォルト文字列	プロファイルから読み出した文字列 当該キーが存在しない場合は \$DefStr を返します。	M061 M063

COMポート/メールスロットアクセス (ポート番号=256 はメールスロットを意味します)			
#	関数定義 内容	戻り値/※備考	サンプ プル
1	<b>_ComCreate([PortNo   \$MySlotName])</b> COMポートの初期化を行います。(まだ、オープンはしません) PortNo      - COMポート番号 (COM1～255生成) \$MySlotName - 自メールスロット名 (メールスロット生成) ※未指定のポート情報はプロファイルから読み出されます。 ※引数の指定がない場合は、前回設定値を採用します。	ポート番号 (1～256)	M062 M070 M071 M073 M074 M080
2	<b>_ComCreateByDialog([OldPort])</b> ダイアログにより、ポートのパラメタを設定し、ポートの初期化を行います。 OldPort - 旧ポート番号 ※オープン状態で同一ポートを指定しOKボタンを押した場合は、当該ポートを再オープンします。 ※新たなポートを生成した場合、「OldPort」の指定があれば、(新旧のポート番号が異なる場合) この旧ポートを破棄します。(但し、OldPort=0 の場合は何もしない) ※「OldPort」を指定しないで異なるポートを選択した場合、旧ポートは破棄されずにそのまま残ります。	1～256 - 生成したポート番号 0       - キャンセル/エラー	M070 M071 M072 M080
3	<b>_ComDelete(PortNo)</b> _ComCreate/_ComCreateByDialog で初期化したポートを破棄します。 ポートがオープン状態の場合は、クローズしてから破棄します。 PortNo      - ポート番号 (1～256) ※既に破棄されている場合は何もしません ※ポート情報はプロファイルへ記録されます。	- ※マクロ実行終了時、破棄されていないポートは自動的に破棄されます。	M070 M071 M072 M080
4	<b>_ComOpen (PortNo [, Rate] [, DataLen] [, Parity] [, StopBit])</b> <b>_ComOpen ([RemoteSlotName] [, RemoteHostName])</b> <b>_ComOpen (256)</b> 1) COMポートをオープンします。 PortNo      - COMポート番号 (1～255) Rate        - 通信レート[bps] DataLen     - データ長 (7～8) Parity      - パリティビット(0:なし、1:奇数、2:偶数) StopBit     - ストップビット長 (1～2) 2) メールスロットをオープンします \$RemoteSlotName - 相手のメールスロット名 \$RemoteHostName - 相手のコンピュータ名 (空文字( "" )指定時は自コンピュータ) 3) 現在値でメールスロットをオープン (port = 256 を指定)	- ※当該ポートは、 _ComCreate/_ComCreateByDialog により初期化されていなければ なりません。 ※引数未指定時は、前回値を採用 します。	M062 M070 M071 M072 M080
5	<b>_ComIsOpenPossible (PortNo)</b> ポートがオープン可能か、チェックします。 PortNo      - ポート番号 (1～256)	_TRUE   - オープン可能 _FALSE - オープン不可能	M070 M071 M080
6	<b>_ComIsOpened(PortNo)</b> ポートがオープン状態か、チェックします。 PortNo      - ポート番号 (1～256)	_TRUE   - オープン状態 _FALSE - クローズ状態	M070
7	<b>_ComClose (PortNo)</b> ポートをクローズします。 ポートが既にクローズされている場合は、何もしません。 PortNo      - COMポート番号 (1～256)	-	M062 M070 M071 M072 M080

つづく

8	<b>_ComSendText(PortNo, \$Text1 [, \$Text2]) ... )</b> ポートへテキストデータを送信します。 PortNo - ポート番号 (1~256) Text1~ - 送信するテキスト群	—	M062 M070
9	<b>_ComSendBinary(PortNo, MemBlkNo, loc, len)</b> ポートへバイナリデータを送信します。 PortNo - ポート番号 (1~256) MemBlkNo - メモリブロック番号 loc - 送信データのバイト位置 (メモリブロック内ロケーション) len - 送信データのバイト数	—	M070
10	<b>_ComSendBytes(PortNo, b1 [, b2] ... )</b> ポートへバイトデータ (複数バイト可) を送信します。 PortNo - COMポート番号 (1~256) b1 [, b2] ... - 送信バイトデータ群	—	M070
11	<b>_ComSendPacket(PortNo, MemBlkNo, loc, len)</b> ポートへパケットデータを送信します。 PortNo - ポート番号 (1~256) MemBlkNo - メモリブロック番号 loc - 送信データのバイト位置 (メモリブロック内ロケーション) len - 送信データのバイト数	— ※送信データブロックを DLE・STX~DLE・ETX でサンドイ ッチし、データ中の DLE (0x10) を2バイトの DLE に拡張します	M070
12	<b>_ComGetRate(PortNo)</b> COMポートの設定値 (転送速度[bps]) を取得します PortNo - COMポート番号 (1~255)	転送速度 (1~)	M070
13	<b>_ComGetDataBits(PortNo)</b> COMポートの設定値 (データビット数) を取得します PortNo - COMポート番号 (1~255)	7~8	M070
14	<b>_ComGetParitely(PortNo)</b> COMポートの設定値 (パリティビット) を取得します PortNo - COMポート番号 (1~255)	0 : なし 1 : 奇数パリティ 2 : 偶数パリティ	M070
15	<b>_ComGetStopBits(PortNo)</b> COMポートの設定値 (ストップビット数) を取得します PortNo - COMポート番号 (1~255)	1~2	M070
16	<b>\$_ComGetParam(PortNo)</b> ポートの設定値を文字列で取得します PortNo - ポート番号 (1~256)  ※メールスロット (PortNo=256) の場合は、接続先スロットのパス名を返します。	“COMn, <r>, <d>, <p>, <s>” <r>: 転送レート [bps] <d>: データビット数 (7 or 8) <p>: パリティ (No, Odd or Even) <s>: ストップビット長 (1 or 2)	M070 M071 M080
17	<b>_ComSetDTR(PortNo, fDTR)</b> COMポートのDTR信号を設定します PortNo - COMポート番号 (1~255) fDTR - DTR信号設定値 (_TRUE : 有効状態, _FALSE : 無効状態)	—	M080
18	<b>_ComSetRTS(PortNo, fRTS)</b> COMポートのRTS信号を設定します PortNo - COMポート番号 (1~255) fRTS - RTS信号設定値 (_TRUE : 有効状態, _FALSE : 無効状態)	—	M080
19	<b>_ComGetSignal(PortNo)</b> COMポートの信号状態を取得します。 PortNo - COMポート番号 (1~255)	Bit4(0x10) - CTS 信号状態 Bit5(0x20) - DSR 信号状態 Bit6(0x40) - RING 信号状態 Bit7(0x80) - RLSD 信号状態 いずれも「1」で有効状態	M080

## COMポート／メールスロットイベント登録 (ポート番号=256 はメールスロットを意味します)

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>_ComSetOnRxText (PortNo, \$func)</b> ポートからのテキスト受信時のイベント処理関数登録 PortNo - ポート番号 (1～256) \$func - テキスト受信時に実行する関数名	—	M070
2	<b>_ComSetOnRxCtrl (PortNo, \$func)</b> ポートからの制御コード受信時のイベント処理関数登録 PortNo - ポート番号 (1～256) \$func - 制御コード受信時に実行する関数名	—	M070
3	<b>_ComSetOnRxEsc (PortNo, \$func)</b> ポートからのE S Cシーケンス文字列受信時のイベント処理関数登録 PortNo - ポート番号 (1～256) \$func - E S Cシーケンス受信時に実行する関数名	—	M070
4	<b>_ComSetOnRxPacket (PortNo, \$func, MemBlkNo)</b> ポートからのパケットデータ受信時のイベント処理関数登録 PortNo - ポート番号 (1～256) \$func - パケットデータ受信時に実行する関数名 MemBlkNo - パケットデータを格納するメモリブロック番号	—	M070

## COMポート／メールスロット通信イベント処理関数

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>OnRxText(\$Text)</b> ポートからのテキスト受信時の処理を行います \$Text - 受信したテキストデータ	— ※実際の関数名は、 _ComSetOnRxText() で登録した関 数名となります。	M070
2	<b>OnRxCtrl(Ctrl)</b> ポートからの制御コード受信時の処理を行います Ctrl - 受信した制御コード	— ※実際の関数名は、 _ComSetOnRxCtrl() で登録した関 数名となります。	M070
3	<b>OnRxEsc(\$Esc)</b> ポートからのE S Cシーケンス文字列受信時の処理を行います \$Esc - 受信したE S Cシーケンス文字列	— ※実際の関数名は、 _ComSetOnRxEsc() で登録した関 数名となります。	M070
4	<b>OnRxPacket(len)</b> ポートからのパケットデータ受信時の処理を行います len - 受信したパケットデータのバイト数  ※受信パケットデータは、_ComSetOnRxPacket() で指定したメモリブロックの先頭から格納されます。	— ※実際の関数名は、 _ComSetOnRxPacket() で登録した 関数名となります。	M070

※実際の関数名は、COMポート／メールスロットイベント登録群「\_ComSetOnXXXX()」により指定します。

※上記イベントは、\_DisComEvt()/\_EnaComEvt()により禁止／許可できます。

**XMODEM/YMODEM ファイル転送** (ポート番号=256 はメールスロットを意味します)

#	関数定義 内容	戻り値/※備考	サンプル
1	<u>_ComSendXModemSum</u> (PortNo, \$File) - 128Bytes ブロック, チェックサム <u>_ComSendXModemCrc</u> (PortNo, \$File) - 128Bytes ブロック, CRC <u>_ComSendXModem1K</u> (PortNo, \$File) - 1KBytes ブロック, CRC XMODEM プロトコルによりファイルを送信します。 PortNo - ポート番号 (1~256) \$File - 送信するファイルのパス名	- ※送受信中は状況をログ表示 : - 1024Bytes データ送受信 . - 128Bytes データ送受信 R - リトライ発生 E - 送信終了 C[SUM/CRC/1K] - プロトコル変更	M071 M073
2	<u>_ComRecvXModemSum</u> (PortNo, \$File) - 128Bytes ブロック, チェックサム <u>_ComRecvXModemCrc</u> (PortNo, \$File) - 128Bytes ブロック, CRC <u>_ComRecvXModem1K</u> (PortNo, \$File) - 1KBytes ブロック, CRC XMODEM プロトコルによりファイルを受信します。 PortNo - ポート番号 (1~256) \$File - 受信データを格納するファイルのパス名		M071 M074
3	<u>_ComSendYModem</u> (PortNo, \$Wild [, fSubDir] [, fFolder]) YMODEM プロトコルによりファイルを送信します。 PortNo - ポート番号 (1~256) \$Wild - 送信するファイルのワイルドパス名 fSubDir - サブディレクトリ検索フラグ (TRUE: 検索する, FALSE: 検索しない) fFolder - 送信ファイル名にパス情報を付加するか否かのフラグ TRUE : 送信するファイル名にパス情報を付加する FALSE : 送信するファイル名にパス情報を付加しない	- ※送受信中は状況をログ表示 : - 1024Bytes データ送受信 . - 128Bytes データ送受信 R - リトライ発生 E - 送信終了	M072 M075 M076 M077
4	<u>_ComRecvYModem</u> (PortNo, \$folder, [, fSubDir]) YMODEM プロトコルによりファイルを受信します。 PortNo - ポート番号 (1~256) \$folder - 受信ファイルを格納する先頭フォルダのパス名 fSubDir - 受信ファイル名にパス情報が付加されている場合の動作 TRUE : 先頭フォルダからの相対で、サブフォルダを作成する FALSE : サブフォルダは作成せずに、全て先頭フォルダに格納	※: ファイル名にフォルダパスを付加する場合は、先頭フォルダからの相対パスとなる (ex. .¥SubDir¥ sample.txt)	M072 M078 M079

**標準イベント処理関数**

#	関数定義 内容	戻り値/※備考	サンプル
1	<u>_OnKeyIn</u> (key) キー入力時に実行されます。 key - 入力したキーコード	- ※以下のキーは通知しません。 CTRL+A (0x01): 全テキスト選択 CTRL+C (0x03): 選択テキスト・コピー	M070
2	<u>_OnRClick</u> (x, y, shift, ctrl) マクロウインド上で右クリックした場合に実行されます。 x - クリックしたX位置 y - クリックしたY位置 shift - Shift キーの押下状態 (1: 押下, 0: 非押下) ctrl - Ctrl キーの押下状態 (1: 押下, 0: 非押下)	- ※ x, y はマクロウインドの左上を原点 (0, 0) としたピクセル位置	M010 M070 M071 M072
3	<u>_OnDb1C1k</u> (\$line) マクロウインド上でダブルクリックした場合に実行されます。 \$line - ダブルクリックした位置の行テキスト	- ※ AjcCipSetDb1C1kAct() API で fNtcDb1C1k=TRUE とした場合は実行されません	M070

