DLL Wizard 6.7

リファレンス
Windows®

4D

DLL Wizard リ 6.7 ファレンス
Windows®

このマニュアルに記載されている事項は、将来予告なしに変更されることがあり、いかなる変更に関しても 4D SA は一切の責任を負いかねます。このマニュアルで説明されるソフトウェアは、本製品に同梱の License Agreement（使用許諾契約書）のもとでのみ使用することができます。

ソフトウェアおよびマニュアルの一部または全部を、ライセンス保持者がこの契約条件を許諾した上での個人使用目的以外に、いかなる目的であれ、電子的、機械的、またどのような形であっても、無断で複製、配布することはできません。

4th Dimension、4D Server、4D、4D ロゴ、4D ロゴ、およびその他の 4D 製品の名称は、4D SA の商標または登録商標です。

Microsoft と Windows は Microsoft Corporation 社の登録商標です。

Apple, Macintosh, Mac, Power Macintosh, Laser Writer, Image Writer, ResEdit, QuickTime は Apple Computer Inc.の登録商標または商標です。

その他、記載されている会社名，製品名は、各社の登録商標または商標です。

注意

このソフトウェアの使用に際し、本製品に同梱の License Agreement（使用許諾契約書）に同意する必要があります。ソフトウェアを使用する前に、License Agreement を注意深くお読みください。

目次

はじめに

4th Dimensionは数多くのコマンドからなる強力なプログラミング言語を提供していますが、一般的な4D言語には含まれていない特別な機能を利用したくなることもあります。

多くの場合、必要とされる機能を提供するダイナミック・リンク・ライブラリ（DLL）がすでに存在しています。DLLとはさまざまなアプリケーションから呼び出すことができるコードが集まったものです。別々のアプリケーション間で共有することができ、DLL内の関数の名前を単純にコールするだけで自動的にリンクが行われます。

Windowsプログラミング・インタフェース自体もDLLをベースにしています。機能のほとんどは、"Kernel32.DLL"、"User32.DLL"、あるいは"GDI32.DLL"に存在しています。

Windowsの機能をコールする方法のひとつは、その機能をコールするための4Dプラグインを C または C++ で書くことです。この方法は、複雑なことをさせようとする場合に適切です。しかし、単純なルーチン（例えば、ハードディスクの空き容量を知るためにWindows API関数"GetDiskFreeSpace"をコールすること）をコールするだけならば、それを実行するために4Dプラグインを書くことは時間をかけるだけの価値がないかもしれません。

前もって必要とされる事項：

■ ご使用のコンピュータにネイティブの開発環境がインストールされていること。

■ C または C++ を知っていること。

■ 4Dプラグインを書いて、コンパイルすること。

DLL Wizardはこの問題を回避する簡単な方法を提供します。DLL Wizardを使用すれば、グラフィックインタフェースを通してDLL内に存在している関数を宣言し、それを使用することができます。引数（パラメータ）を渡して、値を得ることもできます。このようなことを実行するために、DLL Wizardはプラグインを作成し、これらの機能を4Dで利用できるようにします。

DLL Wizardはメモリブロックを割り当て、そのブロックから値を読む、あるいは書くことができる、有益なユーティリティ（WinMemという名前のパッケージ内に実装されています）も提供します。

DLL WizardはDLL Toolsプラグインの代わりとなるものです。

ディスクへインストールする

DLL WizardはWindows版の4th Dimensionのみで利用可能です。4Dバージョン6.0以降が必要となります。ディスク空き容量は最小で2MB必要です。

▼ DLL Wizardを、ご使用のハードディスクにインストールするには：

1. DLL Wizard.4DB（または.4DC）とDLL Wizard.4DDをディスクにコピーします。

これで使用できる状態になります。

このデータベースはWindows 95またはWindows NT用に設計されています。DLL Wizardで生成されたプラグインは、Windows 95、Windows NT、Windows 3.1.1上の4D バージョン6およびバージョン3.5で利用できます。

制限

DLL Wizardは、32ビット用DLLのみを取り扱うことができます。

絶対最大値：

■ **総数または設計したプラグイン**：ディスクの空き容量に応ずる。

■ **プラグイン内のプロシージャの数**：数千*。

■ **プロシージャごとの引数（パラメータ）の数**：25。

■ **プラグインでコールされるDLLの数**：255[1]

