ハム太と作るC++ライブラリ

-コントロール編-

haseham

はじめに

・まだ執筆中なんですが、例によって
公開しながらの加筆、という形をとっています。

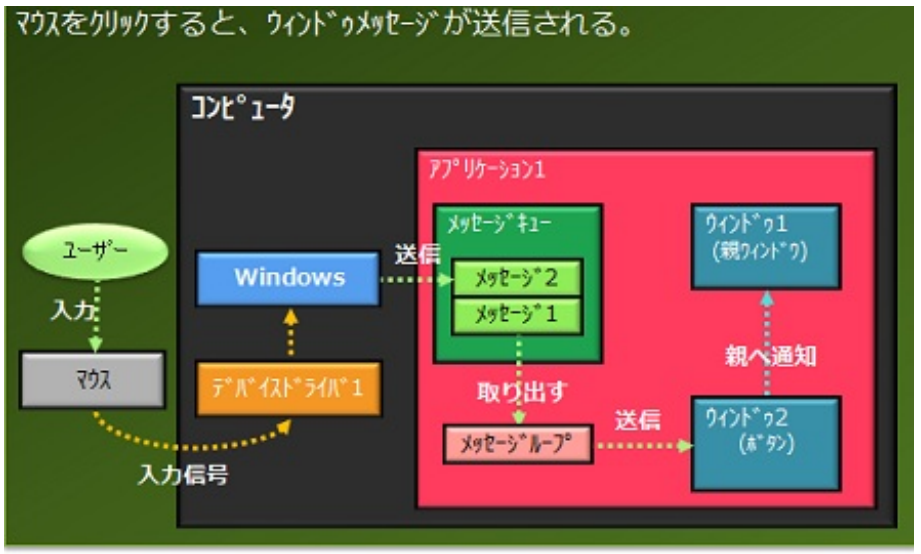
・「構文編」から読まれている皆さんは、
他ｻｲﾄなどをご覧になって、独習を進めておられると思いますが、
本書では、その中でも、特に詰まりそうな箇所について
優先的に書き進めております。

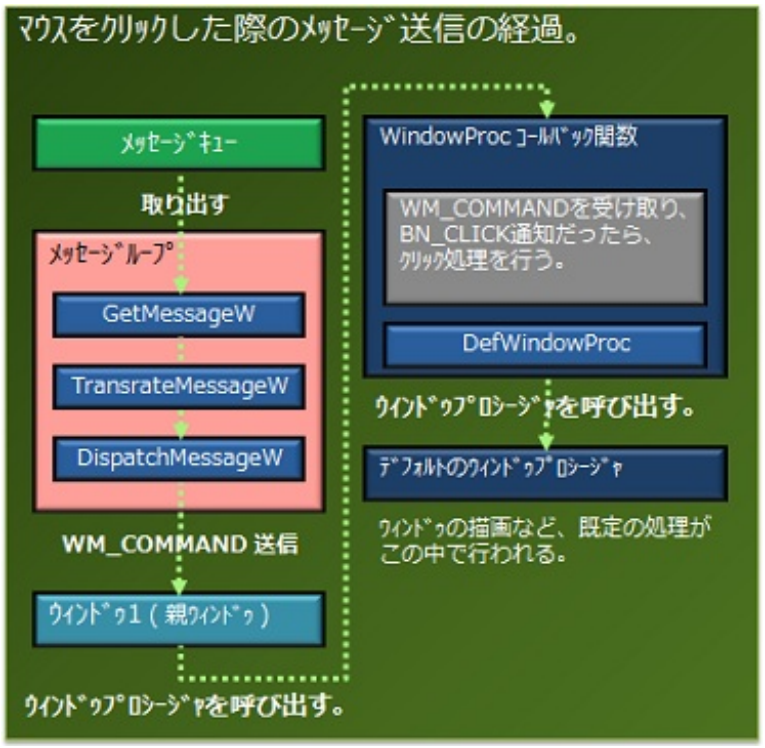
・ある程度理解できた段階でﾊﾏる「ｻﾌﾞｸﾗｽ化」についても、
ｸﾞｸﾞってても理解できなかった方でもわかるように
わかりやすいｻﾝﾌﾟﾙｺｰﾄﾞを載せています。

