Windows プログラミング入門の入門の入門・・くらいの読み物

１． Win32 API

あるプラットフォーム(OS やミドルウェア)向けのソフトウェアを開発する際に使用できる命令や関数の集合のことを API（Application Program Interface）と言います。
Windows アプリケーションの開発を行うとき（ウィンドウを表示したり、メッセージボックスを表示する等）も、この API を使用します。
現在 Windows で使用できる API は Win32API と呼ばれ、その関数の数は Windows2000 にいたるまでに 15000 以上もの数になりました。

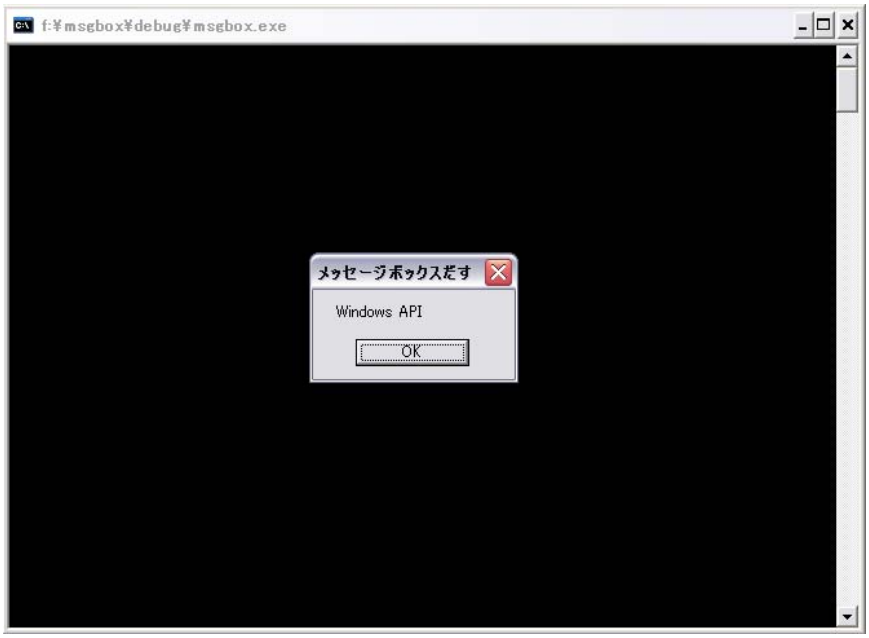
コンソールアプリケーションのような、Windows アプリケーションではないコードでも、実行環境が Windows であることが分かっている場合、API を使用することが出来ます。例えば、次のコードではメッセージボックスを表示させるために API を使用しています。

```
#include <windows.h>

int main()
{
        MessageBox(NULL, "Windows API", "メッセージボックスだす", MB_OK);
        return 0;
}
```

API を使用するため、<Windows.h>をインクルード

API の一つである、MessageBox()関数

実行結果:



Windows に、「メッセージボックスを表示してくれ」と要求しています。
このように、OS である Windows に対し、何らかの処理・動作を要求するときに使う関数群が、Win32API です。
さて、上記では Win32API を使用してはいますが、とても Windows アプリケーションと呼べるものではありません。