1 実際の値は、ご使用になっている構成（OS、メモリ容量、ディスク容量）の制限のため、これらより少なくなります。

基本手順

この章では、DLL Wizardの使用方法を記述しています。

1. **DLL Wizardを開いて、「File」メニューから「New˘」を選択する。**
   - ■ １番目のページ（plug-in）で、プラグインに名前を付けます。
   - ■ ２番目のページ（procedures）で、関数宣言を追加します。
2. **これを行うには、「Add a procedure」ボタンをクリックする（第3章「サンプル」を参照）。**
3. **３番目のページ（final step）で、プラグインがデザインされているデータベースを選択する。**

   DLL Wizardはデータベースの隣に位置するWIN4DXフォルダ内にプラグインを作成します。

   必要な場合は、WIN4DXフォルダが作成されます。

   メモリ割り当てを行う場合は、WinMemプラグインをインストールすることもできます。

   プラグインを修正する場合は、「File」メニューから「Open˙˙」を選択します。

DLL Tools ユーザの方への注意

4Dで行なっている作業のほとんどは、DLL Wizardで作成できます。メソッドにはDLLへのコールのみが含まれ、コールの宣言は含まれません。

DLL Toolsとは違って、ライブラリのロード、解放は必要ありません：

■ 必要なライブラリは起動時に自動的にロードされます。

■ データベースが開かれている間はずっとロードされたままです。

■ 終了時に解放されます。

関数宣言はDLL Wizard内で行われます（その主目的）。4Dメソッド内で、その関数をコールするだけでよいのです（LoadLibraryなどをコールする必要はありません）。コンパイルされたデータベースでDLLをコールするために、4Dコマンドの**EXECUTE**を実行する必要はありません。定数も引数（パラメータ）として渡されます。

メモリ割り当てとアクセス関数は、別々のパッケージで用意されています。

DLL Toolsを使用する場合は以下のようになっていました：

```
Lib:=LoadLibrary("TheDLL.dll")
Call:=DLL declare(Lib;"BOOL TheProcedure(DWORD,DWORD)";0)
Value2:=20
Err:=CallDLL(Call;Value2;ReturnValue)
FreeCall(Call)
FreeLibrary(Lib)
```

DLL Wizardを使用すれば、以下のように簡単になります：

```
ReturnValue:=TheProcedure(Value1;20)
```

サンプルを通して DLL Wizard を学ぼう

以下のサンプルでは、Windows APIのGetCommandLineを使用して、起動時に4Dに与えられたコマンドを取り出すことを可能にしています。この関数を見てみると、Quick ViewはGetCommandLineAとGetCommandlineW関数を提供しています。

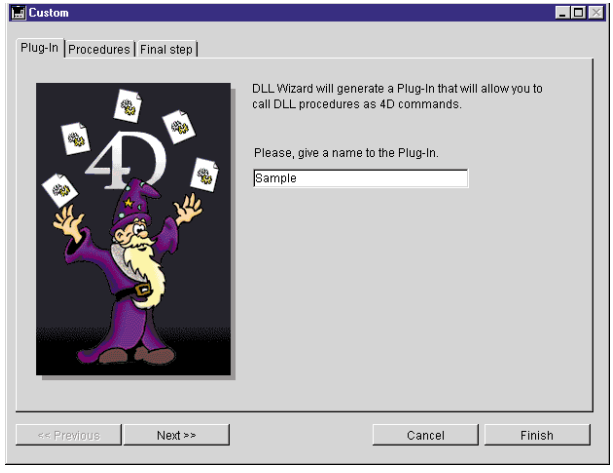
これはKernel32.dllが2つの関数を実装しているからです。1つは標準ASCII文字列を受け取るもの（GetCommandLineA）で、もう1つはワイドキャラクタ文字列を受け取るものです（GetCommandLineW）。CまたはC++のソースコードからGetCommandLineをコールする場合は、リンカーがASCIIを使用しているのかワイドキャラクタを使用しているのかをわかっているため、DLL内の適切な関数をコールします。DLL Wizardを使用する場合、DLLを直接コールできるので、どちらの関数をコールするべきかを知っていなければなりません。ここでは、4DはASCIIコードで動いているため、GetCommandLineAを使用することにします。

Visual C++は、この関数に関して以下のような情報を提供しています：

"GetCommandLine**関数はカレントプロセスのコマンドライン文字列へのポインタを返します。**"
LPTSTR GetCommandLine(VOID)