つづく

4	<b>_OnTmcClose()</b> タイムチャートウインドがクローズされたことを通知します。	— TmcOpen() 実行後ウインドが破棄された時にコールされます。	M120
5	<b>_OnButton(BtnNo)</b> ボタンが押されたことを通知します。 BtnNo - ボタン識別番号 (0～7)	—	M020 M070 M110
6	<b>_OnChkBox(ChkBoxNo, staus)</b> チェックボックスの状態が設定されたことを通知します。 status - チェック状態 (_TRUE:チェックしている, _FALSE:チェックしていない) ※チェック状態を設定時、チェック状態が変化しなくても実行されます。	— ※この関数内から、_ChkBoxSetSts() は実行できません。	M110
7	<b>_OnFileSearch(\$folder)</b> ファイル検索中のフォルダ・パス名を通知します。 \$folder - フォルダ・パス名	_TRUE : ファイル検索を継続する _FALSE : ファイル検索を中止する ※\$_SearchFiles() の実行中にコールされます。	M063
8	<b>_OnTimer(tid)</b> タイマの経過を通知します tid - タイマ識別 ID (1～31)	—	M062
9	<b>_OnMenu(id)</b> ポップアップメニューで項目が選択されたことを通知します。 id - メニュー項目インデックス (0～)	— ※実際の関数名は、_PopupMenu() の\$Func 引数で指定します。	—

※上記関数は、ユーザがマクロテキスト内で、固定の関数名で定義します。

※上記イベントは、\_DisStdEvt()/\_EnaStdEvt()により禁止／許可できます。

### イベントの禁止／許可

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>_DisStdEvt()</b> 標準イベントの実行を禁止します。(※1)	—  ※この関数は、標準イベント関数 (_OnXXX) からはコールできません。	—
2	<b>_EnaStdEvt()</b> 標準イベントの実行を許可します。(※1)		—
3	<b>_GetDisStdEvt()</b> 標準イベント禁止のネストレベルを取得します。	標準イベント禁止のネストレベル = 0 : イベント許可状態 > 0 : イベント禁止状態	—
4	<b>_DisComEvt()</b> COMポート通信イベントの実行を禁止します。(※1)	—  ※この関数は、標準イベント関数 (_OnXXX) からはコールできません。	—
5	<b>_EnaComEvt()</b> COMポート通信イベントの実行を許可します。(※1)		—
6	<b>_GetDisComEvt()</b> COMポート通信イベント禁止のネストレベルを取得します。	COMポート通信イベント禁止のネストレベル = 0 : イベント許可状態 > 0 : イベント禁止状態	—

※1 : イベントの禁止／許可は多重制御します。

つまり、\_DisXXxEvt() と同じ回数 \_EnaXXxEvt() を実行した時に、イベントの実行が許可状態となります。









## ボタン

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>_BtnOpen(BtnNo, \$Caption)</b> ボタンを表示し、使用可能にします。 BtnNo - ボタン識別番号 (0～7) \$Caption - ボタンフェース文字列	— ※ボタンは、コントロールウィンドの上部に表示します。	M020
2	<b>_BtnClose(BtnNo)</b> ボタンを削除します BtnNo - ボタン識別番号 (0～7)	—	M020
3	<b>_BtnEnable(BtnNo, fEnable)</b> ボタンを有効化／無効化します BtnNo - ボタン識別番号 (0～7) fEnable - _TRUE : ボタンを有効化 (押せる状態にする) _FALSE : ボタンを無効化 (押せない状態にする)	—	M020

## チェックボックス

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>_ChkBoxOpen(ChkBoxNo, \$Caption)</b> チェックボックスを表示し、使用可能にします。 ChkBoxNo - チェックボックス識別番号 (0～7) \$Caption - チェックボックス・キャプション文字列	— ※チェックボックスは、コントロールウィンドの上部に表示します。	M010 M070 M110
2	<b>_ChkBoxClose(ChkBoxNo)</b> チェックボックスを削除します ChkBoxNo - チェックボックス識別番号 (0～7)	—	M160
3	<b>_ChkBoxGetSts(ChkBoxNo)</b> チェックボックスのチェック状態を取得します ChkBoxNo - チェックボックス識別番号 (0～7)	チェック状態 _TRUE : チェックしている _FALSE : チェックしていない	M160
4	<b>_ChkBoxSetSts(ChkBoxNo, Status)</b> チェックボックスのチェック状態を取得します ChkBoxNo - チェックボックス識別番号 (0～7) Status - 設定するチェックステータス (_TRUE:チェック, _FALSE:未チェック)	—	M110
5	<b>_ChkBoxEnable(ChkBoxNo, fEnable)</b> チェックボックスを有効化／無効化します ChkBoxNo - チェックボックス識別番号 (0～7) fEnable - _TRUE : チェックボックスを有効化 (チェックできる状態にする) _FALSE : チェックボックスを無効化 (チェックできない状態にする)	—	M160

## タイムチャート・グラフ

#	関数定義 内容	戻り値／※備考	サンプル
1	<b>_TmcOpen([\$Title] [, \$Poroparty] [, x] [, y] [, cx] [, cy])</b> タイムチャートウインドをオープンします \$Title - ウインドタイトル \$Property - プロパティ文字列 (※1) x, y - ウインド表示位置 cx, cy - ウインドのサイズ	— ※既にオープン済の場合は一旦ウインドをクローズしてから再度オープンする。	M120
2	<b>_TmcSetItems(NumOfDataItems)</b> タイムチャート・グラフの データ項目数設定 NumOfDataItems - データ項目数 (1～8)	—	M120
3	<b>_TmcSetRange(Low, High)</b> タイムチャート・グラフのレンジ設定 Low - グラフ低位の値 High - グラフ高位の値 ※グラフのレンジを自動調整する場合は、_TmcSetRange(0, 0); としてください。	— ※Low > High でも OK ※Low = High の場合はグラフレンジを自動調整する	M120
4	<b>_TmcPutData(d1 [, d2] . . . [, d8])</b> <b>_TmcPutData(data[])</b> タイムチャート・グラフへ データ投与 d1 . . . d8 - グラフへ投与するデータ data[] - グラフへ投与するデータの配列 ([1] : d1 . . . [8] : d8)	— ※各データ d1～d8 の描画色は以下 d1:  d5:  d2:  d6:  d3:  d7:  d4:  d8: 	M120
5	<b>_TmcClear()</b> タイムチャート・グラフの クリアー	—	M120
6	<b>_TmcWndMove(x, y)</b> タイムチャート ウインドの移動 x, y - 移動先のスクリーン座標	—	M120
7	<b>_TmcWndSize(cx, cy)</b> タイムチャート 。ウインドサイズ設定 cx, cy - 設定するウインドのサイズ	—	M120
8	<b>_TmcClose()</b> タイムチャートウインドを クローズします	—	M120

※1 : 「タイムチャート・コントロール」の「キャプション文字列によるプロパティ設定」を参照

## その他


#	関数定義 内容	戻り値／※備考	サンプル
1	<b>_ShowText(\$Text)</b> コントロールウインドの右上にテキストを表示します \$Text - 表示するテキスト文字列	—	M070 M071 M080
2	<b>_GetWndPosX()</b> コントロールウインドの左端位置 (スクリーン座標) を取得します。	コントロールウインドの左端位置	M160
3	<b>_GetWndPosY()</b> コントロールウインドの上端位置 (スクリーン座標) を取得します。	コントロールウインドの上端位置	M160

つづく



4	<b>_GetWndWidth()</b> コントロールウインドの幅を取得します	コントロールウインドの幅 (ピクセル数)	M160
5	<b>_GetWndHeight()</b> コントロールウインドの高さを取得します	コントロールウインドの高さ (ピクセル数)	M160
6	<b>_GetCharWidth()</b> 文字の幅を取得します	半角文字の幅 (平均ピクセル数)	M160
7	<b>_GetCharHeight()</b> 文字の高さを取得します	文字の高さ (ピクセル数)	M160
8	<b>_GetLineHeight()</b> 行の高さを取得します	行の高さ (ピクセル数)	M160
9	<b>_Exit( [\$Text] )</b> マクロの実行を終了します \$Text - このテキストは親ウインドから、AjcCipGetExitText()により取得できます。 ※_Exit()で指定するテキストは、言語上は何の効果もありません。 テキストの扱いは、このコントロールを使用するアプリケーションに依存します。 例えば、Exit()で、次に実行するマクロテキストファイルを指定したりできます。	—	M071

## デバッグ機能

#	関数定義 内容	戻り値/※備考	サン プル
1	<b>_VarDump([MaxArrNum])</b> トレース表示ウインドへ全ての変数情報を表示します MaxArrNum - 配列の最大表示項目数 (未指定時は10を仮定)	—	—
2	<b>_Trace (\$fmt [, arg] ...)</b> トレース表示ウインドへ書式文字列を表示します。 \$fmt - 書式文字列 (C言語のprintf()とほぼ同じ) arg - 書式(%...)に対応した引数	—	—
3	<b>_Pause()</b> ※1 全ての実行を一時停止します。 ※メイン処理およびイベント処理の全てが停止状態となります。 ※ステップトレースウインド未オープンの場合、メッセージボックスが表示され、OKボタンを押すと実行を再開します。 ※ステップトレースウインドがオープンされている場合は、停止した旨を表示し、実行継続ボタン(  )で再開します。	—	—
4	<b>_Stop()</b> ※1 メイン処理の実行を一時停止します。 ※メイン処理だけが停止状態となります。イベントは実行されます。 ※_Restart()を実行すると、メイン処理を再開します。	—	M010 M070 M080
5	<b>_Restart ()</b> _Stop()により中断していたメイン処理を再開します。	—	M010 M070 M080
6	<b>_StepTrace(fEnable)</b> ステップトレースの表示を許可/禁止します。 fEnable : TRUE - ステップトレースの表示を許可 FALSE - ステップトレースの表示を禁止	— ※ステップトレース。ウインドが開かれていない場合は何もしません。	

※1 : \_Pause()や\_Stop()は、\_Pause()/\_Stop()を含むステップの次のステップ(\_Stop()は次のメインコードステップ)で停止します。  
(\_Pause()/\_Stop()も数値型関数(単に0を返すだけです)なので、\_Pause()/\_Stop()を含む数式が実行された後に停止します)

## サンプルマクロテキスト一覧

マクロテキスト	内容	備考
M010. txt	右クリックとポップアップメニュー，ユーザイベントの実行	
M020. txt	ロケート&プリント	
M030. txt	スクリーン情報	
M040. txt	配列操作	
M050. txt	メモリブロックの各操作	
M051. txt	メモリブロック（整数）アクセス	
M052. txt	メモリブロック（実数）アクセス	
M060. txt	ファイルのコピー&ベリファイ	
M061. txt	テキストファイルを読み出して 指定した文字列を含む行を表示する	
M062. txt	テキストファイルを読み出して 100ms/行の速度でCOMポートへ送信	
M063. txt	指定フォルダ下のファイルを検索	
M064. txt	ファイルパスの分解，組み立て	
M070. txt	シリアル送受信とイベントの実行	
M071. txt	XMODEM送受信	
M072. txt	YMODEM送受信	
M073. txt	XMODEM送信	
M074. txt	XMODEM受信	
M075. txt	YMODEM送信（フォルダ下の全ファイル）	
M076. txt	YMODEM送信（フォルダ下の全「.C」ファイル	
M077. txt	YMODEM送信（単一ファイル）	
M078. txt	YMODEM受信（サブフォルダを作成しない）	
M079. txt	YMODEM受信（サブフォルダを作成する）	
M080. txt	COMポートの信号設定と読み出し	
M090. txt	時間，ウェイト	
M100. txt	算術演算	
M101. txt	3点から円の中心を求める	
M110. txt	ボタンとチェックボックス	
M120. txt	タイムチャート	
M130. txt	文字列操作	
M140. txt	フォルダ作成，パスチェック	
M150. txt	プロファイル（数値）の読み出し／書き込み	
M160. txt	チェックボックスの生成／消去	

## 5.4. デバッグ機能

デバッグ用の機能として、以下のAPIと内部関数があります。

### API

API名	機能	備考
AjcCipCreateTraceWindow	トレース表示ウインドを開く	
AjcCipDestroyTraceWindow	トレース表示ウインドを閉じる	
AjcCipCreateStepWindow	ステップトレース・ウインドを開く	
AjcCipDestroyStepWindow	ステップトレース・ウインドを閉じる	

### 内部関数

関数名	機能	備考
_VarDump	トレース表示ウインドへ全ての変数情報を表示します	
_Trace	トレース表示ウインドへ書式文字列を表示します。	
_Pause	全ての実行を一時停止します	
_Stop	メイン処理の実行を一時停止します	イベントは実行されます
_Restart	_Stop()により中断していたメイン処理を再開します	
_StepTrace	ステップトレースの表示を許可／禁止します。	