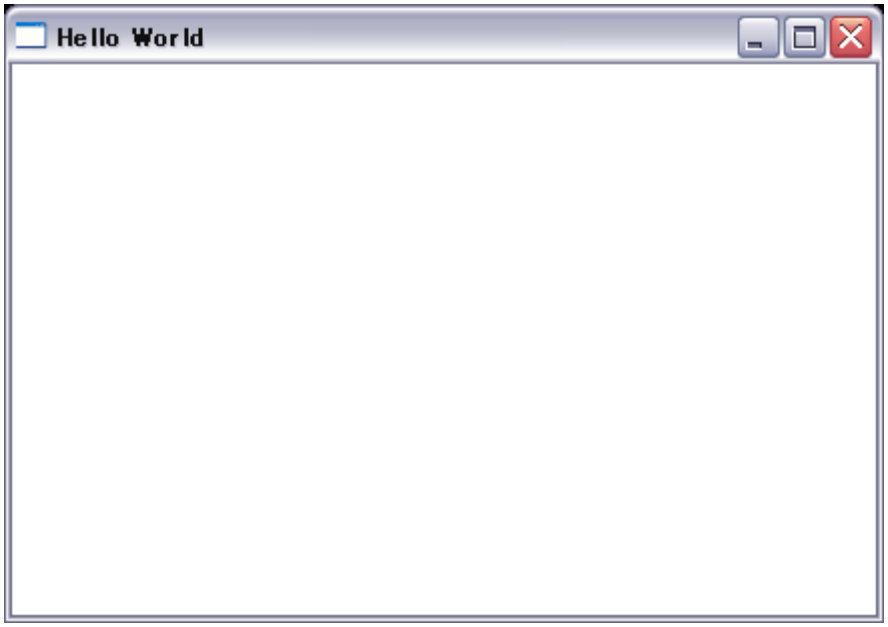
２．複雑な骨組み

一般的な Windows アプリケーションは、メモ帳や Word のように、必ずアプリケーションのウィンドウが表示されます。
ウィンドウを持つ最小のプログラムは、以下のようになります。

```
#include    <windows.h>

/*ウインドウプロシージャのプロトタイプ宣言*/
LRESULT CALLBACK WindowProc(HWND,UINT,WPARAM,LPARAM);

/*  アプリケーションエントリーポイント  */
int WINAPI WinMain(HINSTANCE    hInstance,
                   HINSTANCE    hPrevInstance,
                   LPSTR        CmdLine,
                   int          CmdShow)
{
    HWND            hwnd;   /*  メインウインドウのウインドウハンドル    */
    MSG             msg;    /*  メッセージキューから取得したメッセージ  */
    WNDCLASS        wc;     /*  ウインドウクラス登録用の構造体          */

    wc.style        =0;
    wc.lpfnWndProc  =WindowProc;
    wc.cbClsExtra   =0;
    wc.cbWndExtra   =0;
    wc.hInstance    =hInstance;
    wc.hIcon        =LoadIcon(NULL,IDI_APPLICATION);
    wc.hCursor      =LoadCursor(NULL,IDC_ARROW);
    wc.hbrBackground=(HBRUSH)(COLOR_WINDOW+1);
    wc.lpszMenuName =NULL;
    wc.lpszClassName="Hello";

    if(RegisterClass(&wc)==0)       /*  ウインドウクラス登録    */
        return  0;

    hwnd=CreateWindow(  "Hello",    /*  ウインドウ作成          */
                        "Hello World",
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        NULL,
                        (HMENU)NULL,
                        hInstance,
                        0);
    if(hwnd==NULL)
        return  0;
```

```
    ShowWindow(hwnd,CmdShow);               /*  ウインドウの表示          */
    UpdateWindow(hwnd);                     /*  ウインドウの最初の更新    */

    while(GetMessage(&msg,NULL,0,0))        /*  メッセージループ          */
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    return  msg.wParam;
}

/*  ウインドウプロシージャ  */
LRESULT CALLBACK WindowProc(HWND hwnd,UINT message,WPARAM wparam,LPARAM lparam)
{
    switch(message)
    {
        case WM_DESTROY:
            PostQuitMessage(0);
            return  0;
    }
    return  DefWindowProc(hwnd,message,wparam,lparam);
}
```

実行結果:



ウィンドウを表示するだけの、何も処理しないアプリケーションです。
これだけのことをするために、非常に多くのコードを書かなければなりません。
しかも、今まで見たことのないような型の変数や、やたら多くの引数を持つ関数が呼ばれたりもしています。

Windows は CUI 環境が基本となっている OS とは違い、GUI を持つ OS です。
そのため、一つのウィンドウを表示させるだけでも、非常に多くの決まりごとが存在します。ウィンドウの形やサイズ変更可否、キャプションに指定する文字列等・・
これらを一つずつ指定してやって、初めて Windows 上で動くウィンドウが作成できるのです。

しかし理屈はわかっても、これでは Excel のような Windows アプリケーションを作ろうとしたら、どれほど多くのコードを書かなければならなくなるか、考えただけで気が遠くなってしまうでしょう。

そこで、マイクロソフトは Windows アプリケーションの作成を容易にするために、MFC というクラスライブラリを作成し、開発者に提供しています。

※　GUI・CUI
操作しやすいようにウィンドウやアイコンなどを使い、コンピュータをわかりやすく使用できるようにしたのがＧＵＩ（グラフィカル・ユーザー・インターフェース）。ユーザーがデスクトップのアイコンをマウスでクリックするだけで、簡単にファイルの操作やアプリケーションソフトの起動などを実行できるようになっている。ＧＵＩとは反対に、ファイルの操作や演算の結果を文字のみで表示させるものをＣＵＩ（キャラクター・ユーザー・インターフェース）と呼ぶ。ＣＵＩのＭＳ-ＤＯＳから、ＧＵＩのウィンドウズに移行したことによってパソコンの操作性も格段に向上。初心者も比較的わかりやすくコンピュータを操作することができるようになった。
しかし、コアな開発者ほど CUI を好む傾向がある。これは、GUI は全てのオペレーションをユーザが指定しなければならないのに対し、CUI は覚えてさえしまえば複数の命令を一度に行えるある種の「便利さ」があるためである。