1. 「File」メニューから「New」を選択する。

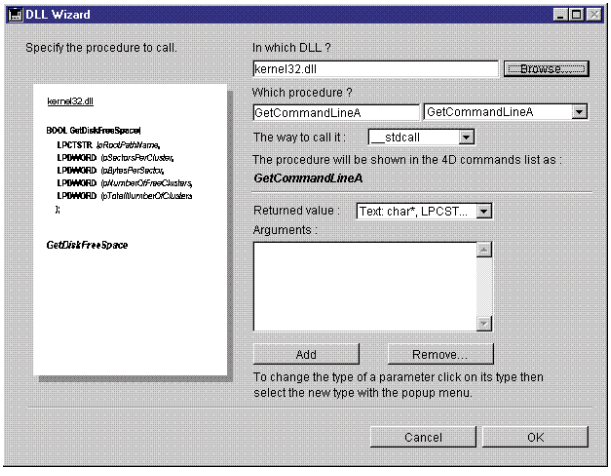
2. 「Plug-In」タブ画面で、プラグインに名前を付ける（Sample）。



3. 「Procedures」ウインドウで、「Add a procedure」ボタンをクリックする。

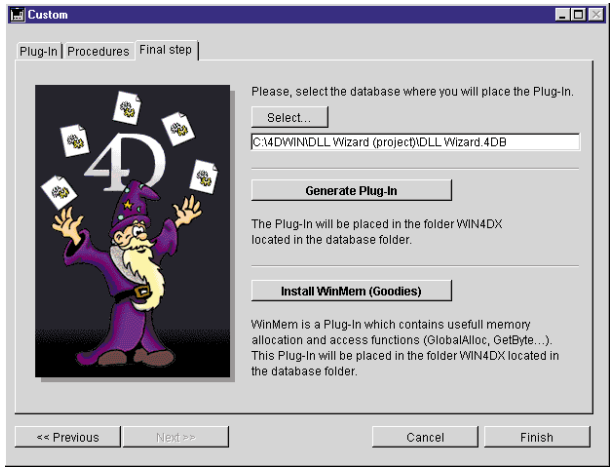
ルーチンエントリフォームで、引数（パラメータ）およびルーチンの戻り値を設定することができます：

- **DLL の名前： kernel32.dll**
- **ルーチンの名前： GetCommandLineA**
- **関数呼び出しのコンベンション（calling convention）を選択： _stdcall**
- **返り値の型を選択： text**



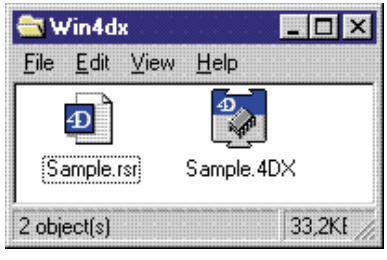
4. **修正の確認をする。**
5. **「Final Step」ウインドウで、作成したプラグインを使用するためのデータベースを選択する。**

6. 「Generate Plug-In」ボタンをクリックする。



WIN4DX フォルダ（指定したデータベースのフォルダ内にあります）を見てみましょう。「Sample.4DX」というプラグインがあるはずです。データベースを開いてみましょう。

プラグインはすでに使用できる状態になっています。



4D メソッド内では、プロシージャは以下のようにしてコールされます：

$cmdline:=*GetCommandLineA*

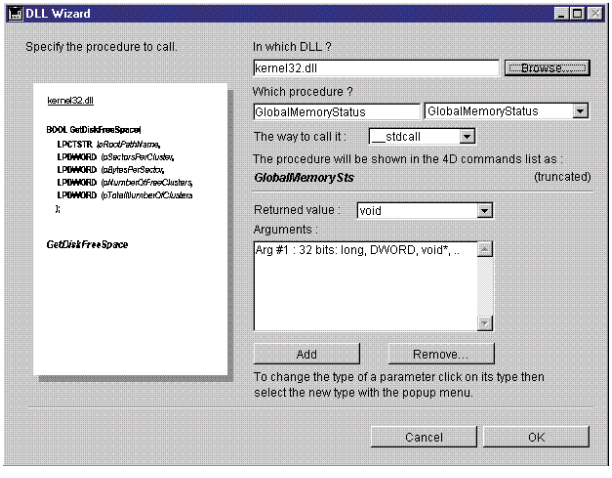
構造体を扱う

GlobalMemoryStatusを使用してみましょう。定義は以下のようになっています（Visual C++）：

```
VOID GlobalMemoryStatus(
    LPMEMORYSTATUS      lpBuffer        // pointer to the memory status structure
);

typedef _MEMORYSTATUS {
    DWORD   dwLength;               // sizeof(MEMORYSTATUS)
    DWORD   dwMemoryLoad;           // percent of memory in use
    DWORD   dwTotalPhys;            // bytes of physical memory
    DWORD   dwAvailPhys;            // free physical memory bytes
    DWORD   dwTotalPageFile;        // bytes of paging file
    DWORD   dwAvailPageFile;        // free bytes of paging file
    DWORD   dwTotalVirtual;         // user bytes of address space
    DWORD   dwAvailVirtual;         // free user bytes
} MEMORYSTATUS, *LPMEMORYSTATUS;
```