・画面描画については、そのうち書きますが、
msdnでも解説されていますので、そちらをご覧ください。↓

http://msdn.microsoft.com/ja-jp/library/windows/desktop/ff381399(v=vs.85).aspx

DefWindowProc関数 (ﾒｯｾｰｼﾞﾙｰﾌﾟとｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬ)





・さて、あなたのつくえの上には、ﾊﾟｿｺﾝがあって、
画面には、**Windows**のﾃﾞｽｸﾄｯﾌﾟ画面や
ｳｨﾝﾄﾞｳが表示されているとします。

・その手前に置かれたｲｽには、
あなた(または別のﾕｰｻﾞｰ)が座っていて、ﾏｳｽを動かしている。

・そしてたまに、ｷｰﾎﾞｰﾄﾞに手をうつして、ｷｰをｶﾀｶﾀ打っている。

・これをもう少しｺﾝﾋﾟｭｰﾀ寄りの視点でまとめると、
おおよそ、次のような流れになります。↓

---

1. ﾕｰｻﾞｰが入力した情報は、入力機器から**Windows**に送られる。

[ﾕｰｻﾞｰ] -(入力)→ [ｷｰﾎﾞｰﾄﾞ orﾏｳｽ] -(入力信号)→ **[Windows]**

---

2. Windowsは、ｱﾌﾟﾘｹｰｼｮﾝﾌﾟﾛｸﾞﾗﾑの「ﾒｯｾｰｼﾞｷｭｰ」に、
ｳｨﾝﾄﾞｳﾒｯｾｰｼﾞを送信する。

[Windows] - (ｳｨﾝﾄﾞｳﾒｯｾｰｼﾞ) → **[** ﾒｯｾｰｼﾞｷｭｰ **]**

---

3. ｱﾌﾟﾘｹｰｼｮﾝ (あなたの作ったﾌﾟﾛｸﾞﾗﾑ) は、
GetMessage関数を、定期的に呼び出して、
「ﾒｯｾｰｼﾞｷｭｰ」に詰め込まれた「ｳｨﾝﾄﾞｳﾒｯｾｰｼﾞ」を
1つずつ取り出しては、それに応じた処理を行います。

・このﾒｯｾｰｼﾞの取り出しは、通常、
「ﾒｯｾｰｼﾞﾙｰﾌﾟ」といわれる無限ﾙｰﾌﾟの中で行います。↓

```
// 「ﾌﾟﾛｼﾞｪｸﾄ」を作成した時に、「Windowｱﾌﾟﾘｹｰｼｮﾝ」を選択した場合は、
// _WinMain関数の最後に、下記のような「ﾒｯｾｰｼﾞﾙｰﾌﾟ」が書かれている。↓

MSG msg; // ｳｨﾝﾄﾞｳﾒｯｾｰｼﾞを受け取るMSG構造体。

while ( ::GetMessageW( &msg, NULL, 0, 0 ) > 0 ) // ｷｭｰがｶﾗなら0、ｴﾗｰ時は-1を返す。
{
   ::TranslateMessage( &msg ); // 文字ｷｰの入力なら、「WM_CHAR」を送信する。
   ::DispatchMessageW( &msg ); // 送信先ｳｨﾝﾄﾞｳの「ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬ」を呼ぶ。
}

return msg.wParam; // ﾌﾟﾛｸﾞﾗﾑを終了する。
```

・**GetMessage関数**に渡した**MSG**構造体には、
送信先のｳｨﾝﾄﾞｳを示す「ﾊﾝﾄﾞﾙ」、(HWND型)
ﾒｯｾｰｼﾞの種類を示す「ﾒｯｾｰｼﾞｺｰﾄﾞ」、
ｸﾘｯｸされた位置などの「ﾊﾟﾗﾒｰﾀ」、(ﾒｯｾｰｼﾞにより異なる)
が格納されます。

・これを、**DispatchMessage関数**に渡すと、
「ｳｨﾝﾄﾞｳｸﾗｽ」を登録した時に指定した「ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬ」が呼ばれます。