３．MFC（Microsoft Foundation Class）

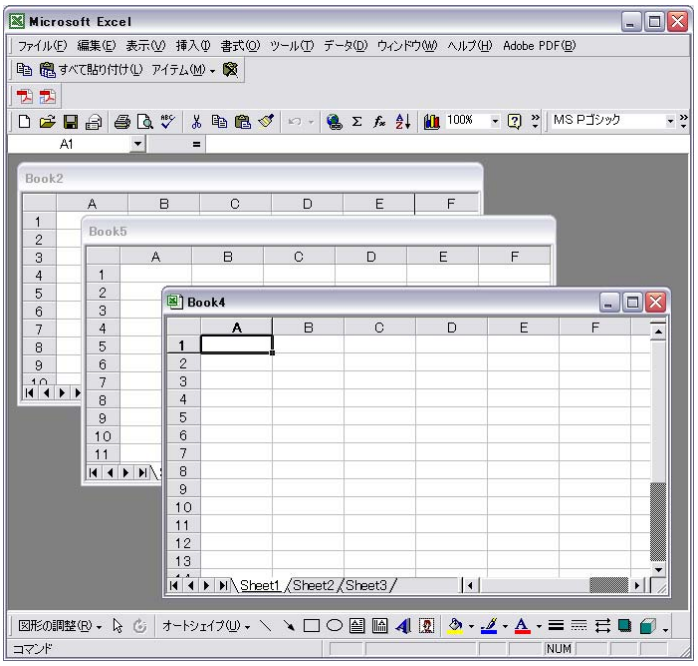
同じ系列の API を集め、それぞれの API に共通の部分をラップして、よりコーディング量が減るように設計されたれたものが、MFC といえるでしょう。
MFC を使用すれば、市販されている程度の高いアプリケーションに負けないソフトウェア開発が可能になります。
そして、プログラミングの大きな手助けとなる、数多くのクラスが用意されています。

※　MFC も全ての API をラップしてあるわけではありません。
　　Windows のシェル API など、わりとよく使用すると思われる API 群は、意外とラップされていないのも現実です。
　　逆にコモンコントロールなどは、うまくラップされています。

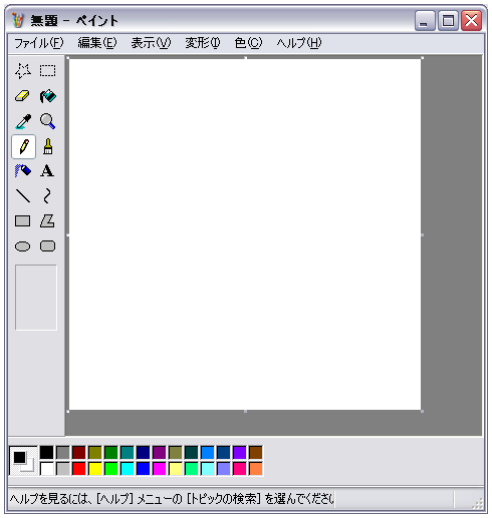
MFC で作れるアプリケーションの形態として、代表的なものは以下の３つです。

- MDI アプリケーション
- SDI アプリケーション
- ダイアログベースアプリケーション

MDI（マルチドキュメントインターフェース）アプリケーションとは、例えば Excel のように、一つの親ウィンドウの中にいくつものドキュメントを開けるタイプのアプリケーションです。一つのアプリケーションで複数のデータを同時に編集することが可能です。



これに対し、SDI（シングルドキュメントインターフェース）アプリケーションとは、ウィンドウズに付属しているペイントやメモ帳のように、一つのウィンドウで一つのデータを編集するタイプのものをいいます。



一つのウィンドウで、
一つのドキュメントを
編集する

最後のダイアログベースアプリケーションとは、一つのウィンドウで一つの処理を行うタイプですが、上記 2 タイプとは確実に違う点があります。
上記 2 タイプは編集データの処理・管理に『ドキュメント・ビュー・アーキテクチャ』という技術が使われるのに対し、ダイアログベースアプリケーションはその技術を使用しません。
また、ウィンドウのサイズ変更もあまり利用されません。
フリーツールなど、シンプルなアプリケーションを実装する場合、よく使われるタイプです。