プロシージャエントリフォームで、以下のように宣言します：



注：プロシージャ名が15文字を越えているので（GlobalMemoryStatus）、名前は切り捨てられます（最初の14文字と最後の1文字）。

▼4Dメソッドでは、以下のようにコールされます：

```
   `バッファを構造体用に割り当てます。
   `WinMem プラグインがインストールされているものとします。
$lpBuffer:= WM GlobalAlloc(GMEM_FIXED;32)
WM SET LONG($lpBuffer;32)                    `バッファのサイズを指示します。
   `（MEMORYSTATUS の定義を参照）

   `呼び出しを実行
GlobalMemorySts($lpBuffer)

   `メンバーのアドレスとオフセットを指定して情報を引き出します。
$dwMemoryLoad:=WM Get long($lpBuffer+4)
$dwTotalPhys:=WM Get long($lpBuffer+8)
$dwAvailPhys:=WM Get long($lpBuffer+12)
$dwTotalPageFile:=WM Get long($lpBuffer+16)
$dwAvailPageFile:=WM Get long($lpBuffer+20)
$dwTotalVirtual:=WM Get long($lpBuffer+24)
$dwAvailVirtual:=WM Get long($lpBuffer+28)

$err:=WM GlobalFree($lpBuffer)
```

DLL の名前

DLL の名前のみを指定した場合、DLL はアプリケーションのフォルダ、カレントフォルダ、あるいはシステムフォルダのどこかに存在していなければなりません。

フルパス名を指定した場合は、DLL は指定したフォルダに存在していなければなりません。

データベースが開かれた際、DLL が見つからない場合は、「XXX.dll not loaded」というアラートが表示されます。通常、Kernel32.dll や User32.dll などでは、このようなアラートは表示されません。

プロシージャの宣言

プロシージャを宣言するには、数と引数（パラメータ）の型と戻り値を知る必要があります。

そのためには、Windows32 ビット API の記述を見る必要があります。これらの情報は、Windows プログラミング本、あるいは Visual C++ や Borland C++ 等の開発システムに付属しているオンラインヘルプで見つけることができます。特別な DLL（例：FAX ソフトウェア）を使用している場合は、ドキュメント、あるいはヘッダファイルを見ても結構です。

データベースが開かれた際、プロシージャが DLL 内に見つからない場合は、Quick View か Dumpbin を使用するか、または「Browse」をクリックして DLL を選択することができます（上のサンプルを参照）。

関数呼び出しのコンベンション

大抵の場合において、プロシージャはPascalコンベンション（_stdCall）を使用して呼び出されます。ご使用のDLLがCコンベンションを使用している場合は、_cdeclを選択してください。

引数（パラメータ）と 4D 変数型

引数（パラメータ）

以下の引数の型を宣言することができます：

■**8ビット**：char, BOOL, BYTE, CHAR, UCHAR, BOOLEAN, CCHAR

■**16ビット**：short, WORD, UWORD, SHORT, USHORT

■**32ビット**：long, word, int, short*, word*, long*, int*, void*, DWORD, LONG, LPVOID, UINT, GLOBALHANDLE, HANDLE, HLOCAL, LPDWORD, LPBOOL, LPBYTE, LPWORD, LPLONG

■**文字列ポインタ**：char*, LPCSTR, LPSTR, LPCTSTR, LPTSTR, NPSTR, PCSTR, PCWSTR, PSTR, PTSTR

■**double値（32ビット）**：double, GLdouble

■**float値（32 ビット）**：float

■**void返り値**：void, VOID

ご使用の関数がこのリストにない引数の型を受け取る場合は、互換性のある引数の型を渡すようにしてください。また、構造体へのポインタを期待している場合は、void*のような32ビットのポインタとして宣言してください。

特別なケース：構造体パラメータ

関数内で構造体を引数（パラメータ）として宣言したい場合（構造体へのポインタではなく、構造体の内容）は、構造体のすべてのメンバーを連続的に渡す必要があります。

▼例： RECT **パラメータを受け取る関数をコールする。**

オリジナルの宣言：

```
void DoSomething (RECT theRect);
```

RECTは、Windows APIでは４つのlongからなる構造体として宣言されています。呼び出しを以下のように宣言する必要があります：

```
Arg #1: long
Arg #2: long
Arg #3: long
Arg #4: long
```