・たとえば、ﾎﾞﾀﾝがｸﾘｯｸされた時に
ｳｨﾝﾄﾞｳの上に絵を表示させたりする場合は、
ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬの中に、その処理を書いておきます。

・このように、ﾊﾟｿｺﾝ上でのあらゆる操作は、
Windowsを介して行われます。

・ｱﾌﾟﾘｹｰｼｮﾝﾌﾟﾛｸﾞﾗﾑは、
WindowsAPIの関数を呼び出すことで
ｺﾝﾋﾟｭｰﾀｰを間接的に操作します。

```
// ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬ ↓

LRESULT CALLBACK WindowProc(
                            HWND h_wnd_, // 送信先ｳｨﾝﾄﾞｳのﾊﾝﾄﾞﾙ
                            UINT   msg_,    // ﾒｯｾｰｼﾞｺｰﾄﾞ
                            WPARAM wp_, // Wﾊﾟﾗﾒｰﾀｰ
                            LPARAM  lp_    // Lﾊﾟﾗﾒｰﾀｰ
                           )
{

 switch ( msg_ ) // ﾒｯｾｰｼﾞｺｰﾄﾞを判定していく。
 {

  // ------------
  case WM_LBOTTONDOWN: // 左ｸﾘｯｸなら、

   // ここに、 左ｸﾘｯｸされた時の処理を書く。
   break;
  // ------------

 } // end switch

 ::DefWindowProcW( h_wnd_, msg_, wp_, lp_ ); // 親ｳｨﾝﾄﾞｳに「通知ｺｰﾄﾞ」を送信する。

} // end proc
```

・ﾒｯｾｰｼﾞ処理は、体感的には、
入力されたと同時に実行されているかのように見えますが、
厳密には、そうではありません。

・「ﾒｯｾｰｼﾞﾙｰﾌﾟ」は、ものすごい速度で回っているので気付かないんですが、
「ﾒｯｾｰｼﾞｷｭｰ」の中では、多数のﾒｯｾｰｼﾞが、常に順番待ちをしていますので、
送信されたからといって、すぐに取り出して処理することはできません。

・ｳｨﾝﾄﾞｳﾒｯｾｰｼﾞは、通常、**Windows**が送信するものですが、
ｱﾌﾟﾘｹｰｼｮﾝから送信することもできます。↓

```
// ﾒｯｾｰｼﾞｷｭｰにﾒｯｾｰｼﾞを送信したら、すぐに制御を戻す。↓
PostMessageW( h_wnd_, msg_, wparam_, lparam_ );
```

```
// ﾒｯｾｰｼﾞｷｭｰに送信したﾒｯｾｰｼﾞが処理されるまで制御を戻さない。↓
SendMessageW( h_wnd_, msg_, wparam_, lparam_ );
```

・「ﾒｯｾｰｼﾞｷｭｰ」は、「ｽﾚｯﾄﾞ」ごとに持つことができます。

・「ｽﾚｯﾄﾞ」というのは、同じﾌﾟﾛｾｽ内で、
さらに複数の処理を、同時に実行するための
より細かな実行単位です。(ｽﾚｯﾄﾞについては、詳細は後述します)

・上の図では、「ﾌﾟﾛｾｽは、同時には動いていない」と書きましたが、
最近は、演算装置(ｺｱ)が複数付いた「ﾏﾙﾁｺｱﾌﾟﾛｾｯｻ」ですから、
同時に動いています。

・ボタンなどのコントロールは、通常、ウィンドゥの上に表示されますが、
これは、ボタンが、ウィンドゥの子ウィンドゥとして登録されているからです。

・子ウィンドゥでは、クリックなどのイベントが発生すると、
ウィンドゥメッセージ (**WM_**ほにゃらら) が送信されず、
親ウィンドゥに向けて、**WM_COMMAND**が送信されます。