シンプルな計算機機能を
提供する

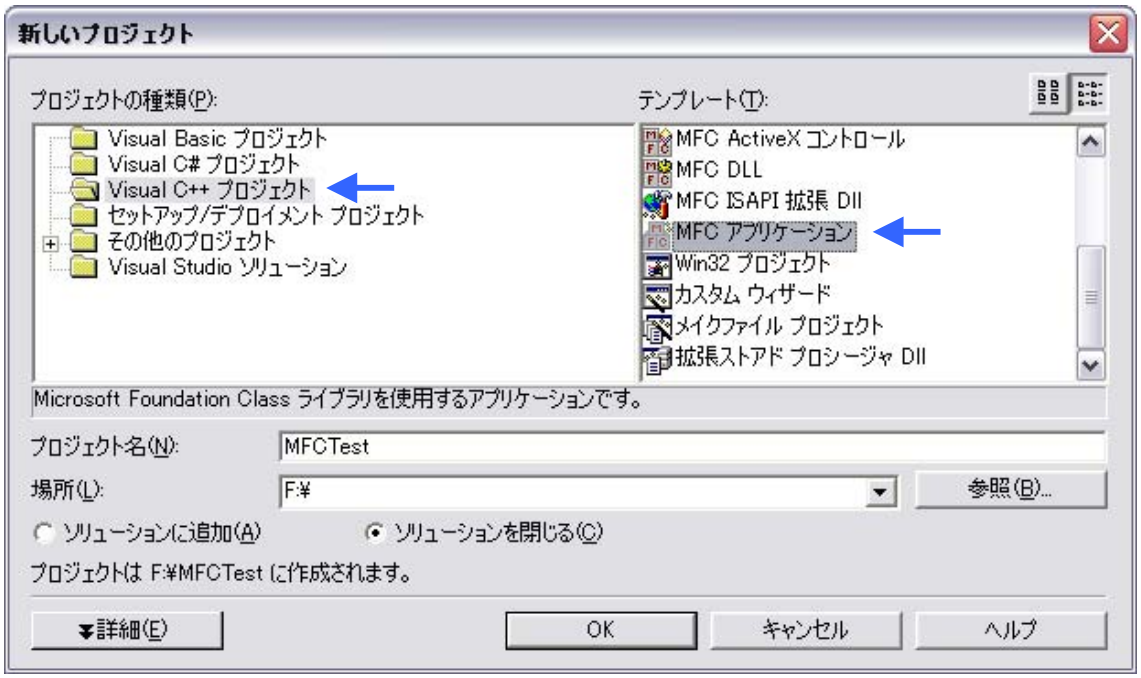
※　ここでは『ドキュメント・ビュー・アーキテクチャ』については深く説明できませんが、非常に簡単に言うと「データ処理はドキュメントクラスで、UI 処理はビュークラスで行うよう設計されており、この 2 つをあわせて一単位としてデータを扱う技術」です。

本研修では、仕組みが一番シンプルなダイアログベースを用いて、AppWizard の使用方法からイベントハンドラの作成方法までを学んでいきます。

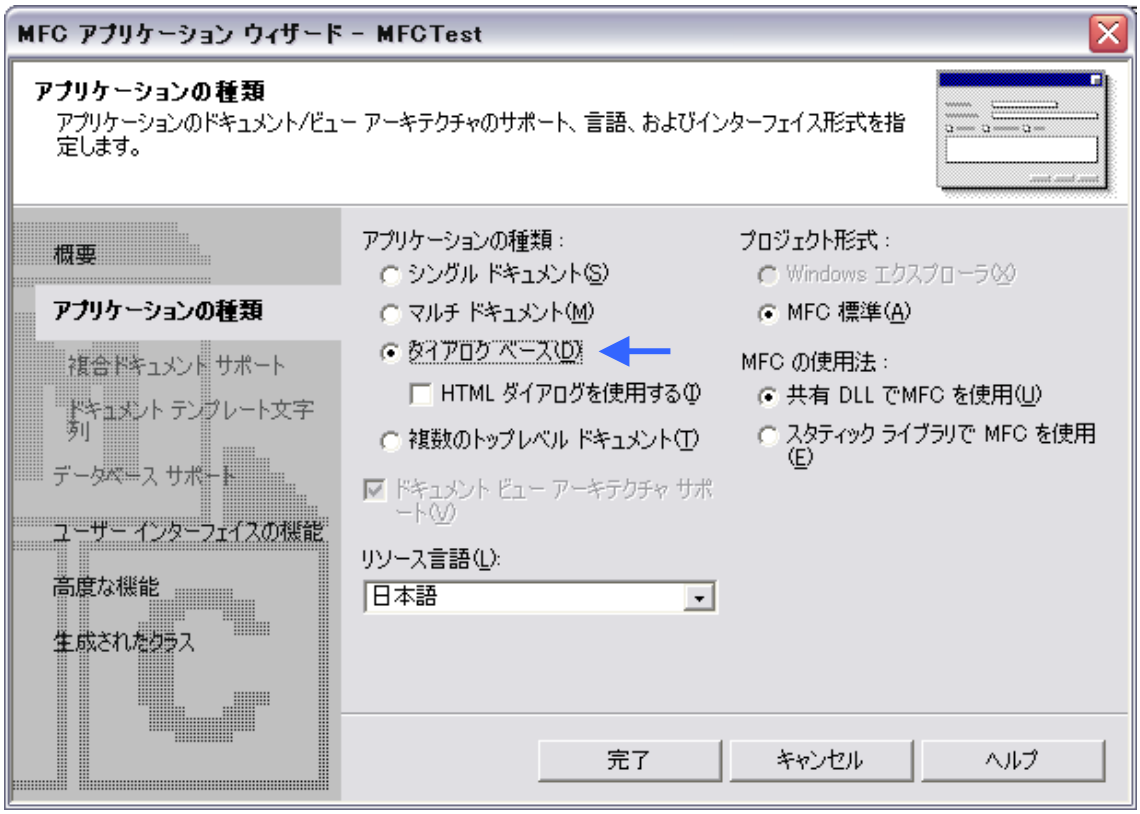
４． AppWizard

AppWizard を用いて MFC アプリケーションのプロジェクトを作成する手順を示します。

１）VisualStudio.Net の「プロジェクトの新規作成」にて表示されるダイアログボックスで、「プロジェクトの種類」には『Visual C++ プロジェクト』、「テンプレート」には『MFC アプリケーション』を選択し、OK ボタンを押下します。