4D変数、あるいは定数を渡すことができます。DLL Wizardは、realからlongへの型変換のように、自動的に型変換を処理することが可能です。

4D 変数型

宣言された型へ渡すことができる4D変数型には以下のものがあります：

■ 8ビット：（char, BOOL, BYTE, CHAR, UCHAR, BOOLEAN, CCHAR）
4D変数のinteger型、longint型またはreal型を渡すことができます。最も適切な変数型は4D変数のInteger型かLongint型です。4D変数のReal型を渡すこともできますが、時間がかかります（型変換が行われるため）。ご使用の関数がsigned byteを期待している場合は、SetByte関数の例を参照してください。unsigned byteをsigned byteに変換する方法が示されています。

■ 16ビット：（short, WORD, UWORD, SHORT, USHORT）
4D変数のinteger型、longint型、またはreal型を渡すことができます。最も適切な変数型は4D変数のlongint型です。4D変数のReal型を渡すこともできますが、時間がかかります（型変換が行われるため）。

■ 32ビット：（long, word, int, void*, DWORD, LONG…）
4D変数のinteger型、longint型、またはreal型を渡すことができます。最も適切な変数型は4D変数のlongint型です。4D変数のReal型を渡すこともできますが、時間がかかります（型変換が行われるため）。

■ 文字列ポインタ：（char*, LPCSTR, LPSTR, LPCTSTR, LPTSTR…）

4D変数のalpha型（文字型）またはtext型を渡すことができます。alpha型（文字型）を渡す場合は、DLL関数が適切なサイズの文字列を返すことを確認してください。仮に、20字の長さのalpha型変数を宣言していて、80字の長さの文字列が返ってくるDLL関数を呼び出している場合は、最初の20字分しか4D変数の中には入りません。ご使用のDLLが、より大きな文字列を返す場合は、text型変数を使うことができます。text型変数の制限は32000字までです。

4Dは内部的にはMacintosh拡張キャラクタを使用しているので、DLL関数へ渡される文字列は関数を呼び出す前に自動的にANSIに変換され、その後Macintosh ASCIIコードへ再変換されます。

■ float値（32ビット）：（float）

4D変数のinteger型、longint型、またはreal型を渡すことができます。最も適切な変数型は4D変数のreal型です。その他の型を渡すこともできますが、時間がかかります（型変換が行われるため）。4D変数のreal型はdoubleなので、精度が失われる可能性があります。

■ double値（64ビット）：（double, GLdouble）

4D変数のinteger型、longint型、またはreal型を渡すことができます。最も適切な変数型は4D変数のreal型です。その他の型を渡すこともできますが、時間がかかります（型変換が行われるため）。

DLL Wizard によって提供されているこのプラグインは、グローバルハンドルやポインタを割り当てることのできる便利な関数を持っています。

これらの関数は、基本的な Windows のメモリ管理ルーチンへ簡単にアクセスできるように設計されています。

WinMem プラグインは、割り当てられたブロック内部にあるデータを読む、あるいは書き込むことができるようにする関数も持っています（「WinMem：アクセス」の章を参照してください）。

WinMem：割り当て（Allocation）

WM GlobalAlloc

WM GlobalAlloc (Flags; Size) → Long

引数	タイプ	説明
Flags	LongExpr	オブジェクトの割り当て属性
Size	LongExpr	割り当てるバイト数

説明

WM Global Alloc は、Windows API GlobalAlloc 関数にマップされています。要求された量のメモリを割り当て、割り当てたブロックのハンドルを返します。

DLL Wizard を使用して宣言された関数の中には、引数としてメモリブロックへのポインタを要求するものもあります。その場合のブロックは GlobalAlloc で作成できます。