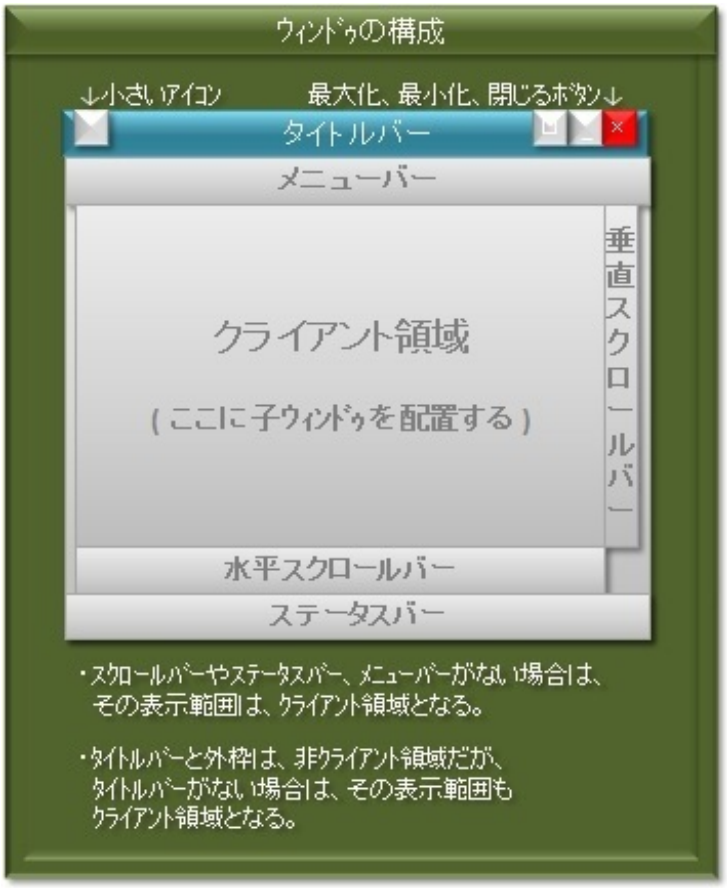
(※ **SendMessage**関数などで送信した場合は、送信される。)

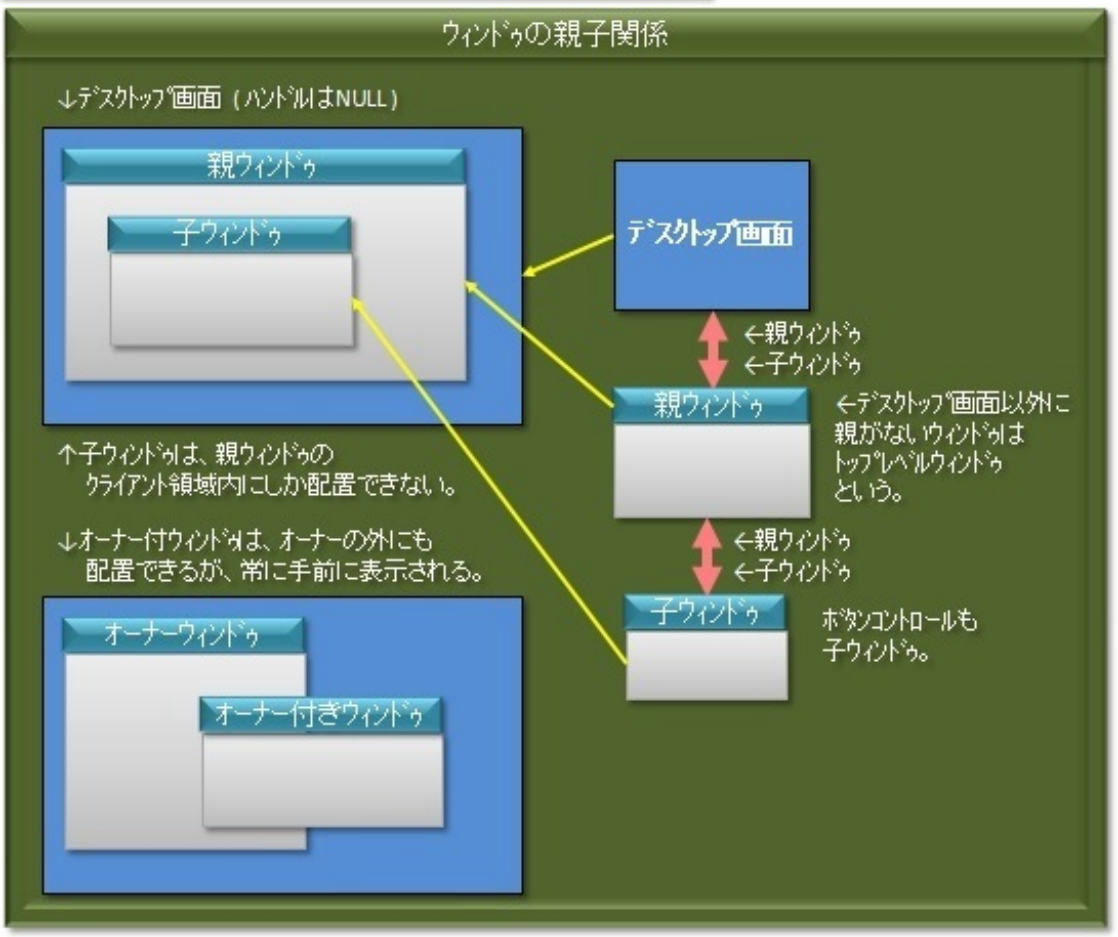
・そんなわけで、子ウィンドゥのイベント処理は、
親ウィンドゥ側のウィンドゥプロシージャの中に
書いていくことになるわけです。