2）「MFC アプリケーションウィザード」が表示されるので、「アプリケーション」タブをクリックし、「アプリケーションの種類」で『ダイアログ ベース』を選択し、完了ボタンを押下します。



3）ウィザードによってスケルトンコードが生成され、ダイアログエディタが表示されます。

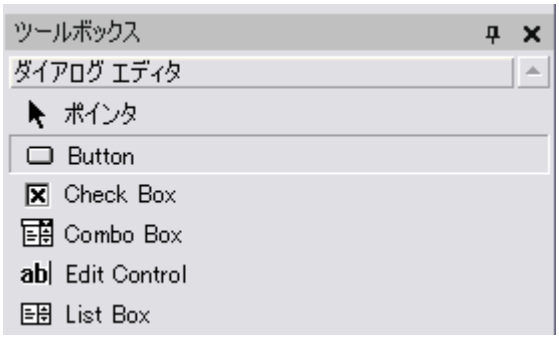
ここまでで、何も処理しないダイアログベースのアプリケーションが生成されています。
ビルドして、実行させることが出来るはずです。
実行すると、OK ボタンとキャンセルボタンを持ったダイアログが表示されます。
ダイアログの右上には、ウィンドウズに共通の×ボタンもついています。

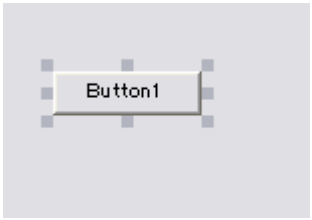
５．コントロールの配置

まっさらのダイアログでは面白くないので、次は自分でボタンを置いてみましょう。

ボタンやエディットボックス等、ユーザがダイアログを操るために使用する部品を『コントロール』と言います。
ダイアログへのコントロールの配置は、ダイアログエディタを使用します。
恐らく、プロジェクトを作成したままのときはダイアログエディタが表示されていますが、一応その表示方法から書いていきます。まず、ボタンを置いてみましょう。

1）メニューの[表示]-[リソースビュー]を選択し、リソースビューを表示させます。
2）リソースビューからプロジェクトのリソースを選択し、[Dialog]から編集したいダイアログリソースを選択します（IDD_プロジェクト名_DIALOG になっているかと思います)。
3）メニューの[表示]-[ツールボックス]を選択し、ツールボックスを表示させます。
4）ツールボックスの[ダイアログエディタ]を開き、配置したいコントロールを選択します。





5）ダイアログエディタ上のダイアログをクリックし、コントロールを配置します。

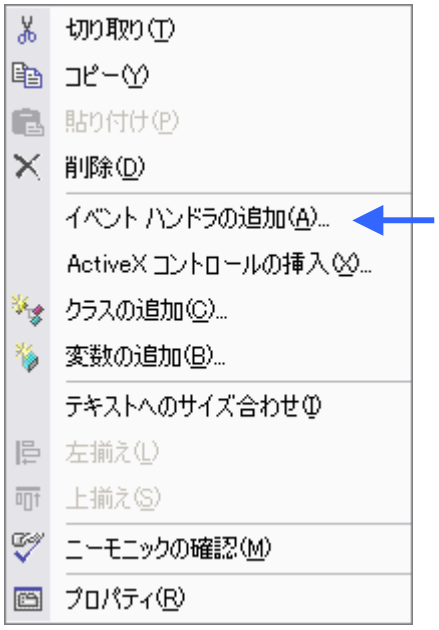
これで配置は終了です。ビルドして実行すれば、配置したコントロールがちゃんと表示されています。