以下の値をそのまま使用するか、もしくは加算して使います：

0 GMEM_FIXED、固定したメモリを割り当てます。戻り値はメモリブロックへのポインタです。使用前にブロックをロックまたはロック解除する必要はありません。

2 GMEM_MOVABLE、移動可能なメモリを割り当てます。戻り値はメモリオブジェクトへのハンドルになります。メモリブロックへのポインタを得るためにGlobal-Lockを呼び出す必要があります。

8192 GMEM_DDESHARE、割り当てられたブロックはDDEの対話やクリップボード操作に使用されます。

256 GMEM_DISCARDABLE、割り当てられたブロックを放棄することができます。このフラグはGMEM_FIXEDと一緒にすることはできません。

16 GMEM_NOCOMPACT、割り当て要求を満たすためにメモリを圧縮することはしません。

32 GMEM_NODISCARD、割り当て要求を満たすためにメモリを放棄することはしません。

64 GMEM_ZEROINIT、メモリ内容を0に初期化します。

関数が失敗した場合は、戻り値は0になります。

WM GlobalFree

WM GlobalFree (Handle) → Long

引数	タイプ	説明
Handle	LongExpr	グローバルメモリオブジェクトへのハンドル

説明

WM GlobalFreeは、GlobalAlloc関数で割り当てられたメモリブロックを解放します。

関数が成功した場合、戻り値は0になります。失敗した場合、戻り値はグローバルメモリオブジェクトのハンドルの値になります。

WM GlobalLock

WM GlobalLock (Handle) → Long

引数	タイプ	説明
Handle	LongExpr	グローバルメモリオブジェクトへのハンドル

説明

WM GlobalLockは、グローバルメモリオブジェクトをロックして、オブジェクトメモリブロックの最初のバイトへのポインタを返します。一旦ロックされると、メモリブロックはロックが解除されるまで、移動または放棄されることはありません。

この関数を使用する必要があるのは、GMEM_MOVABLEフラグを使用して割り当てられたオブジェクトに対してのみです。

ハンドルをロックするたびに１つずつ増え、ロック解除するたびに１つずつ減る内部的なカウンタがあります。ブロックは、このカウンタが0になった時にだけ本当にロック解除されます。このような方法は、ハンドルをロックするAというプロシージャが、同じハンドルをロック／ロック解除するBというプロシージャを呼び出して、ハンドルがロックされたままの状態でプロシージャAに戻ってくる場合に便利です。

WM GlobalUnlock

WM GlobalUnlock (Handle) → Long

引数	タイプ	説明
Handle	LongExpr	グローバルメモリオブジェクトへのハンドル

説明

WM GlobalUnlockは、グローバルメモリオブジェクトをロック解除します。一旦ロック解除されると、メモリブロック上のポインタは有効ではなくなるので、それを使用するとアクセス権例外（access privilege exception）を引き起こす可能性があります。

この関数を使用する必要があるのは、GMEM_MOVABLEフラグを使用して割り当てられたオブジェクトに対してのみです。

ハンドルは、内部カウンタが0にセットされた時にだけ本当にロック解除されます。この関数を呼び出した後でもオブジェクトがロックされたままの場合（ロックカウンタが0になっていない場合）、戻り値は1になります。オブジェクトがロック解除されている場合は0になります。

WM GlobalSize

WM GlobalSize (Handle) → Long

引数	タイプ	説明
Handle	LongExpr	グローバルメモリオブジェクトへのハンドル

説明

WM GlobalSizeは、指定したメモリオブジェクトのサイズをバイト数で返します。

指定したハンドルがすでに有効でない場合、またはオブジェクトがすでに放棄されている場合、戻り値は0になります。

注意：メモリブロックのサイズは、メモリが割り当てられた時に要求されたサイズよりも大きいことがあります。したがって、例えばブロック内に保管した要素の数などの計算は、このサイズを信頼して行うことができません。

WM GlobalReAlloc

WM GlobalReAlloc (Handle; Size; Flags) → Long

引数	タイプ	説明
Handle	LongExpr	グローバルメモリオブジェクトへのハンドル
Size	LongExpr	ブロックの新しいサイズ
Flags	LongExpr	オブジェクトの再割り当ての方法

説明

WM GlobalReAllocは、指定したメモリオブジェクトのサイズや属性を変更します。

指定したハンドルがすでに有効でない場合、またはオブジェクトがすでに放棄されている場合、戻り値は0になります。

注意：メモリブロックのサイズは、メモリが割り当てられた時に要求されたサイズよりも大きいことがあります。したがって、例えばブロック内に保管した要素の数などの計算は、このサイズを信頼して行うことができません。

WinMem ：アクセス

これらの関数を使用すると、メモリ内のどこにでも特定の値を入れることができるようになります。これは、DLL関数が特別な値を含んだ構造体へのポインタを必要とする場合に便利です。メモリ管理関数を使用すれば、メモリブロックを割り当て、そのブロックをこれらのルーチンを使用して満たすことができます。

注意：構造体を扱う場合は、バイトアラインメントに注意してください。

WM SET BYTE

WM SET BYTE (Pointer; Value)

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス
Value	LongExpr	セットするバイト値（0～255）

説明

WM SET BYTEは、指定したアドレスへunsignedバイトをセットします。unsignedバイトは1バイト長（8ビット）で、0から255までの値を持つことができます。