### 5.4.1. トレース表示

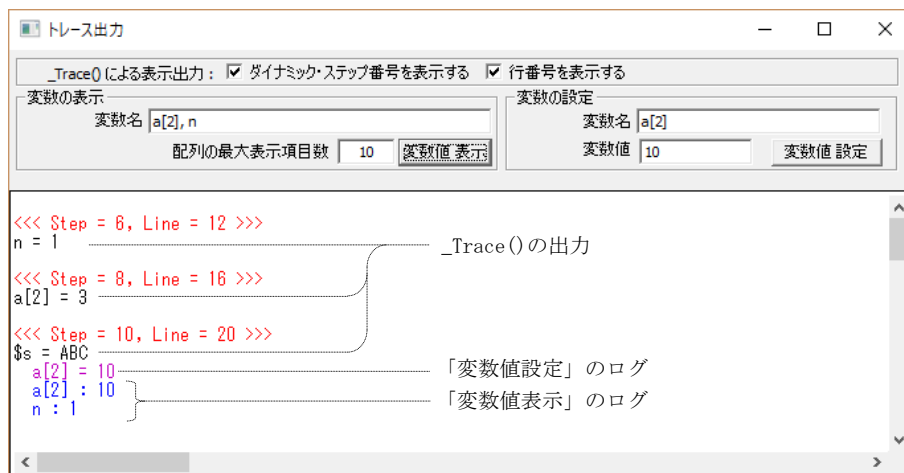
デバッグ用の表示ウインドとしてトレース表示ウインドを開くことができます。

トレース表示ウインドは、\_Trace()実行時に自動的に開きます。

また、AjcCipCreateTraceWindow() APIで強制的に開くこともできます。

トレース表示ウインドは、\_Trace()内部関数による出力内容を表示します。

\_Trace()の構文は、C言語のprintf()と同じで、プログラムの内部情報を表示することができます。



2つのチェックボックス「ダイナミックステップ番号を表示する」と「行番号を表示する」は、\_Trace()による表示出力に、ダイナミックステップ番号や行番号情報を付加します。(上記表示例中の「<<< Step = 6, Line = 12 >>>」等)

ダイナミックステップ番号とは、実行開始時のステップを1とした、実行開始時からの実行ステップ数を示します。

行番号は、マクロテキストファイルの行番号を示します。

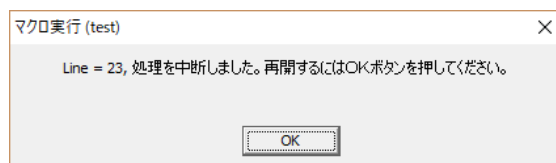
また、\_Pause()によりプログラムを停止した状態で、変数の表示や設定が可能です。

「変数の表示」中の「変数名」に、表示したい変数を入力し、「変数値表示」ボタンを押すと、トレース表示ウインドに当該変数の値が表示されます。複数の変数を表示する場合は、カンマ(,)で区切って変数名を入力します。

「配列の最大表示項目数」は、変数が配列である場合の表示する最大の配列要素数を指定します。

「変数の設定」中の「変数名」に設定したい変数名を、「変数値」に設定する値を入力し、「変数値設定」ボタンを押すと、当該変数に値が設定されます。変数の設定では、1つの変数名だけを入力してください。また、配列を指定する場合は、配列の要素を特定するように変数名を入力します。(ex. arr[3])

\_Pause()では、以下のようなメッセージが表示されますので、「OK」ボタンを押すと実行を再開します。



※ ステップトレース・ウインドを開いている場合は上記メッセージは表示されません。

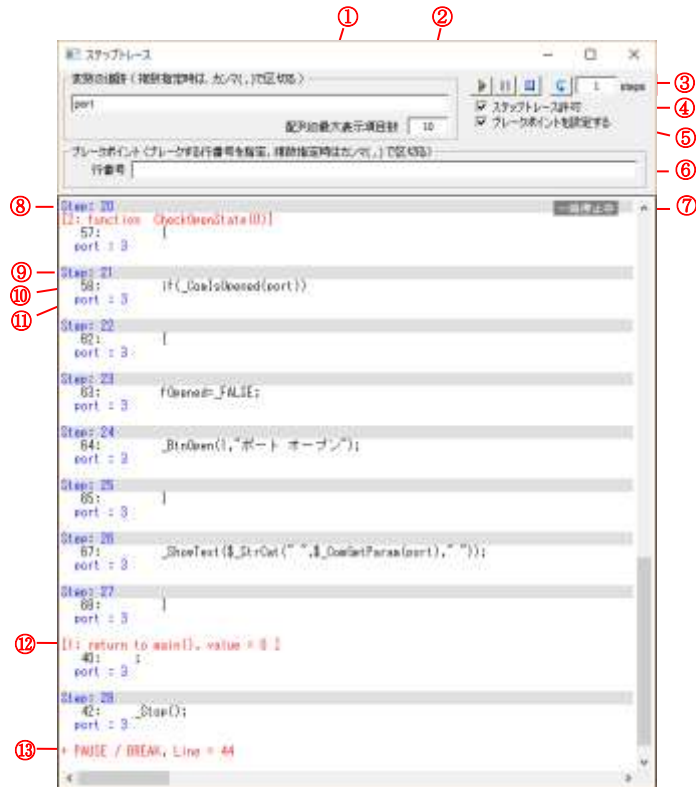
この場合は、ステップトレース・ウインドの実行継続ボタンを押すことにより実行を再開します。







### 5.4.2. ステップトレース

AjcCipCreateStepWindow () A P I により、ステップトレース・ウインドを開くことができます。

但し、ステップトレース・ウインドはマクロテキストの実行中には開くことができず、また、マクロ実行中はウインドを閉じることができません。(つまり、マクロテキストを実行する前に開いておかねばなりません)

ステップトレース・ウインドの外観は以下のとおりです。



#	内容
1	実行ステップ毎に表示する変数を指定
2	①で指定した変数が配列の場合、表示する最大項要素数
3	マクロテキストの実行操作ボタンとステップ数  : 実行継続 (マクロテキストの実行再開)  : 一時停止  : 停止 (マクロテキストの実行中止)  : ステップ実行 (指定ステップ数実行後に、再び一時停止状態となる) <input type="text" value="1"/> steps : ステップ実行するステップ数
4	ステップトレースの許可(チェック)/禁止(未チェック) ステップトレースを禁止した場合は、何も表示なくなります。 _StepTrace()内部関数でも同様の操作が可能です。
5	ブレークポイントの有効化(チェック)/無効化(未チェック), ⑥に入力した行番号でブレークします。
6	ブレークする行番号の入力(⑤が未チェックの場合は表示されません)
7	実行状態の表示(実行中/一時停止中/停止中(マクロテキストが実行されていない状態))
8	関数の実行開始を示す表示(この例では、CheckOpenState()に実行が開始したことを示す) 先頭の数字「n:」は、関数のネストレベルを意味します。
9	ダイナミックステップ番号
10	マクロテキストの行番号と、実行ステップ命令の表示
11	①で指定した変数値の表示
12	関数からの戻り値を示す表示(この例では、戻り値=0で、main()関数へ戻ったことを示す) 先頭の数字「n:」は、関数のネストレベルを意味します。
13	一時停止状態となったことを示す。以下の条件で一時停止状態となります。 ・Pause()が実行された ・⑥で指定した行番号(ブレークポイント)に達した ・一時停止ボタン(  ) 押下 ・ステップ実行ボタン(  ) によるステップ実行満了

## 5.5. コントロールのスタイル

V T 1 0 0エミュレーションウインド・コントロール (AjrCtrlVT100 クラス) と同じです。

## 5.6. キャプション文字列によるプロパティの設定

V T 1 0 0エミュレーションウインド・コントロール (AjrCtrlVT100 クラス) と同じです。

## 5.7. 構造体

### 5.7.1. 引数情報(AJCCIPAGV)

```
typedef struct {
    UI      seq;           // 引数の順序番号 (0 ~)
    EAJCCIPTYP typ;        // タイプ
    AJCCIP_UVAL val;       // 引数値
    BOOL     fArr;         // 配列フラグ
} AJCCIPAGV, *PAJCCIPAGV;
typedef const AJCCIPAGV *PCAJCCIPAGV;
```

※ typ には、以下の引数タイプ値が設定されています。

- AJCIPT\_INTEGER - 整数
- AJCIPT\_REAL - 実数
- AJCIPT\_STRING - 文字列

※ fArr は、引数に配列全体が指定されたことを示すフラグです。(TRUE で引数=配列全体)

※ val は、以下の情報を格納した共用体です。

```
typedef union {
    SLL     sll;           // 整数 (符号付き/符号無し 兼用)
    double  dbl;           // 実数
    C_BCP   pStr;          // 文字列 (へのポインタ)
    struct {HAJCVMG hVmg; BCP pName;} arr; // 配列 (VMG インスタンス, 変数名)
} AJCCIP_UVAL, *PAJCCIP_UVAL;
typedef const AJCCIP_UVAL *PCAJCCIP_UVAL;
```

※ typ, fArr により、AJCCIP\_UVAL 共用体中、以下の項目を使用します。

#	fArr	typ	AJCCIP_UVAL	備考
1	FALSE	AJCIPT_INTEGER	sll	整数(64 Bits)
2	FALSE	AJCIPT_REAL	dbl	実数(64 Bits)
3	FALSE	AJCIPT_STRING	pStr	文字列へのポインタ
4	TRUE	—	arr	変数管理のインスタンスハンドルと変数名

### 5.7.2. 関数呼び出し情報(AJCCIPCALL)

```
typedef struct {
    BCP      pFunName;          // ファンクション名へのポインタ
    UI       lno;              // 行番号
    PAJCCIPTKN pReturn;        // 戻りアドレス
    HAJCVMG  rchVmg;           // 戻り値を格納するVMG ハンドル
    BCP      rcpSym;           // 戻り値を格納する変数へのポインタ
    ULL      rcIx;             // 戻り値を格納する配列インデックス
    UI       nArg;             // 引数の個数
    HAJCFQUE hQueAgv;          // 引数値群 (「AJCCIPAGV」の固定長キュー)
} AJCCIPCALL, *PAJCCIPCALL;
typedef const AJCCIPCALL *PCAJCCIPCALL;
```

※ pFunName に関数名 (文字列) のアドレス、lno にマクロテキストファイルの行番号 (1 ～) が設定されています。

※ rchVmg～rcIx は、関数の戻り値を受け取る引数の変数管理インスタンス (rchVmg)、変数名へのポインタ (rcpSym)、配列インデックス (rcIx) が設定されています。関数が戻り値を返す場合は、rchVmg～rcIx により変数値を設定します。

戻り値を受け取る変数がない場合は、rchVmg=NULL が設定されています。

※その他のメンバは、内部的に使用されますので無視してください。

### 5.7.3. トークン情報(AJCCIPTKN)

```
typedef struct _AJCCIPTKN {
    EAJTKCODE tkn;            // トークンコード
    UB        flgCtrl;        // 'do', 'while', 'function' 関連のフラグ情報
    UB        filler;         // -
    UW        lno;            // 行番号
    union {
        struct _AJCCIPTKN *pJmp; // ジャンプ先トークンへのポインタ
        BCP      pStr;          // 文字列へのポインタ
        SLL      val;          // 数値定数 (整数)
        double   dbl;          // "      (実数)
    } u;
} AJCCIPTKN, *PAJCCIPTKN;
typedef const AJCCIPTKN *PCAJCCIPTKN;
```

pTkn->lno に次に実行するマクロテキストの行番号 (1 ～) が設定されています。

その他のメンバは、内部的に使用されますので無視してください。

## 5.8. API の実行順序

マクロインタプリタを実行するには、以下の手順でAPIを実行します。

例えば「実行開始」ボタン(仮に、IDC\_CMD\_RUN とする)等の処理で、以下のように記述します。

```
static BOOL      fRun = FALSE;          // マクロ実行中を示すフラグ
static BOOL      fExit = FALSE;        // ウインド破棄を示すフラグ

AJC_DLGPROC(Main, IDC_CMD_RUN )
{
    // マクロテキストの読み出し
    if (AjcCipLoad(hWndCip, InpFile)) {
        PAJCCIPCALL pCall;
        // 関数呼び出し情報設定
        if (pCall = AjcCipCallInfoInit(hWndCip, "main")) {          // 呼び出し情報生成

            // 呼び出す関数への引数を設定
            {
                AjcCipCallInfoInsInteger(hWndCip, pCall, 1234);      // 引数=整数
                AjcCipCallInfoInsReal   (hWndCip, pCall, 3.14159);    // 引数=実数
                AjcCipCallInfoInsString (hWndCip, pCall, "string");    // 引数=文字列
            }
            // 関数に引数がある場合のみ
            // 引数のタイプに合わせて、
            // 引数の個数分だけ記述する
            //
            //
            //
            // マクロテキストの実行
            fRun = TRUE;
            if (AjcCipExec(hWndCip, pCall)) AjcCipPrintF(hWndCip, "¥n マクロの実行を終了しました。¥n");
            else AjcCipPrintF(hWndCip, "¥n マクロの実行中にエラーが発生しました。¥n");
            AjcCipCallInfoDelete(hWndCip, pCall);
            fRun = FALSE;
            // プログラム終了判断
            if (fExit) DestroyWindow(hDlg);    // ①
        }
        else AjcCipPrintF(hWndCip, "マクロテキストに main() 関数がありません。¥n");
    }
    else AjcCipPrintF(hWndCip, "マクロの読み出しを失敗しました。¥n");

    return TRUE;
}
```