６．イベントハンドラ

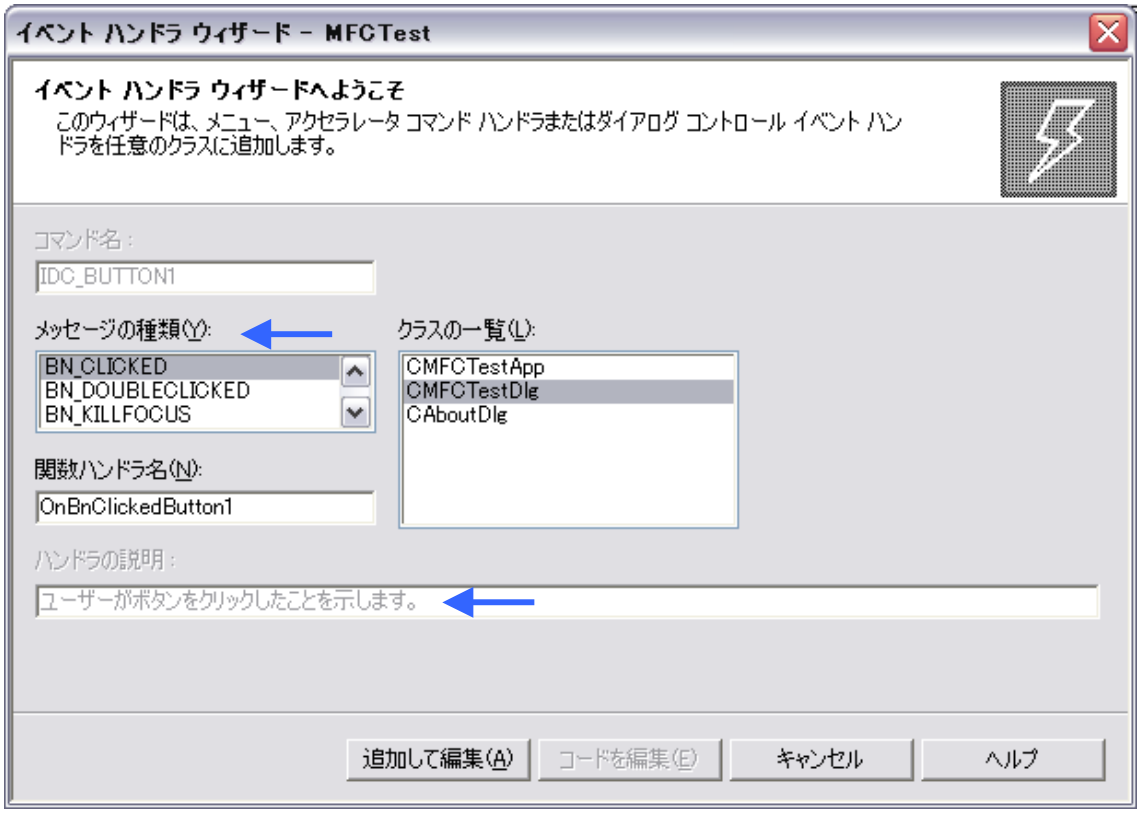
コントロールの配置は出来ましたが、このままではボタンを押しても何も起こりません。
何も書いていないので当然です。

では、ボタンが押されたときに、何かメッセージを出してみましょう。

１）　ダイアログエディタの配置したボタン上で右クリックし、[イベントハンドラの追加]を選択します。



2）　イベントハンドラウィザードダイアログボックスが表示されます。「メッセージの種類」リストボックスでは、どのイベントへのハンドリングかを選択します。ここではボタンのクリックイベントに対する処理を作成したいので、BN_CLICKED を選択します。ちなみに他にも色々種類がありますが、リストボックスの選択を変更するごとに一番下の説明文も変化します。



3）　「追加して編集」ボタンを押下すると、イベントハンドラ関数が作成されます。

```
void CMFCTestDlg::OnBnClickedButton1()
{
	// TODO : ここにコントロール通知ハンドラ コードを追加します。
}
```

さて、これはどうやら何かのクラスのメンバ関数として宣言されているようです。このクラスは、プロジェクトのはじめに AppWizard によって作られた、このアプリケーションのダイアログ用のクラスです。ダイアログのボタンが押されると、押されたよ、というメッセージがこのダイアログクラスに送られます。従って、ボタン押下のイベントハンドラを作ると、ダイアログクラスにその関数が作成されます。

４）　メッセージボックスで何かを表示してみましょう。イベントハンドラの内容を次のように書き換えてください。

```
void CMFCTestDlg::OnBnClickedButton1()
{
	AfxMessageBox("ボタンが押されたよ");
}
```

※　AfxMessageBox()関数は、メッセージボックスを表示するための MFC のグローバル関数です。このように書くと OK ボタンをもったメッセージボックスを表示してくれます。引数を指定することにより、「Yes」「No」のボタンや、「はい」「いいえ」、「OK」「キャンセル」「ヘルプ」などのボタンも自由に表示させることが出来ます。

５）　ビルドし、実行すると、ボタンを押すたびにメッセージボックスが表示されるようになっているはずです。

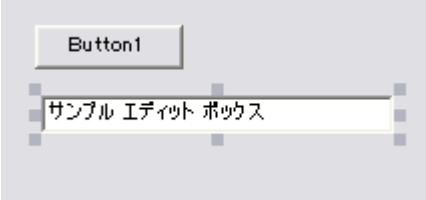
このように、Windows アプリケーションの開発では、「イベントが起きた」→「処理をする」という形式でコーディングしていきます。このようなコーディング形式の概念を『イベントドリブン(イベント駆動)』とよびます。これに対し、ユーザに逐次入力を求めるプログラムを『逐次実行方プログラム』などと呼んだりします。