例題

-128から+127のsigned値を渡したい場合は、以下の変換アルゴリズムを使用して、ルーチンへ渡す前に値を変換する方法があります：

```
If (value<0)
   value:=256+value
End if
WM SET BYTE (pt;value)
```

WM SET SHORT

WM SET SHORT (Pointer; Value)

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス
Value	LongExpr	セットするshort値（0～65535）

説明

WM SET SHORTは、指定したアドレスへunsigned short値をセットします。unsigned shortは2バイト長（16ビット）で、0から65535までの値を持つことができます。

例題

-32768から+32767のsigned値を渡したい場合は、以下の変換アルゴリズムを使用して、ルーチンへ渡す前に値を変換する方法があります：

```
If (value<0)
   value:=65536+value
End if
WM SET SHORT (pt;value)
```

WM SET LONG

WM SET LONG (Pointer; Value)

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス
Value	LongExpr	セットするsigned long値

説明

WM SET LONGは、指定したアドレスへlong値をセットします。long値は4バイト長（32ビット）で、-2147483648から+2147483647までの値を持つことができます。

WM SET DOUBLE

WM SET DOUBLE (Pointer; Value)

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス
Value	RealExpr	セットする double 値

説明

WM SET DOUBLE は、指定したアドレスへ double 値をセットします。double 値は 8 バイト長（64 ビット）で、real 値を持つことができます。64 ビット上に格納された real 値は正数または負数の場合があり、15 個の有効数字を持った 1.7 E-308 から 1.7 E+308 の範囲になります。

例題

WM SET DOUBLE (pt;3.5667)

WM SET FLOAT

WM SET FLOAT (Pointer; Value)

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス
Value	RealExpr	セットする float 値

説明

WM SET FLOAT は、指定したアドレスへ float 値をセットします。float 値は 4 バイト長（32 ビット）で、real 値を持つことができます。32 ビット上に格納された real 値は正数または負数の場合があり、9 個の有効数字を持った 3.4 E-38 から 3.4 E+38 の範囲になります。4D の real 変数は double なので、型変換が行われる際に精度が失われる可能性があります。

例題

WM SET FLOAT(pt;3.5667)

WM SET CSTRING

WM SET CSTRING (Pointer; Text)

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス
Text	TextExpr	コピーするテキスト

説明

WM SET CSTRINGは、指定したアドレスへ指定したテキストをコピーします。このテキストはANSIキャラクタへ変換されます（4Dは内部的にMac ASCIIコード表現を使用しています）。DLL関数のほとんどは文字列がNullで終わっていると仮定しているので、テキストの最後にはnullキャラクタが加えられます。

ご使用の関数がUNICODEを必要とする場合は、Kernel32.DLLにあるWideCharToMultiByte等のような、変換を行うDLL関数を最初に呼び出す必要があります。

例題

WM SET CSTRING(pt;"Hello World")

WM SET PSTRING

WM SET PSTRING (Pointer; Text)

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス
Text	TextExpr	コピーするテキスト

説明

WM SET PSTRINGは、指定したアドレスへ指定したテキストをコピーします。

このテキストはANSIキャラクタへ変換されます（4Dは内部的にMac ASCIIコード表現を使用しています）。テキストの最初に文字列の長さを示すキャラクタが加えられます。ご使用の関数がUNICODEを必要とする場合は、Kernel32.DLLにあるWideCharToMultiByte等のような、変換を行うDLL関数を最初に呼び出す必要があります。

例題

WM SET PSTRING(pt;"Hello World")

WM SET STRING

WM SET STRING (Pointer; Text)

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス
Text	TextExpr	コピーするテキスト

説明

WM SET STRINGは、指定したアドレスへ、指定したテキストをコピーします。

このテキストはANSIキャラクタへ変換されます（4Dは内部的にMac ASCIIコード表現を使用しています）。テキストの最初に文字列の長さを示すキャラクタが加えられます。ご使用の関数がUNICODEを必要とする場合は、Kernel32.DLLにあるWideCharToMultiByte等のような、変換を行うDLL関数を最初に呼び出す必要があります。

例題

WM SET STRING(pt;"Hello World")

WM SET BLOB

WM SET BLOB (Pointer; Value; Length)

引数	タイプ	説明
Pointer	LengExpr	メモリアドレス
Value	BlobExpr	コピーするバイト
Length	LongExpr	セットするバイト数

説明

WM SET BLOBは指定したアドレスへBLOBの所定のバイト数をコピーします。データはBLOBのオフセット0からコピーされます。

WM SET BLOB(pt;vBlob;200)