※ AjcCipExec() は、マクロテキストの実行が終了するまで制御を返しません。(但し、内部でAjcDoEvent()によりWindowsのイベントは処理します)

※ マクロテキストの実行中 (AjcCipExec()の実行中) は、本コントロール (ウインド) を破棄しないでください。

上記例では、AjcCipExec()の実行中は「fRun = TRUE」とし、この間はコントロールを破棄できないことを示しています。

例えば本コントロールを含むダイアログボックスで、「fRun = TRUE」の時に、をクローズしようとした場合は、IDCANCEL を以下のように処理します。

```
AJC_DLGPROC(Main, IDCANCEL )
{
    if (fRun) {
        AjcCipStop(hWndCip);
        fExit = TRUE;
    }
    else {
        DestroyWindow(hDlg);
    }
    return TRUE;
}
```

つまり、マクロテキスト実行中の場合、IDCANCEL ではウインドを破棄せずに、AjcCipStop()によりマクロテキストの実行を中止し「fExit = TRUE」とします。

これにより、AjcCipExec()は制御を返すので、この後にウインドを破棄するようにします。(上図の①)

本コントロールを (ダイアログではなく) 通常ウインドの子ウインドとした場合は、WM\_CLOSE メッセージで同様の処理を行います。

```
AJC_WNDPROC(Main, WM_CLOSE )
{
    if (fRun) {
        AjcCipStop(hWndCip);
        fExit = TRUE;
    }
    else {
        DestroyWindow(hwnd);
    }
    return 0;
}
```

## 5.9. サポートAPI

C言語ライクなマクロ・インタプリタのサポートAPI一覧を以下に示します。

#	関数名	内容
1	AjcCipSetOption	オプションの設定
2	AjcCipSetDbIClkAct	ダブルクリック時の動作設定
3	AjcCipLoad	コード（マクロテキストファイル）の読み出し
4	AjcCipExec	コード（マクロテキスト）の実行
5	AjcCipEvent	ユーザイベントの関数の実行
6	AjcCipStop	コード（マクロテキスト）の実行を中止する
7	AjcCipGetDroppedFile	ドロップされたファイル名取得
8	AjcCipGetDroppedDir[Ex]	ドロップされたディレクトリ名取得
9	AjcCipPrintF	ウインドへ書式文字列表示
10	AjcCipNtcErrByUser	ユーザからのエラー通知
11	AjcCipGetExitText	_Exit()マクロ関数により指定されたテキストの取得
12	AjcCipCallInfoInit	関数呼び出し情報の初期化
13	AjcCipCallInfoDelete	関数呼び出し情報の破棄
14	AjcCipCallInfoInsInteger	関数呼び出し情報へ引数（整数）追加
15	AjcCipCallInfoInsReal	関数呼び出し情報へ引数（実数）追加
16	AjcCipCallInfoInsString	関数呼び出し情報へ引数（文字列）追加
17	AjcCipGetRetInt	関数の戻り値取得（整数）
18	AjcCipGetRetReal	関数の戻り値取得（実数）
19	AjcCipGetRetStr	関数の戻り値取得（文字列）
20	AjcCipCreateTraceWindow	トレース表示 ウインド生成
21	AjcCipDestroyTraceWindow	トレース表示 ウインド破棄
22	AjcCipCreateStepWindow	ステップ・トレース ウインド生成
23	AjcCipDestroyStepWindow	ステップ・トレース ウインド破棄
24	AjcCipEnableStepTrace	ステップ・トレースの禁止／許可
25	AjcCipGet VT00Handle	VT100 エミュレーションウインドのハンドル取得
26	AjcCipGetStatus	ステータス情報取得

※ APIは全てバイト文字用です（ワイド文字には対応していません）



**5.9.1. オプションの設定 (AjcCipSetOption)**

**形 式** : BOOL AjcCipSetOption (HWND hwnd, BOOL fEnableOption, EAJCCIPTYP NumType, BOOL fSigned);

**引 数** : hwnd                    - コントロールのウインドハンドル  
 fEnableOption        - 「option」文の解析許可 (TRUE:解析する, FALSE:解析しない)  
 NumType              - 数値のタイプ (AJCIPT\_INTEGER or AJCIPT\_REAL)  
 fSigned               - 整数「AJCIPT\_INTEGER」時の符号の有無 (TRUE:符号あり, FALSE:符号無)

**説 明** : オプション(数値のタイプ)の設定を行います。  
 以下のように、数値オプションが設定されます。

#	NumType	fSigned	設定内容
1	AJCIPT_INTEGER	TRUE	符号付き整数
2	AJCIPT_INTEGER	FALSE	符号無し整数
3	AJCIPT_REAL	—	実数

fEnableOption=TRUE とした場合は、マクロソース先頭の option 文で、オプションの設定を変更可能です。  
 fEnableOption=FALSE とした場合は、マクロソースの先頭に option 文を記述できません。

**戻り値** : TRUE - 成功  
 FALSE - 失敗

**5.9.2. ダブルクリック時の動作設定 (AjcCipSetDb1C1kAct)**

**形 式** : BOOL AjcCipSetDb1C1kAct (HWND hwnd, BOOL fNtcDb1C1k);

**引 数** : hwnd                    - コントロールのウインドハンドル  
 fNtcDb1C1k           - ダブルクリック時の動作指定  
                          TRUE - 親ウインドにダブルクリックを通知する  
                          FALSE - マクロテキスト内の「\_OnDb1C1k()」を実行する (デフォルト)

**説 明** : ウインドをダブルクリックした時の動作を設定します。  
 fNtcDb1C1k=TRUE とした場合は、ダブルクリックイベント (AJCCIPN\_DBLCLK) を親ウインドに通知します。  
 この場合、マクロテキスト中の「\_OnDb1C1k()」は実行されません。  
 fNtcDb1C1k=FALSE とした場合は、ダブルクリックイベント (AJCCIPN\_DBLCLK) を親ウインドに通知しないで、マクロテキスト中に「\_OnDb1C1k()」関数があればこれを実行します。

**戻り値** : TRUE - 成功  
 FALSE - 失敗

**5.9.3. コード (マクロテキストファイル) の読み出し (AjcCipLoad)**

**形 式** : BOOL AjcCipLoad (HWND hwnd, C\_BCP pMacFile);

**引 数** : hwnd                    - コントロールのウインドハンドル  
 pMacFile               - マクロテキストファイルのパス名

**説 明** : マクロテキストファイルの読み出しと簡単な構文チェックを行います。  
 エラーを検出した場合は、エラー通知コールバック関数によりエラーの内容が通知されます。  
 マクロテキストファイルの読み出しと構文チェックでエラーが検出された場合は、AjcCipExec() を実行できません。

**戻り値** : TRUE - 成功  
 FALSE - 失敗

**5.9.4. コード（マクロテキスト）の実行（AjcCipExec）**

**形 式** : `BOOL AjcCipExec (HWND hwnd, PAJCCIPCALL pCall);`

**引 数** : `hwnd`                - コントロールのウインドハンドル  
          `pCall`            - 関数呼び出し情報へのポインタ

**説 明** : マクロテキスト内の関数を実行します。  
 呼び出す関数は、あらかじめ `AjcCipCallInfoInit()` により設定しておきます。  
`AjcCipLoad()` により、マクロコードの読み出しが正常に終了していなければなりません。

この API は、マクロテキストが実行されていない状態において、マクロコード中の開始関数 (`main()` 等) を実行する場合に使用します。

**戻り値** : `TRUE` - 成功  
          `FALSE` - 失敗

**5.9.5. ユーザイベント関数の実行（AjcCipEvent）**

**形 式** : `BOOL AjcCipEvent (HWND hwnd, PAJCCIPCALL pCall);`

**引 数** : `hwnd`                - コントロールのウインドハンドル  
          `pCall`            - イベント関数呼び出し情報へのポインタ

**説 明** : マクロテキスト内のイベント処理関数を実行します。  
 呼び出す関数は、あらかじめ `AjcCipCallInfoInit()` により設定しておきます。  
`AjcCipExec()` により、マクロコードが実行状態でなければなりません。

この API は、マクロテキストが実行されている状態において、マクロコード中の特定の関数を実行することによりイベントの実行を行います。（イベント処理関数は、非同期にマクロのステップ実行の間に割り込んで実行されます）  
 このイベントは、全て保留され、`AjcCipEvent()` を呼んだ回数だけイベント処理関数が実行されます。  
 このイベント処理は、標準イベントとして実行され、イベントを禁止／許可するには、マクロテキスト内において `_DisStdEvt()` / `_EnaStdEvt()` を実行します。

**戻り値** : `TRUE` - 正常  
          `FALSE` - 失敗

**備 考** : この API を実行した場合、`AjcCipCallInfoInit()` 作成した関数呼び出し情報を、`AjcCipCallInfoDelete()` で削除しないでください。呼び出し情報のリソース解放は、内部で実行されます。  
`AjcCipEvent()` の実装例は、サンプルプログラム 1 (`S_Cip01.c`) を参照してください。

**5.9.6. コード（マクロテキスト）の実行中止（AjcCipStop）**

**形 式** : `BOOL AjcCipStop (HWND hwnd);`

**引 数** : `hwnd`                - コントロールのウインドハンドル

**説 明** : マクロテキストの実行を中止します。

**戻り値** : `TRUE` - 成功  
          `FALSE` - 失敗

**5.9.7. ドロップされたファイル名取得(AjcCipGetDroppedFile)**

**形 式** : `BOOL AjcCipGetDroppedFile (HWND hwnd, UT buf[MAX_PATH]);`

**引 数** : `hwnd`        - コントロールのウインドハンドル  
          `buf`        - ファイルのパス名を格納するバッファのアドレス

**説 明** : マクロコードからの表示出力用ウインドへドラッグ&ドロップされたファイルのパス名を取得します。  
 本関数は、通知メッセージ (AJCCIPN\_DROPFILE) が通知された場合、ドロップされたファイルのパス名を取得するために実行します。  
 複数のファイルがドロップされた場合は、lParam で通知されたファイルの個数分だけ本関数を実行します。

**戻り値** : `TRUE` - ファイルのパス名をバッファに格納した  
          `FALSE` - ドロップされたファイル名なし

**5.9.8. ドロップされたディレクトリ名取得(AjcCipGetDroppedDir[Ex])**

**形 式** : `BOOL AjcCipGetDroppedDir (HWND hwnd, UB buf[MAX_PATH]);`  
          `BOOL AjcCipGetDroppedDirEx (HWND hwnd, UB buf[MAX_PATH] , BOOL fTailIsDelimiter);`

**引 数** : `hwnd`                - コントロールのウインドハンドル  
          `buf`                - ファイルのパス名を格納するバッファのアドレス  
          `fTailIsDelimiter` - `TRUE` : ディレクトリパス名の末尾に「¥」を付加する  
                             `FALSE` : ディレクトリパス名の末尾に「¥」を付加しない

**説 明** : マクロコードからの表示出力用ウインドへドラッグ&ドロップされたディレクトリのパス名を取得します。  
 本関数は、通知メッセージ (AJCCIPN\_DROPDIR) が通知された場合、ドロップされたディレクトリのパス名を取得するために実行します。  
 複数のディレクトリがドロップされた場合は、lParam で通知されたディレクトリの個数分だけ本関数を実行します。

**戻り値** : `TRUE` - ディレクトリのパス名をバッファに格納した  
          `FALSE` - ドロップされたディレクトリ名なし

**5.9.9. ウインドへ書式文字列表示 (AjcCipPrintf)**

**形 式** : `BOOL AjcCipPrintf (HWND hwnd, C_BCP pFmt, ...);`

**引 数** : `hwnd`                - コントロールのウインドハンドル  
          `pFmt`              - 書式文字列 (printf() とおなじ)

**説 明** : マクロコードからの表示出力用ウインドへ書式生成した文字列を出力します。  
 出力する文字列は、最大で 1 2 8 3 バイトまでで、長くなった場合はカットされます。

**戻り値** : `TRUE` - 成功  
          `FALSE` - 失敗

**5.9.10. ユーザ関数からのエラー通知 (AjcCipNtcErrByUser)**

**形 式** : BOOL AjcCipNtcErrByUser (HWND hwnd, PCAJCCIPCALL pCall, C\_BCP pFmt, ...);

**引 数** : hwnd                - コントロールのウインドハンドル  
          pCall            - 呼び出し情報へのポインタ  
          pFmt            - 書式文字列 (printf() と同じ)