※　実際には、Windows では「イベントが起きた」→「OS（Windows）がアプリケーションに対し”イベントが起きたよ”というメッセージを送る」→「メッセージを受け取り処理する」という流れになっています。Windows アプリケーションは、常に OS からのメッセージを待っており、そのメッセージにより様々な処理を行います。メッセージの種類は「マウスが動いた」の場合もありますし、「時間がたった」の場合もあります。アプリケーションは様々なメッセージの中から、自分の処理に必要なメッセージのみを受け取り、処理します。このため、Windows のイベントハンドラは「メッセージハンドラ」とも呼ばれます。

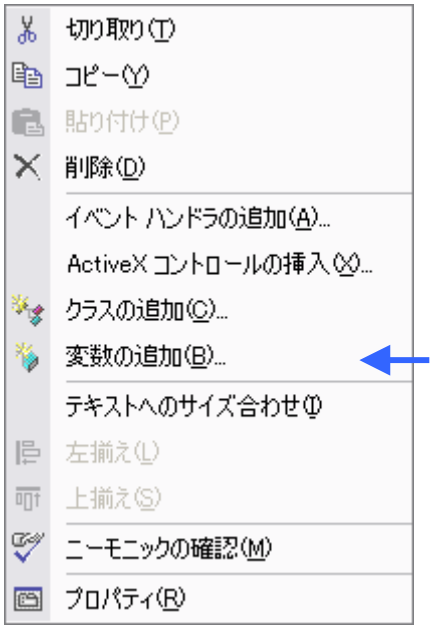
７．コントロール変数

もう一つ進んで、こんどはエディットボックスを追加し、その内容をメッセージボックスで表示してみましょう。ダイアログ上のコントロールの内容を取得したり変更したりするには、ダイアログのクラスに、そのコントロールを操るメンバ変数を定義する必要があります。