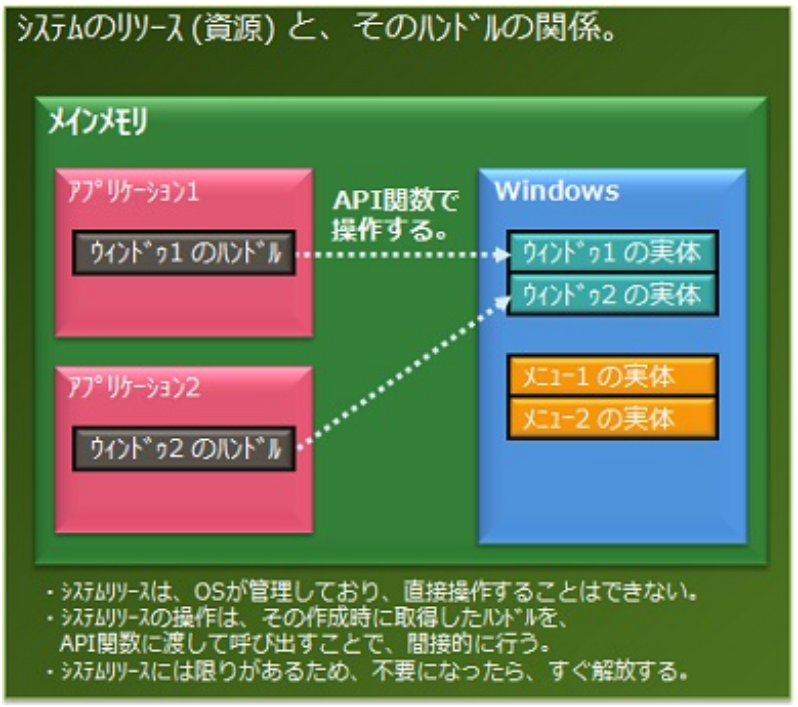
・**WM_COMMAND**のパラメータには、通知の種類や、
送信者である子ウィンドゥの識別**ID**などが格納されています。

・コモンコントロールの場合は、**WM_COMMAND**ではなく、
WM_NOTIFYが送信されます。

CreateWindow関数 (ｳｨﾝﾄﾞｳを作成する)







・「ｳｨﾝﾄﾞｳ」を、画面上に表示させるには、
次のような手順が必要です。 ↓

---

1. 「ｳｨﾝﾄﾞｳｸﾗｽ」を登録する。(※いわゆる「ｸﾗｽ」とは別物)
2. それを指定して、「ｳｨﾝﾄﾞｳ」を作成する。
3. その「ｳｨﾝﾄﾞｳ」を表示させる。