**説 明** : 関数実行用コールバック関数 (*cbFunc*) からのエラー通知関数です。  
 通知する文字列は、最大で 1 2 8 3 バイトまでで、長くなった場合はカットされます。  
 pCall は、関数実行用コールバック関数 (*cbFunc*) へ通知された pCall 引数を指定します。  
 本関数では、以下のような文字列を作成し、エラー通知用コールバック関数 (*cbErr*) をコールします。

作成する文字列: “ファイルパス名(行番号): 書式生成した文字列”

**戻り値** : TRUE - 正常  
          FALSE - エラー

**5.9.11. \_Exit()マクロ関数により指定されたテキストの取得 (AjcCipGetExitText)**

**形 式** : C\_BCP AjcCipGetExitText (HWND hwnd);

**引 数** : hwnd                - コントロールのウインドハンドル

**説 明** : マクロテキスト内の関数「\_Exit()」で指定された引数「\$Text」を取得します。  
 \_Exit() で引数が設定されていない場合は空文字列(“”)を返します。

**戻り値** : ≠NULL - 正常 (\_Exit() で指定されたテキストへのポインタ)  
          =NULL - エラー

**5.9.12. 関数呼び出し情報の初期化 (AjcCipCallInfoInit)**

- 形 式** : PAJCCIPCALL AjcCipCallInfoInit (HWND hwnd, C\_BCP pFuncName);
- 引 数** : hwnd                    - コントロールのウインドハンドル  
pFuncName                - 関数名へのポインタ
- 説 明** : プログラムから、マクロコード内の関数を呼び出すための、関数呼び出し情報を生成し初期化します。
- 戻り値** : ≠NULL - 成功 (生成した関数呼び出し情報のアドレス)  
=NULL - 失敗
- 備 考** : プログラムから、マクロコード内の関数を呼び出すには、以下の A P I を使用し、関数呼び出し情報を作成し AjcCipExec() により、当該関数を実行します。

#	関数名	内容
1	AjcCipCallInfoInit	マクロコード内の関数を呼び出すための、関数呼び出し情報を生成し初期化します。
2	AjcCipCallInfoInsInteger	関数呼び出し情報へ引数 (整数) 追加
3	AjcCipCallInfoInsReal	関数呼び出し情報へ引数 (実数) 追加
4	AjcCipCallInfoInsString	関数呼び出し情報へ引数 (文字列) 追加
5	AjcCipCallInfoDelete	関数呼び出し情報の破棄

AjcCipCallInfoInit() により関数呼び出し情報を作成したら、AjcCipCallInfoInsInteger(), AjcCipCallInfoInsReal() あるいは、AjcCipCallInfoInsString() により関数へ渡す引数を設定します。

引数は、マクロコード内の呼び出す関数における引数の個数と型が一致していなければなりません。

次に、AjcCipExec() を呼び出して、マクロコード内の関数を実行します。

実行が終了したら (AjcCipExec() が戻ったら) AjcCipCallInfoDelete() で、関数呼び出し情報を破棄します。

(例)

**マクロコード内の関数**

```
function func(n, $s)
{
    . . .
}
```

**func() の呼び出し手順 (数値タイプ=実数の場合)**

```
PAJCCIPCALL pCall = NULL;
if (pCall = AjcCipCallInfoInit("func")) {           // 関数呼び出し情報初期化
    AjcCipCallInfoInsReal (pCall, 5.67);             // 第1引数設定 (実数)
    AjcCipCallInfoInsString(pCall, "ABCDEFGH ");     // 第2引数設定 (文字列)
    AjcCipExec (hCip, pCall);                       // マクロコード内関数実行
    AjcCipCallInfoDelete(pCall);                     // 関数呼び出し情報破棄
}
```

**5.9.13. 関数呼び出し情報の破棄 (AjcCipCallInfoDelete)**

- 形 式** : BOOL AjcCipCallInfoDelete (HWND hwnd, PAJCCIPCALL pCallInfo);
- 引 数** : hwnd                    - コントロールのウインドハンドル  
pCallInfo                - 関数呼び出し情報のアドレス
- 説 明** : AjcCipCallInfoInit() で作成した関数呼び出し情報のリソースを解放します。
- 戻り値** : TRUE - 正常  
FALSE - エラー

**5.9.14. 関数呼び出し情報へ引数（整数）追加（AjcCipCallInfoInsInteger）**

**形 式** : BOOL AjcCipCallInfoInsInteger (HWND hwnd, PAJCCIPCALL pCallInfo, SLL value);

**引 数** : hwnd                    - コントロールのウインドハンドル  
           pCallInfo         - 関数呼び出し情報のアドレス  
           value             - 追加する引数の値（整数）

**説 明** : 関数呼び出し情報に引数（整数）を追加します。

**戻り値** : TRUE - 成功  
           FALSE - 失敗

**5.9.15. 関数呼び出し情報へ引数（実数）追加（AjcCipCallInfoInsInteger）**

**形 式** : BOOL AjcCipCallInfoInsReal (HWND hwnd, PAJCCIPCALL pCallInfo, double value);

**引 数** : hwnd                    - コントロールのウインドハンドル  
           pCallInfo         - 関数呼び出し情報のアドレス  
           value             - 追加する引数の値（実数）

**説 明** : 関数呼び出し情報に引数（実数）を追加します。

**戻り値** : TRUE - 成功  
           FALSE - 失敗

**5.9.16. 関数呼び出し情報へ引数（文字列）追加（AjcCipCallInfoInsInteger）**

**形 式** : BOOL AjcCipCallInfoInsString (HWND hwnd, PAJCCIPCALL pCallInfo, C\_BCP pStr);

**引 数** : hwnd                    - コントロールのウインドハンドル  
           pCallInfo         - 関数呼び出し情報のアドレス  
           value             - 追加する引数値（文字列）へのポインタ

**説 明** : 関数呼び出し情報に引数（文字列）を追加します。

**戻り値** : TRUE - 成功  
           FALSE - 失敗

**5.9.17. 関数の戻り値取得（整数）（AjcCipGetRetInt）**

- 形 式** : C\_ULLP AjcCipGetRetInt (HWND hwnd, UIP pNum);
- 引 数** : hwnd                    - コントロールのウインドハンドル  
pNum                        - 配列要素数を格納するバッファのアドレス（不要時は NULL）
- 説 明** : AjcCipExec () で実行した数値型関数の戻り情報を取得します、  
呼び出した関数が、配列を返す場合、\*pNum に配列の要素数が設定されます。
- 戻り値** : ≠NULL : 64 ビット整数値・数値配列の先頭アドレス  
=NULL : エラー

**5.9.18. 関数の戻り値取得（実数）（AjcCipGetRetReal）**

- 形 式** : const double \*AjcCipGetRetReal (HWND hwnd, UIP pNum);
- 引 数** : hwnd                    - コントロールのウインドハンドル  
pNum                        - 配列要素数を格納するバッファのアドレス（不要時は NULL）
- 説 明** : AjcCipExec () で実行した数値型関数の戻り情報を取得します、  
呼び出した関数が、配列を返す場合、\*pNum に配列の要素数が設定されます。
- 戻り値** : ≠NULL : double 型・数値配列の先頭アドレス  
=NULL : エラー

**5.9.19. 関数の戻り値取得（文字列）（AjcCipGetRetStr）**

- 形 式** : const double \*AjcCipGetRetStr (HWND hwnd, UIP pNum);
- 引 数** : hwnd                    - コントロールのウインドハンドル  
pNum                        - 配列要素数を格納するバッファのアドレス（不要時は NULL）
- 説 明** : AjcCipExec () で実行した文字型関数の戻り情報を取得します。  
呼び出した関数が、配列を返す場合、\*pNum に配列の要素数が設定されます。
- 戻り値** : ≠NULL : 文字列へのポインタ配列の先頭アドレス  
=NULL : エラー

**5.9.20. トレース表示 ウインド生成（AjcCipCreateTraceWindow）**

- 形 式** : AjcCipCreateTraceWindow (HWND hwnd);
- 引 数** : hwnd                    - コントロールのウインドハンドル
- 説 明** : トレース表示ウインドを生成します。  
このウインドには、\_Trace () 関数による表示内容が出力されます。
- 戻り値** : TRUE - 成功  
FALSE - 失敗

**5.9.21. トレース表示 ウインド破棄 (AjcCipDestroyTraceWindow)**

**形 式** : AjcCipDestroyTraceWindow (HWND hwnd);

**引 数** : hwnd                      - コントロールのウインドハンドル

**説 明** : トレース表示ウインドを閉じて、破棄します。

**戻り値** : TRUE - 成功  
FALSE - 失敗

**5.9.22. ステップトレース表示 ウインド生成 (AjcCipCreateStepWindow)**

**形 式** : AjcCipCreateStepWindow (HWND hwnd);

**引 数** : hwnd                      - コントロールのウインドハンドル

**説 明** : ステップトレース表示ウインドを生成します。  
このウインドには、ステップ毎の実行内容がトレース出力されます。

**戻り値** : TRUE - 成功  
FALSE - 失敗

**5.9.23. ステップトレース表示 ウインド破棄 (AjcCipDestroyStepWindow)**

**形 式** : AjcCipDestroyStepWindow (HWND hwnd);

**引 数** : hwnd                      - コントロールのウインドハンドル

**説 明** : ステップトレース表示ウインドを生成します。  
マクロ実行中の場合は、ステップトレースウインドを破棄できません。

**戻り値** : TRUE - 成功  
FALSE - 失敗

**5.9.24. ステップ・トレースの禁止／許可 (AjcCipEnableStepTrace)**

**形 式** : AjcCipEnableStepTrace (HWND hwnd, BOOL fEnable);

**引 数** : hwnd                      - コントロールのウインドハンドル  
fEnable                      - ステップトレース表示の禁止／許可フラグ (TRUE:許可, FALSE:禁止)

**説 明** : ステップトレースの表示を禁止／許可します。

**戻り値** : TRUE - 成功  
FALSE - 失敗



**5.9.25. VT100 エミュレーションウインドのハンドル取得(AjcCipGet VT100Handle)**

**形 式** : HWND AjcCipGetVT00Handle(HWND hwnd) ;

**引 数** : hwnd - コントロールのウインドハンドル

**説 明** : VT100 エミュレーションウインド (本コントロールの子ウインド) のハンドルを取得します。

**戻り値** : ≠NULL : VT100 エミュレーションウインドのハンドル  
=NULL : エラー

**5.9.26. ステータス情報取得(AjcCipGetStatus)**

**形 式** : BOOL AjcCipGetStatus (HWND hwnd, PAJCCIPSTATE pState) ;

**引 数** : hwnd - コントロールのウインドハンドル  
pState - ステータス情報を格納するバッファのアドレス

**説 明** : マクロテキストの実行状態等のステータスを取得します。  
pState で示すバッファに以下の情報が設定されます。

```
typedef struct {
    AJCCIPSTSTYPE    eType; // 数値タイプ
    AJCCIPSTSEXEC    eExec; // 実行状態
} AJCCIPSTATE, *PAJCCIPSTATE;
```

各メンバの値は以下の通りです。

eType (数値タイプ)	
AJCCIPSTS_UNKOWN	未設定
AJCCIPSTS_SINT	符号付き整数(64 bit)
AJCCIPSTS_UINT	符号なし整数(64 bit)
AJCCIPSTS_REAL	実数(double)

eExec (実行状態)	
AJCCIPSTS_NOTREADY	実行不能状態
AJCCIPSTS_READY	実行可能状態
AJCCIPSTS_EXEC	実行中

**戻り値** : ≠NULL : VT100 エミュレーションウインドのハンドル  
=NULL : エラー

## 5.10. 通知情報の取得 A P I

コントロールからの通知メッセージ (WM\_COMMAND) に伴うパラメータを取得する A P I 群です。

「AjcSetCmdWithHdl(TRUE);」を実行した場合、各通知メッセージ (WM\_COMMAND) の lParam は通知内容に伴うパラメータは通知されず、lParam=コントロールのウインドハンドルとなります。

この場合、通知内容に伴うパラメータは以下の A P I で取得します。

#	関 数 名	内 容
1	AjcCipGetNtcEvpFun	外部関数呼び出し情報の取得
2	AjcCipGetNtcText	通知テキスト (エラーメッセージ, ダブルクリック行) の取得
3	AjcCipGetNtcFiles	ドロップしたファイル数の取得
4	AjcCipGetNtcDirs	ドロップしたディレクトリ数の取得