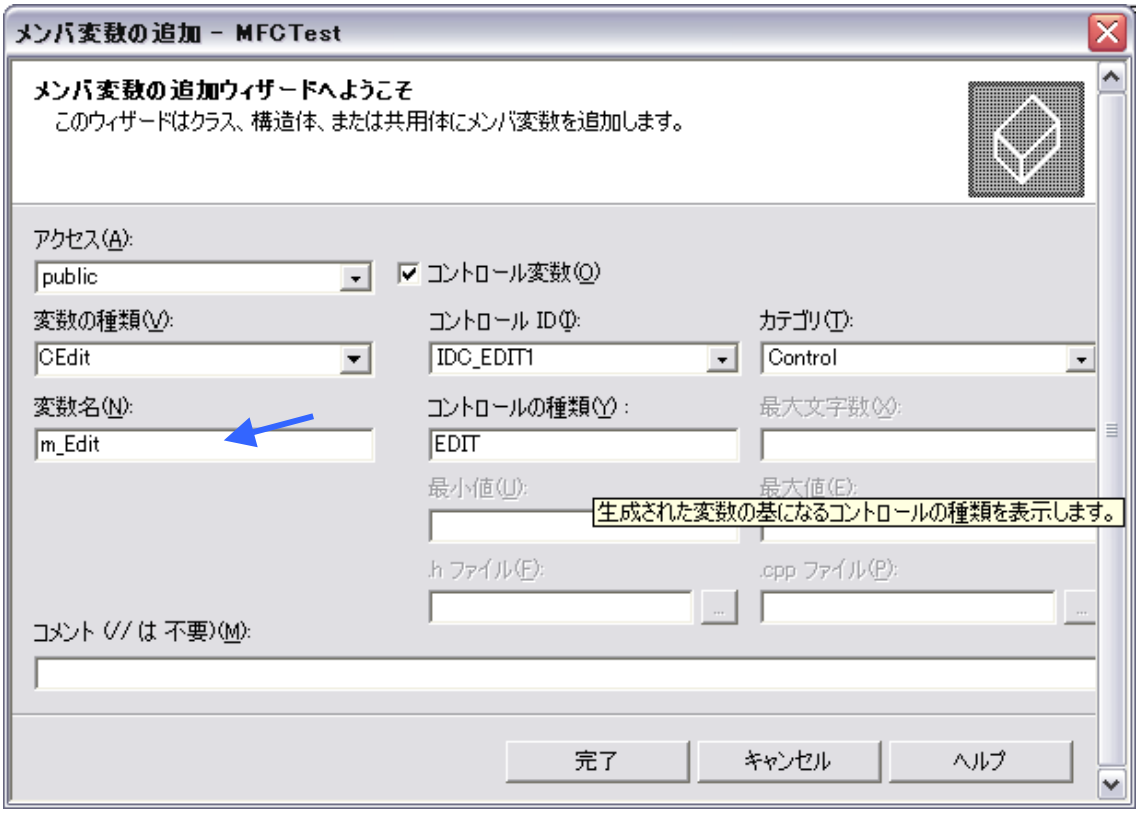
１）ダイアログエディタでエディットボックスをダイアログに配置します。



２）配置したエディットボックス上で右クリックし、「変数の追加」を選択します。



3）「メンバ変数の追加」ダイアログボックスが表示されるので、「変数名」に変数の名前を入力します。例では「m_Edit」としています。



4）完了ボタンを押下すると、ダイアログクラスに m_Edit メンバ変数が追加されます。この変数はダイアログ上のエディットボックスと関連づいています。

※　どのようにして関連づいているのでしょうか？答えは、ダイアログクラスの DoDataExchange()メンバ関数に記述されています。メンバ変数を追加すると、ここに DDX_Control(pDX, IDC_EDIT1, m_Edit);という一文が追加されます。この関数はダイアログ生成時に呼ばれ、リソース上のエディットボックス IDC_EDIT1 とメンバ変数の m_Edit を関連付けますが、ここで重要なことがあります。**ダイアログエディタ上でエディットボックスを削除しても、（この関数に追加された一文を含む）メンバ変数追加にて追加されたコード部分は削除されません。**ダイアログエディタ上でこのエディットボックスを削除し、この一文を残したままにしておくと、ダイアログが表示された瞬間にメモリアクセスエラーが起きてアプリケーションは終了してしまいます。メンバ変数を作ったコントロールを削除するときは、メンバ変数の定義と、DoDataExchange()メンバ関数内の一文の削除を忘れないように注意しましょう。

５）先ほど作成したイベントハンドラを表示させ、次のように変更してください。

```
void CMFCTestDlg::OnBnClickedButton1()
{
	CString str;
	m_Edit.GetWindowText(str);
	AfxMessageBox(str);
}
```

※　ここで、CString という新しいクラスが出てきました。このクラスは、STL の string クラスのように、可変長文字列を操る MFC のクラスです。このクラスは、文字列の代入、比較、抽出等、文字列の操作が容易に出来るようになっています。例えば、次のように書くことが出来ます。

```
CString str1 = “モモンガ”;
CString str2 = “モモンガ”;
CString str3 = “アルマジロ”;
if(str1 == str2){
	printf(“%s と%s は同じ内容です¥n”, str1, str2);
}
str += str3;
printf(“%s などという生き物はオランよ”);
```

直接の代入が可能

内容の比較が簡単

const char* として扱える（Unicode プロジェクトは除く）

文字列同士の演算が簡単

== 実行結果 ==

モモンガとモモンガは同じ内容です。
モモンガアルマジロなどという生き物はオランよ

この CString のように、MFC には知っていると非常に便利な**単純値型クラス**が存在します。点をあらわす CPoint、矩形をあらわす CRect、縦横のサイズをあらわす CSize などがよく使われます。

6）ビルドして実行すると、ボタンを押すたびにエディットボックスの内容がメッセージボックスに表示されるようになっているはずです。

イベントハンドラで使用したGetWindowText()という関数ですが、これはCEditクラスのメンバ関数です。m_Editというメンバ変数を追加したのですから、ダイアログクラスのクラス宣言にはメンバ変数の定義があるはずです。そこを見てみましょう。
ダイアログクラスのヘッダは、「プロジェクト名 Dlg.h」という名前で存在しているはずです。
そこには、次のような定義がしてあります。

```
CEdit m_Edit;
```

コントロールのメンバ変数は、MFCのコントロールクラスのインスタンスとして宣言されます。MFCのコントロールクラスには、コントロールの操作に必要なメンバ関数が多数存在しています。ためしに、「CEdit」の部分にカーソルを当ててF1キーを押し、ヘルプの内容を見てみましょう。たくさんのメンバ関数があると思います。エディットコントロールに対する何かの操作をしたいときは、この中から機能を探せばよい、ということになります。

しかし、この中にはGetWindowText()関数は存在しません。ヘルプの一番上のほうを見てみましょう。『基本クラスのメンバ』という項目があり、その下に
CObject のメンバ
CCmdTarget のメンバ
CWnd のメンバ
と3つ項目が並んでいると思います。
これが何を表しているかというと、CEditクラスは「CObject」「CCmdTarget」「CWnd」という3つのクラスを親に持つクラスである、ということです。
実は、全てのコントロールクラスは「CWnd」クラスの子クラスであり、「CWnd」で定義されているメンバ変数全てを使用できます。GetWindowText()関数はこの「CWnd」のメンバ関数です。「CWnd」のメンバ関数の量は非常に多く、全て覚えることは難しいでしょう。コントロールの関数を探す方法としては、まずコントロールクラスのヘルプを見て、なかったらCWndのヘルプに移る、という方法がよいと思います。そのうち、勘で大体分かるようになってきます。

※ GetWindowText()関数に限ったことではありませんが、子クラスによってその関数の結果は変わってきます。今回CEditではエディットボックスに入力されている文字が返ってきましたが、ボタンのコントロールクラスである「CButton」で呼ぶと、ボタンに表示されている文字列（「OK」とか「キャンセル」とか）が返ってきます。