これらの関数を使用すると、メモリ内のどこにある値でも読むことができます。これは、DLL関数が構造体、あるいは読み出す必要がある特別な値を含んでいるデータへのポインタを返す場合に便利です。

WM Get byte

WM Get byte (Pointer) → Value

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス

説明

WM Get byteは、指定したアドレスにあるunsignedバイトを読み出します。unsignedバイトは1バイト長（8ビット）で、0から255までの値を持つことができます。

例題

signed値（-128から+127）を読み出したい場合は、読み出した後で以下の変換アルゴリズムを使用して変換することができます：

```
value:=WM Get byte(pt)
If (value>127)
   value:=value-256
End if
```

WM Get short

WM Get short (Pointer) → Value

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス

説明

WM Get shortは、指定したアドレスにあるunsigned short値を読み出します。unsigned shortは2バイト長（16ビット）で、0から65535までの範囲の値を持つことができます。

例題

signed値（-32768から+32767）を読み出したい場合は、読み出した後で以下の変換アルゴリズムを使用して変換することができます：

```
value:=WM Get short(pt)
If (value>32767)
   value:=value-65536
End if
```

WM Get long

WM Get long (Pointer) → Value

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス

説明

WM Get longは、指定したアドレスにあるlong値を読み出します。long値は4バイト長（32ビット）で、-2147483648から+2147483648までの値を持つことができます。

WM Get double

WM Get double (Pointer) → Value

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス

説明

WM Get doubleは、指定したアドレスにあるdouble値を読み出します。double値は8バイト長（64ビット）で、real値を持つことができます。64ビット上に格納されたreal値は、正数または負数の場合があり、15個の有効数字を持った1.7 E-308から1.7 E+308の範囲になります。

例題

```
C_REAL(dbl)
dbl:=WM Get double(pt)
```

WM Get float

WM Get float (Pointer) → Value

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス

説明

WM Get floatは、指定したアドレスにあるfloat値を読み出します。float値は4バイト長（32ビット）で、real値を持つことができます。32ビット上に格納されたreal値は、正数または負数の場合があり、9個の有効数字を持った3.4 E-38から3.4 E+38の範囲になります。4Dのreal変数はdoubleなので、型変換が行われる際に精度が失われる可能性があります。

例題

```
C_REAL(dbl)
dbl:=WM Get float(pt)
```

WM Get cstring

WM Get cstring (Pointer) → Text

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス

説明

WM Get cstringは、指定したアドレスに格納されているテキスト（C文字列と仮定）のコピーを返します。このテキストはANSIからMacintoshキャラクタへ変換されます（4Dは内部的にMac ASCIIコード表現を使用しています）。

UNICODE文字列を読み出したい場合は、Kernel32.DLLにあるMultiByteToWideChar等のような、変換を行うDLL関数を最初に呼び出す必要があります。

例題

```
C_TEXT(vText)
vText:=WM Get cstring(pt)
```

WM Get pstring

WM Get pstring (Pointer) → Text

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス

説明

WM Get pstringは、指定したアドレスに格納されているテキスト（P文字列と仮定）のコピーを返します。このテキストはANSIからMacintoshキャラクタへ変換されます（4Dは内部的にMac ASCIIコード表現を使用しています）。

UNICODE文字列を読み出したい場合は、Kernel32.DLLにあるMultiByteToWideChar等のような、変換を行うDLL関数を最初に呼び出す必要があります。

例題

```
C_TEXT(vText)
vText:=WM Get pstring(pt)
```

WM Get string

WM Get string (Pointer; Length) → Text

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス
Length	IntegerExpr	コピーするバイト数

説明

WM Get stringは、指定したアドレスに格納されているテキストの所定のバイト数のコピーを返します。このテキストはANSIからMacintoshキャラクタへ変換されます（4Dは内部的にMac ASCIIコード表現を使用しています）。

UNICODE文字列を読み出したい場合は、Kernel32.DLLにあるMultiByteToWideChar等のような、変換を行うDLL関数を最初に呼び出す必要があります。

例題

```
C_TEXT(vText)
vText:=WM Get string(pt;len)
```

WM Get BLOB

WM Get BLOB (Pointer; Length) → Blob

引数	タイプ	説明
Pointer	LongExpr	メモリアドレス
Length	LongExpr	取り出すバイト数

説明

WM Get BLOBは、指定したアドレスにある所定のバイト数のコピーを含んでいるBLOBを返します。

例題

```
C_BLOB(vBlob)
vBlob:=WM Get BLOB(pt;200)
```

コマンド索引

G

S