## 5.10.1. 外部関数呼び出し情報の取得 (AjcCipGetNtcEvpFun)

**形 式** : PAJCCIPVFPUN AjcCipGetNtcEvpFun (HHWND hwnd);

**引 数** : hwnd                      - コントロールのウインドハンドル

**説 明** : 外部関数呼び出し通知 (AJCCIPN\_FUN) 時の、外部関数呼び出し情報を取得します。

**戻り値** : 外部関数呼び出し情報 (AJCCIPVFPUN) へのポインタ

## 5.10.2. 通知テキストの取得 (AjcCipGetNtcText)

**形 式** : BCP AjcCipGetNtcText (HHWND hwnd);

**引 数** : hwnd                      - コントロールのウインドハンドル

**説 明** : エラー発生通知 (AJCCIPN\_ERR) 時のエラーメッセージテキスト、あるいは、ダブルクリック通知 (AJCCIPN\_DBLCLK) 時のダブルクリックした行テキストを取得します。

**戻り値** : テキスト (文字列) へのポインタ

## 5.10.3. ドロップしたディレクトリ数の取得 (AjcCipGetNtcDirs)

**形 式** : UI AjcCipGetNtcDirs (HHWND hwnd);

**引 数** : hwnd                      - コントロールのウインドハンドル

**説 明** : ディレクトリドロップ通知 (AJCCIPN\_DROPDIR) 時の、ドロップされたディレクトリの個数を取得します。

**戻り値** : ドロップされたディレクトリの個数

## 5.10.4. ドロップしたファイル数の取得 (AjcCipGetNtcFiles)

**形 式** : UI AjcCipGetNtcFiles (HHWND hwnd);

**引 数** : hwnd                      - コントロールのウインドハンドル

**説 明** : ファイルドロップ通知 (AJCCIPN\_DROPFILE) 時の、ドロップされたファイルの個数を取得します。

**戻り値** : ドロップされたファイルの個数

## 5.11. 通知メッセージ

通知メッセージは、WM\_COMMAND メッセージにより通知されます。  
WM\_COMMAND メッセージの wParam には以下の情報が設定されます。

- ・ LOWORD(wParam) - コントロールの識別 I D
- ・ HIWORD(wParam) - 通知メッセージコード

通知メッセージコードは、以下のとおりです。

#	通知メッセージコード	内容	備考
1	AJCCIPN_FUN	外部関数呼び出し通知	
2	AJCCIPN_EXE	ステップ実行通知	
3	AJCCIPN_ERR	エラー発生通知	
4	AJCCIPN_DBLCLK	ダブルクリック通知	
5	AJCCIPN_DROPFILE	ファイルドロップ通知	
6	AJCCIPN_DROPDIR	ディレクトリドロップ通知	

### 5.11.1. 外部関数呼び出し通知 (AJCCIPN\_FUN)

**説明** : 外部関数（内臓関数には存在しない関数）の呼び出しを通知します。  
lParam には、外部関数呼び出しに関する以下の構造体へのポインタが設定されます。

```
typedef struct {
    UI      argc;           // 引数の個数
    PCAJCCIPAGV *argv;      // 引数情報
    PCAJCCIPCALL pCall;     // 呼び出し情報
} AJCCIPEVPFUN, *PAJCCIPEVPFUN;
```

**パラメタ** : wParam - コントロール識別 I D と、通知メッセージコード (AJCCIPN\_FUN)  
lParam - 外部関数呼び出し情報のアドレス (PAJCCIPEVPFUN)

**戻り値** : なし (0 を返してください)

### 5.11.2. ステップ実行通知 (AJCCIPN\_EXE)

**説明** : このイベントは 1 ステップ実行毎に発生します。  
lParam には、実行命令に関する以下の構造体へのポインタが設定されます。

```
typedef struct {
    PCAJCCIPTKN pTkn;       // トークン情報
    BOOL        fStop;      // 実行中止フラグ
} AJCCIPEVPEXE, *PAJCCIPEVPEXE;
```

この構造体のメンバ「fStop」に TRUE を設定すると、マクロテキストの実行を中止します。

**パラメタ** : wParam - コントロール識別 I D と、通知メッセージコード (AJCCIPN\_EXE)  
lParam - 実行命令に関する情報のアドレス (PAJCCIPEVPEXE)

**戻り値** : なし (0 を返してください)

### 5.11.3. エラー通知 (AJCCIPN\_ERR)

**説明** : マクロテキストの読み出し中、あるいは、実行中にエラーが発生したことを通知します。  
lParam には、エラー情報に関する以下の構造体へのポインタが設定されます。

```
typedef struct {
    AJCCIP_ERR  err;           // エラーコード
    UI          lno;           // 行番号
    C_BCP       pMsg;          // エラーメッセージテキストへのポインタ
} AJCCIPVPEPERR, *PAJCCIPVPEPERR;
```

エラーコード(err)の値は以下の通りです。

名称	内容	名称	内容
AJCIP_ERR_NOLOAD	マクロファイル未読み出し	AJCIP_ERR_PARAM	パラメタエラー
AJCIP_ERR_NOTREADY	マクロ実行準備ができていない	AJCIP_ERR_CALC	演算エラー
AJCIP_ERR_MEM	メモリエラー	AJCIP_ERR_EXEC	実行時エラー
AJCIP_ERR_VMG	変数アクセスエラー	AJCIP_ERR_USER	ユーザからのエラー通知
AJCIP_ERR_SYNTAX	シンタックスエラー	AJCIP_ERR_INTERNAL	内部エラー

**パラメタ** : wParam - コントロール識別 ID と、通知メッセージコード (AJCCIPN\_ERR)  
lParam - エラー情報のアドレス (PAJCCIPVPEPERR)

**戻り値** : なし

### 5.11.4. ダブルクリック通知 (AJCCIPN\_DBLCLK)

**説明** : コントロールウインド上でダブルクリックしたことを通知します。  
このイベントは、AjcCipSetDb1ClkAct() で、fNtcDb1Clk=TRUE を指定した場合に発生します。  
lParam には、ダブルクリックした行テキストへのポインタが設定されます。

**パラメタ** : wParam - コントロールの識別 ID と通知メッセージコード (AJCCIPN\_DBLCLK)  
lParam - ダブルクリックした、行テキストへのポインタ

**戻り値** : なし (ゼロを返してください)

### 5.11.5. ファイルドロップ通知 (AJCCIPN\_DROPFILE)

**説明** : マクロコードからの表示出力用ウインドへファイルがドロップされたことを通知します。  
本通知では、ドロップされたファイルの個数だけを通知します。  
ドロップされたファイルのパス名は、AjcCipGetDroppedFile() により取得します。

**パラメタ** : wParam - コントロールの識別 ID と通知メッセージコード (AJCCIPN\_DROPFILE)  
lParam - ドロップされたファイルの個数

**戻り値** : なし (ゼロを返してください)

### 5.11.6. ディレクトリドロップ通知 (AJCCIPN\_DROPDIR)

**説明** : マクロコードからの表示出力用ウインドへディレクトリがドロップされたことを通知します。  
本通知では、ドロップされたディレクトリの個数だけを通知します。  
ドロップされたディレクトリのパス名は、AjcCipGetDroppedDir[Ex]() により取得します。

**パラメタ** : wParam - コントロールの識別 ID と通知メッセージコード (AJCCIPN\_DROPDIR)  
lParam - ドロップされたディレクトリの個数

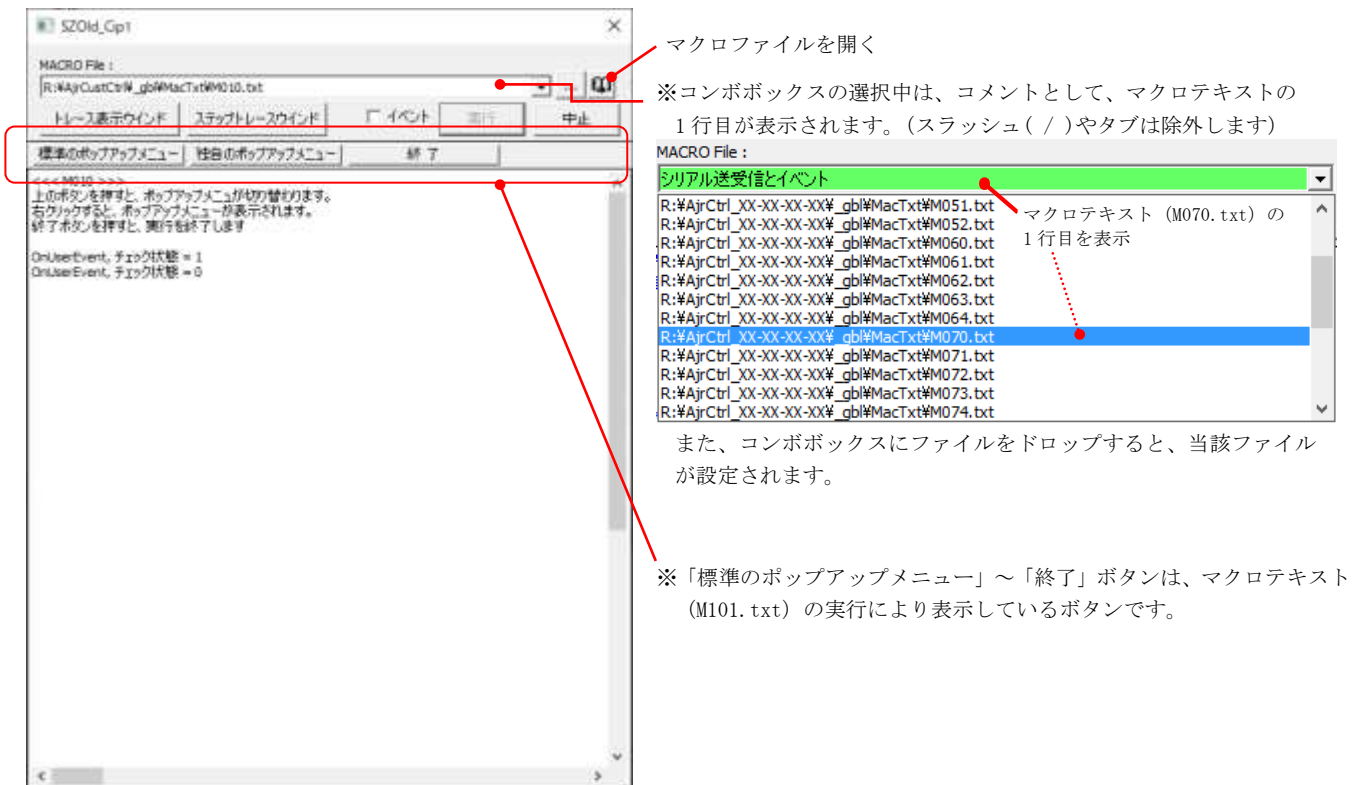
**戻り値** : なし (ゼロを返してください)

## 5.12. サンプルプログラム

## 5.12.1. サンプルプログラム 1

以下のサンプルプログラムは、指定したマクロテキスト・ファイルを実行します。  
マクロテキスト中の、main() 関数 (引数無し) を実行します。

「トレース表示ウインド」ボタンを押すと、トレースの表示用ウインドが表示されます。  
「ステップトレースウインド」ボタンを押すと、ステップトレース・ウインドが表示されます。  
マクロテキスト・ファイルをコンボボックスか、「[...]」ボタンで選択し、「実行」ボタンを押すとマクロテキストを実行します。  
選択したマクロテキストファイルは、コンボボックスに記録されます。  
「中止」ボタンを押すと、マクロテキストの実行を中止します。  
「イベント」チェックボックスは、クリックするとマクロテキスト内のユーザイベント関数「OnUserEvent()」を実行します。



```

1 : //
2 : //  SZ01d_Cip1.c
3 : //
4 :
5 : #pragma warning(disable:4996)
6 : #include  <AjrcstXX.h>
7 : #include  <math.h>
8 : #include  <tchar.h>
9 : #include  "resource.h"
10 :
11 : //-----//
12 : // ワーク
13 : //-----//
14 : static  HINSTANCE  hInst;           // D L L インスタンスハンドル
15 : static  HWND       hDlgMain;       // ダイアログボックスハンドル
16 : static  HWND       hWndCip;        // C I P ウインドハンドル
17 : static  HICON      hIcoOpen;
18 : static  BC         InpPath[MAX_PATH]; // マクロファイルパス
19 : static  BOOL        ixLast;         // コンボボックス直前の選択項目
20 : static  int         DlgWidth;       // メインダイアログサイズ
21 : static  int         DlgHeight;
22 : static  int         yCip;          // C I P コントロールのY位置
23 :
24 : static  BOOL        fRun  = FALSE;  // マクロ実行中を示すフラグ
25 : static  BOOL        fExit = FALSE;  // プログラム終了を示すフラグ
26 :
27 : //-----//
28 : //  内部サブ関数
29 : //-----//