・「ﾌﾟﾛｼﾞｪｸﾄ」を作成した時に、
「**Window**ｱﾌﾟﾘｹｰｼｮﾝ」を選択した場合は、
ﾒｲﾝとなるｳｨﾝﾄﾞｳを作成する処理がすでに書いてあるはずですが、
ここではｸﾗｽ化した場合のｻﾝﾌﾟﾙｺｰﾄﾞで説明していきます。 ↓

```
BOOL CForm::Init(
                HWND h_parent_,          // 親ｳｨﾝﾄﾞｳのﾊﾝﾄﾞﾙ
                const wchar_t* p_name_, // ｳｨﾝﾄﾞｳｸﾗｽの名前
                const wchar_t* p_text_,   // ﾀｲﾄﾙﾊﾞｰに表示される文字列
                int x_, int y_,          // ｳｨﾝﾄﾞｳの表示位置 (左上端の座標)
                int width_, int height_ // ｳｨﾝﾄﾞｳのｻｲｽﾞ
```

```
                    )
{

// ----------------------
// ｳｨﾝﾄﾞｳｸﾗｽのｽﾀｲﾙを決めておく。

DWORD c_style = CS_BYTEALIGNWINDOW | CS_BYTEALIGNCLIENT |
// 速度的に有利 (ｸﾗｲｱﾝﾄ領域のX座標が8の倍数になるように配置 )

// CS_CLASSDC | // このｸﾗｽに属する すべてのｳｨﾝﾄﾞｳで、 DCを共有する。
CS_OWNDC | // このｸﾗｽに属するｳｨﾝﾄﾞｳには、 それぞれ独自のDCを割り当てる。
// CS_PARENTDC | // 子ｳｨﾝﾄﾞｳの場合は、 これが必要。
// (子ｳｨﾝﾄﾞｳのｸﾘｯﾋﾟﾝｸﾞ領域に、 親ｳｨﾝﾄﾞｳのｸﾘｯﾋﾟﾝｸﾞ領域を設定)
// CS_HREDRAW | CS_VREDRAW | // ｳｨﾝﾄﾞｳｻｲｽﾞが変化した際に、 画面を更新する。
// ( ※ちらつきが発生するので使用しない。
//     代わりに、 WM_SIZEではInvalidateRect関数を呼んでいる )
CS_DBLCLKS; // Wｸﾘｯｸ時にもﾒｯｾｰｼﾞを発生させる。
// CS_NOCLOSE | //「閉じる」 を無効にします。
//CS_SAVEBITS // ﾒﾆｭｰやﾀﾞｲｱﾛｸﾞなどで使用、 無くてもいい。

// ----------------------
// ｳｨﾝﾄﾞｳｸﾗｽを登録する。

WNDCLASSEXW wc = {}; // ｳｨﾝﾄﾞｳｸﾗｽ構造体

wc.hInstance = p_appli->GetInstance(); // ｱﾌﾟﾘｹｰｼｮﾝｲﾝｽﾀﾝｽのﾊﾝﾄﾞﾗ HINSTANCE
// これは、 _tWinMain関数の第一引数のhInstance。 関数から取得することもできる。 ↑
wc.lpszClassName = p_name_; // ｳｨﾝﾄﾞｳｸﾗｽ名。
wc.lpfnWndProc = WindowProc_Init; // ★ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬ。 (初期化用を格納しておく)
wc.style = c_style; // ｳｨﾝﾄﾞｳｸﾗｽのｽﾀｲﾙ。 ( 先ほどﾌﾗｸﾞ定数を詰め込んだ整数値 )
wc.cbSize = sizeof(WNDCLASSEXW); // この構造体のｻｲｽﾞ。 (※必須)

wc.hIcon     = NULL;  // 大きめのｱｲｺﾝのﾊﾝﾄﾞﾙ。
wc.hIconSm = NULL;  // 小さめｱｲｺﾝのﾊﾝﾄﾞﾙ。 (ﾀｲﾄﾙﾊﾞｰ)
wc.hCursor  = NULL;  // ﾏｳｽﾎﾟｲﾝﾀのﾊﾝﾄﾞﾙ。