```

```

30 : AJC_DLGPROC_DEF(Main);
31 : static VO SubSetDefaultFile(HWND hDlg, C_BCP pFileName);
32 :
33 : //=====//
34 : //
35 : // WinMain
36 : //
37 : //=====//
38 : int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int iCmdShow)
39 : {
40 :     MSG     msg;
41 :
42 :     hInst = hInstance;
43 :
44 :     //----- プロファイル先をレジストリとする -----//
45 :     AjcSetProfileIsRegistry(TRUE);
46 :     AjcSetRegOptionVolatile (FALSE);
47 :     //----- メイン・ダイアログオープン -----//
48 :     hDlgMain = CreateDialog(hInst, MAKEINTRESOURCE(IDD_DLGMAIN), NULL, AJC_DLGPROC_NAME(Main));
49 :     //----- ダイアログ表示 -----//
50 :     ShowWindow(hDlgMain, SW_SHOW);
51 :     //----- メッセージループ -----//
52 :     while (GetMessage(&msg, NULL, 0, 0)) {
53 :         do {
54 :             if (IsDialogMessage(hDlgMain, &msg)) break;
55 :             TranslateMessage(&msg);
56 :             DispatchMessage (&msg);
57 :         } while (0);
58 :     }
59 :     return (int)msg.wParam ;
60 : }
61 : //=====//
62 : //
63 : // ダイアログ・プロシージャ
64 : //
65 : //=====//
66 : //----- ダイアログ初期化 -----//
67 : AJC_DLGPROC(Main, WM_INITDIALOG      )
68 : {
69 :     int     ix;
70 :     RECT     r;
71 :
72 :     hDlgMain = hDlg;
73 :     hWndCip = GetDlgItem(hDlg, IDC_CIP);
74 :
75 :     // 最小ウインドサイズ設定
76 :     GetWindowRect(hDlg, &r);
77 :     DlgWidth  = r.right - r.left;
78 :     DlgHeight = r.bottom - r.top;
79 :
80 :     // C I P コントロールの Y 位置設定
81 :     GetWindowRect(hWndCip, &r);
82 :     ScreenToClient(hDlg, (LPPPOINT)&r);
83 :     yCip = r.top;
84 :
85 :     // C I P コントロール配置
86 :     GetClientRect(hDlg, &r);
87 :     SetWindowPos(hWndCip, NULL, 0, yCip, r.right - r.left, (r.bottom - r.top) - yCip, SWP_NOZORDER);
88 :
89 :     // アイコン生成
90 :     hIcoOpen = (HICON)LoadImage(hInst, MAKEINTRESOURCE(IDI_OPEN), IMAGE_ICON, 16, 16, LR_DEFAULTCOLOR);
91 :     // ボタンへアイコン表示
92 :     SendDlgItemMessage(hDlg, IDC_CMD_OPEN, BM_SETIMAGE, IMAGE_ICON, (LPARAM)hIcoOpen);
93 :
94 :     // コンボボックス初期化
95 :     AjcSbcComboBox (GetDlgItem(hDlg, IDC_CBO_FILE), 100, MAX_PATH);
96 :     AjcSbcLoadItems (GetDlgItem(hDlg, IDC_CBO_FILE), "S_SbcComboBox");
97 :     AjcSbcSetCompExact (GetDlgItem(hDlg, IDC_CBO_FILE), FALSE); // 大小区別しないで比較
98 :     AjcSbcShowFirstLine(GetDlgItem(hDlg, IDC_CBO_FILE), "/*¥t¥r¥n");
99 :     AjcSbcTipCtrl (GetDlgItem(hDlg, IDC_CBO_FILE), TRUE);
100 :    // チップテキストコントロール生成
101 :    AjcTipTextCreate();
102 :
103 :    // デフォルトのマクロファイル名設定
104 :    if (AjcGetDlgItemCboCount(hDlg, IDC_CBO_FILE) == 0) {
105 :        SubSetDefaultFile(hDlg, "M010. txt"); SubSetDefaultFile(hDlg, "M075. txt");
106 :        SubSetDefaultFile(hDlg, "M020. txt"); SubSetDefaultFile(hDlg, "M076. txt");
107 :        SubSetDefaultFile(hDlg, "M021. txt"); SubSetDefaultFile(hDlg, "M077. txt");
108 :        SubSetDefaultFile(hDlg, "M030. txt"); SubSetDefaultFile(hDlg, "M078. txt");
109 :        SubSetDefaultFile(hDlg, "M040. txt"); SubSetDefaultFile(hDlg, "M079. txt");
110 :        SubSetDefaultFile(hDlg, "M050. txt"); SubSetDefaultFile(hDlg, "M080. txt");
111 :        SubSetDefaultFile(hDlg, "M051. txt"); SubSetDefaultFile(hDlg, "M090. txt");
112 :        SubSetDefaultFile(hDlg, "M052. txt"); SubSetDefaultFile(hDlg, "M100. txt");
113 :        SubSetDefaultFile(hDlg, "M060. txt"); SubSetDefaultFile(hDlg, "M101. txt");
114 :        SubSetDefaultFile(hDlg, "M061. txt"); SubSetDefaultFile(hDlg, "M110. txt");
115 :        SubSetDefaultFile(hDlg, "M062. txt"); SubSetDefaultFile(hDlg, "M120. txt");
116 :        SubSetDefaultFile(hDlg, "M063. txt"); SubSetDefaultFile(hDlg, "M130. txt");
117 :        SubSetDefaultFile(hDlg, "M064. txt"); SubSetDefaultFile(hDlg, "M140. txt");
118 :        SubSetDefaultFile(hDlg, "M070. txt"); SubSetDefaultFile(hDlg, "M150. txt");
119 :        SubSetDefaultFile(hDlg, "M071. txt"); SubSetDefaultFile(hDlg, "M160. txt");

```

```

120 :         SubSetDefaultFile(hDlg, "M072.txt");
121 :         SubSetDefaultFile(hDlg, "M073.txt");
122 :         SubSetDefaultFile(hDlg, "M074.txt");
123 :     }
124 :     // コンボボックスからマクロファイル名取得
125 :     ix = AjbGetDlgItemCboIx(hDlg, IDC_CBO_FILE);
126 :     AjbGetDlgItemCboItem(hDlg, IDC_CBO_FILE, ix, InpPath, MAX_PATH);
127 :
128 :     return TRUE;
129 : }
130 : //----- ウインド破棄 -----//
131 : AJC_DLGPROC(Main, WM_DESTROY
132 : {
133 :     // コンボボックス設定内容退避
134 :     AjbSbcSaveItems(GetDlgItem(hDlg, IDC_CBO_FILE), "S_SbcComboBox");
135 :     // アイコン破棄
136 :     DeleteObject(hIcoOpen);
137 :     // チップテキストコントロール破棄
138 :     AjbTipTextDelete();
139 :     // プログラム終了
140 :     PostQuitMessage(0);
141 :     return TRUE;
142 : }
143 : //----- サイズ変更 -----//
144 : AJC_DLGPROC(Main, WM_SIZING
145 : {
146 :     LPRECT p = (LPRECT)lParam;
147 :     int w = p->right - p->left;
148 :     int h = p->bottom - p->top;
149 :
150 :     if (w < DlgWidth) p->right = p->left + DlgWidth;
151 :     if (h < DlgHeight) p->bottom = p->top + DlgHeight;
152 :
153 :     return TRUE;
154 : }
155 : //----- サイズ変更 -----//
156 : AJC_DLGPROC(Main, WM_SIZE
157 : {
158 :     int w = LOWORD(lParam);
159 :     int h = HIWORD(lParam);
160 :
161 :     SetWindowPos(hWndCip, NULL, 0, yCip, w, h - yCip, SWP_NOZORDER);
162 :     return TRUE;
163 : }
164 : //----- ファイル選択ボタン(...) -----//
165 : AJC_DLGPROC(Main, IDC_CMD_FILE
166 : {
167 :     static BC path[MAX_PATH] = {0};
168 :
169 :     if (HIWORD(wParam) == BN_CLICKED) {
170 :         if (AjbGetOpenFile(hDlgMain, // オーナーウインドハンドル
171 :             "Get open file", // ウインドタイトル
172 :             "txt/*.txt/all/*.*", // フィルタ
173 :             "txt", // デフォルト拡張子
174 :             path, // パス名格納バッファ
175 :             MAX_PATH)) { // バッファの文字数
176 :             SendDlgItemMessage(hDlg, IDC_CBO_FILE, CB_INSERTSTRING, 0, (LPARAM)path);
177 :             strcpy(InpPath, path);
178 :         }
179 :     }
180 :     return TRUE;
181 : }
182 : //----- ファイルを開くボタン -----//
183 : AJC_DLGPROC(Main, IDC_CMD_OPEN
184 : {
185 :     int ix = AjbGetDlgItemCboIx(hDlg, IDC_CBO_FILE);
186 :     UT path[MAX_PATH];
187 :     if (ix >= 0) {
188 :         AjbGetDlgItemCboItem(hDlg, IDC_CBO_FILE, ix, path, MAX_PATH);
189 :         ShellExecute(NULL, TEXT("open"), path, NULL, NULL, SW_SHOWNORMAL);
190 :     }
191 :     return TRUE;
192 : }
193 : //----- トレース表示ウインド ボタン -----//
194 : AJC_DLGPROC(Main, IDC_CMD_TRC
195 : {
196 :     if (HIWORD(wParam) == BN_CLICKED) {
197 :         AjbCipCreateTraceWindow(hWndCip);
198 :     }
199 :     return TRUE;
200 : }
201 : //----- ステップトレース・ウインド ボタン -----//
202 : AJC_DLGPROC(Main, IDC_CMD_STEP
203 : {
204 :     if (HIWORD(wParam) == BN_CLICKED) {
205 :         AjbCipCreateStepWindow(hWndCip);
206 :     }
207 :     return TRUE;
208 : }
209 : //----- 実行ボタン -----//

```

```

210 : AJC_DLGPROC(Main, IDC_CMD_RUN      )
211 : {
212 :     C_BCP    pTxt;
213 :
214 :     if (HIWORD(wParam) == BN_CLICKED) {
215 :         int ix = AjcGetDlgItemCboIx(hDlg, IDC_CBO_FILE);
216 :         if (ix >= 0) {
217 :             AjcGetDlgItemCboItem(hDlg, IDC_CBO_FILE, ix, InpPath, sizeof InpPath);
218 :             if (AjcCipLoad(hWndCip, InpPath)) {
219 :                 PAJCCIPCALL pCall;
220 :                 // 関数呼び出し情報設定
221 :                 if (pCall = AjcCipCallInfoInit(hWndCip, "main")) { // 呼び出し情報生成
222 :                     EnableWindow(GetDlgItem(hDlg, IDC_CMD_RUN ), FALSE);
223 :                     EnableWindow(GetDlgItem(hDlg, IDC_CMD_STOP ), TRUE);
224 :                     EnableWindow(GetDlgItem(hDlg, IDC_CHK_EVENT), TRUE);
225 :                     // 実行
226 :                     fRun = TRUE;
227 :                     if (AjcCipExec(hWndCip, pCall)) {
228 :                         AjcCipPrintF(hWndCip, "¥n");
229 :                         // _Exit() によるテキスト表示
230 :                         if (pTxt = AjcCipGetExitText(hWndCip)) {
231 :                             AjcCipPrintF(hWndCip, "_Exit() により設定されたテキスト='¥s'¥n", pTxt);
232 :                         }
233 :                         AjcCipPrintF(hWndCip, "マクロの実行を終了しました。¥n");
234 :                     }
235 :                     else {
236 :                         AjcCipPrintF(hWndCip, "マクロの実行中にエラーが発生しました。¥n");
237 :                     }
238 :                     AjcCipCallInfoDelete(hWndCip, pCall);
239 :                     fRun = FALSE;
240 :                     EnableWindow(GetDlgItem(hDlg, IDC_CMD_RUN ), TRUE);
241 :                     EnableWindow(GetDlgItem(hDlg, IDC_CMD_STOP ), FALSE);
242 :                     EnableWindow(GetDlgItem(hDlg, IDC_CHK_EVENT), FALSE);
243 :                     // プログラム終了判断
244 :                     if (fExit) {
245 :                         DestroyWindow(hDlg);
246 :                     }
247 :                 }
248 :                 else {
249 :                     AjcCipPrintF(hWndCip, "マクロテキストに main() 関数がありません。¥n");
250 :                 }
251 :             }
252 :             else {
253 :                 AjcCipPrintF(hWndCip, "マクロの読み出しを失敗しました。¥n");
254 :             }
255 :         }
256 :     }
257 :     return TRUE;
258 : }
259 : //----- 中止ボタン -----//
260 : AJC_DLGPROC(Main, IDC_CMD_STOP      )
261 : {
262 :     if (HIWORD(wParam) == BN_CLICKED) {
263 :         AjcCipStop(hWndCip);
264 :     }
265 :     return TRUE;
266 : }
267 : //----- マクロへのイベント通知 (イベント・チェックボックス) -----//
268 : AJC_DLGPROC(Main, IDC_CHK_EVENT      )
269 : {
270 :     if (HIWORD(wParam) == BN_CLICKED) {
271 :         PAJCCIPCALL pCall;
272 :         // 関数呼び出し情報設定
273 :         if (pCall = AjcCipCallInfoInit(hWndCip, "OnUserEvent")) {
274 :             // 第1引数(チェック状態)
275 :             AjcCipCallInfoInsInteger(hWndCip, pCall, AjcGetDlgItemChk(hDlg, IDC_CHK_EVENT));
276 :             // イベント実行
277 :             AjcCipEvent(hWndCip, pCall);
278 :         }
279 :     }
280 :     return TRUE;
281 : }
282 : //----- マクロインタプリタからの通知 -----//
283 : AJC_DLGPROC(Main, IDC_CIP      )
284 : {
285 :     switch (HIWORD(wParam)) {
286 :         case AJCCIPN_FUN: // ●外部関数呼び出し通知
287 :             {
288 :                 PAJCCIPVFPFUN pCbp = (PAJCCIPVFPFUN)lParam;
289 :                 AjcCipNtcErrByUser(hWndCip, pCbp->pCall, AJCLNGSEL("関数 <¥s> は未定義です", "Undefined function <¥s>."),
290 :                                     pCbp->pCall->pFunName);
291 :                 break;
292 :             }
293 :         case AJCCIPN_EXE: // ●ステップ実行通知
294 :             {
295 :                 break;
296 :             }
297 :         case AJCCIPN_ERR: // ●エラー発生通知
298 :             {
299 :                 PAJCCIPVFPERR pCbp = (PAJCCIPVFPERR)lParam;

```



```

300 :         AjcCipPrintF(hWndCip, "err = %d, lno=%d¥n  %s¥n", pCbp->err, pCbp->lno, pCbp->pMsg);
301 :         break;
302 :     }
303 : }
304 :     return TRUE;
305 : }
306 : //----- キャンセル -----//
307 : AJC_DLGPROC(Main, IDCANCEL          )
308 : {
309 :     if (fRun) {
310 :         AjcCipStop(hWndCip);
311 :         fExit = TRUE;
312 :     }
313 :     else {
314 :         DestroyWindow(hDlg);
315 :     }
316 :     return TRUE;
317 : }
318 : //-----//
319 : AJC_DLGMAP_DEF(Main)
320 :     AJC_DLGMAP_MSG(Main, WM_INITDIALOG      )
321 :     AJC_DLGMAP_MSG(Main, WM_DESTROY        )
322 :     AJC_DLGMAP_MSG(Main, WM_SIZING         )
323 :     AJC_DLGMAP_MSG(Main, WM_SIZE          )
324 :
325 :     AJC_DLGMAP_CMD(Main, IDC_CMD_FILE      )
326 :     AJC_DLGMAP_CMD(Main, IDC_CMD_OPEN     )
327 :     AJC_DLGMAP_CMD(Main, IDC_CMD_TRC      )
328 :     AJC_DLGMAP_CMD(Main, IDC_CMD_STEP     )
329 :     AJC_DLGMAP_CMD(Main, IDC_CMD_RUN      )
330 :     AJC_DLGMAP_CMD(Main, IDC_CMD_STOP     )
331 :     AJC_DLGMAP_CMD(Main, IDC_CHK_EVENT    )
332 :     AJC_DLGMAP_CMD(Main, IDC_CIP         )
333 :     AJC_DLGMAP_CMD(Main, IDCANCEL        )
334 : AJC_DLGMAP_END
335 :
336 : //-----//
337 : // コンボボックスにデフォルトマクロファイル設定 //
338 : //-----//
339 : static VOID SubSetDefaultFile(HWND hDlg, C_BCP pFileName)
340 : {
341 :     BC      path[MAX_PATH];
342 :     BCP     pFile;
343 :
344 :     AjcGetAppPath(path, MAX_PATH);
345 :     strcat(path, "..¥¥MacTxt¥¥");
346 :     strcat(path, pFileName);
347 :     GetFullPathName(path, sizeof InpPath, InpPath, &pFile);
348 :     AjcSetDlgItemCboAdd(hDlg, IDC_CBO_FILE, -1, InpPath, AJCCBF_ALL);
349 : }

```

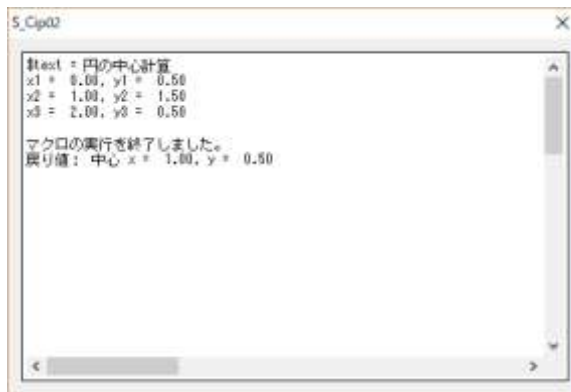
### 5.12.2. サンプルプログラム 2

プログラムから、マクロテキスト内の引数付き関数をコールするサンプルプログラムです。

このサンプルプログラムは、マクロテキスト (M990.txt) の関数「func」を実行します。

S\_Cip02.txt の関数「func」は、1つのテキスト(\$text)と3つの座標値(x1, y1, x2, y2, x3, y3)を受け取り、この3つの点を通る円の中心座標値を、要素数=2の配列で返します。

#### 実行画面



#### マクロテキスト(M990.txt)

```
option double;

//-----//
// 3点から円の中心を求める //
// 戻り値 : 円の中心座標 ([0]=x, [1]=y) //
//-----//
function func($text, x1, y1, x2, y2, x3, y3)
{
    _Printf("$text = %s\n", $text);

    _Printf("x1 = %5.2f, y1 = %5.2f\n", x1, y1);
    _Printf("x2 = %5.2f, y2 = %5.2f\n", x2, y2);
    _Printf("x3 = %5.2f, y3 = %5.2f\n", x3, y3);

    define rc[2];

    rc[0] = ((y1 - y3) * (M(y1) - M(y2) + M(x1) - M(x2)) -
              (y1 - y2) * (M(y1) - M(y3) + M(x1) - M(x3))) /
              (2.0 * (y1 - y3) * (x1 - x2) - 2.0 * (y1 - y2) * (x1 - x3));

    rc[1] = ((x1 - x3) * (M(x1) - M(x2) + M(y1) - M(y2)) -
              (x1 - x2) * (M(x1) - M(x3) + M(y1) - M(y3))) /
              (2.0 * (x1 - x3) * (y1 - y2) - 2.0 * (x1 - x2) * (y1 - y3));

    return rc[];
}
//-----//
// nの二乗値を返す //
//-----//
function M(n)
{
    return n * n;
}
```

```

1 : //
2 : //  S_Cip02.c
3 : //
4 :
5 : #pragma warning(disable:4996)
6 : #include <AjrCtlXX.h>
7 : #include "resource.h"
8 :
9 : //-----//
10 : //   ワーク                                     //
11 : //-----//
12 : static HINSTANCE      hInst;                // D L Lインスタンスハンドル
13 : static HWND           hDlgMain;             // ダイアログボックスハンドル
14 : static HWND           hWndCip;              // C I Pウインドハンドル
15 : static BOOL           fRun  = FALSE;        // マクロ実行中を示すフラグ
16 : static BOOL           fExit = FALSE;        // プログラム終了を示すフラグ
17 :
18 : //-----//
19 : //   内部サブ関数                               //
20 : //-----//
21 : AJC_DLGPROC_DEF(Main);
22 :
23 : //=====//
24 : //                                     //
25 : //   W i n M a i n                               //
26 : //                                     //
27 : //=====//
28 : int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR szCmdLine, int iCmdShow)
29 : {
30 :     MSG      msg;
31 :
32 :     hInst = hInstance;
33 :
34 :     //----- プロファイル先をレジストリとする -----//
35 :     AjcSetProfileIsRegistry(TRUE);
36 :     AjcSetRegOptionVolatile (FALSE);
37 :     //----- メイン・ダイアログオープン -----//
38 :     hDlgMain = CreateDialog(hInst, MAKEINTRESOURCE(IDD_MAIN), NULL, AJC_DLGPROC_NAME(Main));
39 :     //----- ダイアログ表示 -----//
40 :     ShowWindow(hDlgMain, SW_SHOW);
41 :     //----- メッセージループ -----//
42 :     while (GetMessage(&msg, NULL, 0, 0)) {
43 :         do {
44 :             if (IsDialogMessage(hDlgMain, &msg)) break;
45 :             TranslateMessage(&msg);
46 :             DispatchMessage (&msg);
47 :         } while (0);
48 :     }
49 :     return (int)msg.wParam ;
50 : }
51 : //=====//
52 : //                                     //
53 : //   ダイアログ・プロシージャ                               //
54 : //                                     //
55 : //=====//
56 : //----- ダイアログ初期化 -----//
57 : AJC_DLGPROC(Main, WM_INITDIALOG      )
58 : {
59 :     hDlgMain = hDlg;
60 :     hWndCip  = GetDlgItem(hDlg, IDC_CIP);
61 :
62 :     // タイマを設定し、1秒後にマクロ実行
63 :     SetTimer(hDlg, 1, 1000, NULL);
64 :
65 :     return TRUE;
66 : }
67 : //----- ウインド破棄 -----//
68 : AJC_DLGPROC(Main, WM_DESTROY      )
69 : {
70 :     // プログラム終了
71 :     PostQuitMessage(0);
72 :
73 :     return TRUE;
74 : }

```

```

75 : //----- WM_TIMER -----//
76 : AJC_DLGPROC(Main, WM_TIMER          )
77 : {
78 :     const double * pResult;
79 :     UI              n;
80 :     BC  InpFile[MAX_PATH];
81 :
82 :     // タイマ停止
83 :     KillTimer(hDlg, 1);
84 :
85 :     // マクロテキストパス設定
86 :     AjcGetAppPath(InpFile, MAX_PATH);
87 :     AjcPathCat(InpFile, "..¥¥MacTxt¥¥M990.txt", MAX_PATH);
88 :
89 :     // マクロの読み出しと実行
90 :     if (AjcCipLoad(hWndCip, InpFile)) {
91 :         PAJCCIPCALL pCall;
92 :         if (pCall = AjcCipCallInfoInit(hWndCip, "func")) {
93 :             fRun = TRUE;
94 :             // 引数設定
95 :             AjcCipCallInfoInsString(hWndCip, pCall, "円の中心計算"); // 第1引数(タイトル)
96 :             AjcCipCallInfoInsReal (hWndCip, pCall, 0.0);           // 第2引数(x1)
97 :             AjcCipCallInfoInsReal (hWndCip, pCall, 0.5);           // 第3引数(y1)
98 :             AjcCipCallInfoInsReal (hWndCip, pCall, 1.0);           // 第4引数(x2)
99 :             AjcCipCallInfoInsReal (hWndCip, pCall, 1.5);           // 第5引数(y2)
100 :            AjcCipCallInfoInsReal (hWndCip, pCall, 2.0);           // 第6引数(x3)
101 :            AjcCipCallInfoInsReal (hWndCip, pCall, 0.5);           // 第7引数(y3)
102 :            // マクロ実行
103 :            if (AjcCipExec(hWndCip, pCall)) {
104 :                AjcCipPrintF(hWndCip, "¥n");
105 :                AjcCipPrintF(hWndCip, "マクロの実行を終りました。¥n");
106 :                pResult = AjcCipGetRetReal(hWndCip, &n);
107 :                AjcCipPrintF(hWndCip, "戻り値： 中心 x = %5.2f, y = %5.2f¥n", pResult[0], pResult[1]);
108 :            }
109 :            else {
110 :                AjcCipPrintF(hWndCip, "マクロの実行中にエラーが発生しました。¥n");
111 :            }
112 :            AjcCipCallInfoDelete(hWndCip, pCall);
113 :            fRun = FALSE;
114 :            // プログラム終了判断
115 :            if (fExit) {
116 :                DestroyWindow(hDlg);
117 :            }
118 :        }
119 :        else {
120 :            AjcCipPrintF(hWndCip, "マクロテキストに func() 関数がありません。¥n");
121 :        }
122 :    }
123 :    else {
124 :        AjcCipPrintF(hWndCip, "マクロの読み出しを失敗しました。¥n");
125 :    }
126 :
127 :    return TRUE;
128 : }
129 : //----- キャンセル -----//
130 : AJC_DLGPROC(Main, IDCANCEL          )
131 : {
132 :     if (fRun) {
133 :         AjcCipStop(hWndCip);
134 :         fExit = TRUE;
135 :     }
136 :     else {
137 :         DestroyWindow(hDlg);
138 :     }
139 :     return TRUE;
140 : }
141 : //-----//
142 : AJC_DLGMAP_DEF(Main)
143 :     AJC_DLGMAP_MSG(Main, WM_INITDIALOG      )
144 :     AJC_DLGMAP_MSG(Main, WM_DESTROY        )
145 :     AJC_DLGMAP_MSG(Main, WM_TIMER          )
146 :     AJC_DLGMAP_CMD(Main, IDCANCEL          )
147 : AJC_DLGMAP_END
148 :

```