wc.lpszMenuName = NULL; // ﾒﾆｭｰを付ける場合は、その名前。
```

```
wc.cbClsExtra   = 0; // ｳｨﾝﾄﾞｳｸﾗｽごとに固有のﾃﾞｰﾀを格納できる。
wc.cbWndExtra = 0; // ｳｨﾝﾄﾞｳごとに固有のﾃﾞｰﾀを格納できる。
wc.hbrBackground = (HBRUSH) COLOR_APPWORKSPACE + 1; // 作業領域の背景色。

// ｳｨﾝﾄﾞｳｸﾗｽを登録する。
if ( ::RegisterClassExW( &wc ) == 0 ) return FALSE;

// ------------------------
// ｳｨﾝﾄﾞｳｽﾀｲﾙを決めておく。 ( 後で再設定できるので、適当でいい )

DWORD w_ex_style = WS_EX_CONTROLPARENT | // ﾀﾌﾞ移動を有効にする。
// WS_EX_APPWINDOW | // 一番上にあるｳｨﾝﾄﾞｳを表示するときに、強制的にﾀｽｸﾊﾞｰに含みます。
// WS_EX_ACCEPTFILES | // ｳｨﾝﾄﾞｳにﾌｧｲﾙをﾄﾞﾛｯﾌﾟした際、WM_DROPFILESが送られる。
// WS_EX_LAYERED | // 半透明になる。( 不透明度は、SetLayeredWindowAttributes関数で指定 )
// WS_EX_TRANSPARENT | // 透明になる。 ( 下のｳｨﾝﾄﾞｳが更新された時にだけ、WM_PAINTを受信 )
WS_EX_TOPMOST | // 常に最前面に表示される。( SetWindowPos関数で変更する)
// WS_EX_TOOLWINDOW | // [X]ﾎﾞﾀﾝだけ表示。
// ( hwnd引数にNULLを指定すると、ﾀｽｸﾊﾞｰに表示されないﾄｯﾌﾟﾚﾍﾞﾙｳｨﾝﾄﾞｳ扱いになる。
//   一方、 hwnd引数に親のhwndを指定すると、 親ｳｨﾝﾄﾞｳと連動して最小化するｳｨﾝﾄﾞｳになる。
//   hwnd引数に親のhwndを指定して、さらにWS_CHILDとｳｨﾝﾄﾞｳIDを指定した場合は、
//   親ｳｨﾝﾄﾞｳのｸﾗｲｱﾝﾄ領域内だけで動作する子ｳｨﾝﾄﾞｳになる。 )
// WS_EX_CONTEXTHELP | // ﾀｲﾄﾙﾊﾞｰに[?]ﾎﾞﾀﾝ を表示。 ( 子ｳｨﾝﾄﾞｳが WM_HELPを受け取る )

// WS_EX_MDICHILD // MDI 子ｳｨﾝﾄﾞｳを作成する。
// WS_EX_NOPARENTNOTIFY // 子ｳｨﾝﾄﾞｳの場合、作成･破棄時に
                        // 親ｳｨﾝﾄﾞｳに WM_PARENTNOTIFYを送らない。
WS_EX_COMPOSITED; // ｳｨﾝﾄﾞｳへの描画が、自動的にﾀﾞﾌﾞﾙﾊﾞｯﾌｧﾘﾝｸﾞされ、ちらつきがなくなる。

DWORD w_style = WS_OVERLAPPED | // ｵｰﾊﾞｰﾗｯﾌﾟｳｨﾝﾄﾞｳを作成する。
// ( ﾀｲﾄﾙﾊﾞｰと外枠があり､親ｳｨﾝﾄﾞｳの外へ移動できる。いわゆる「ｳｨﾝﾄﾞｳ」のこと。 )
WS_CAPTION | // ﾀｲﾄﾙﾊﾞｰを持つｳｨﾝﾄﾞｳを作成する。
WS_SYSMENU|// ﾀｲﾄﾙﾊﾞｰに、最大化、最小化、閉じるなどのﾎﾞﾀﾝを配置する。
WS_HSCROLL | WS_VSCROLL | // 縦･横のｽｸﾛｰﾙﾊﾞｰを追加する。
WS_THICKFRAME |   // ｻｲｽﾞ変更を許可する。 ( ﾘｻｲｽﾞ用の太めの枠が付く )
WS_MINIMIZEBOX |  // 最小化ﾎﾞﾀﾝの使用を許可する。
```

```
WS_MAXIMIZEBOX | // 最大化ﾎﾞﾀﾝの使用を許可する。
WS_VISIBLE ; // 表示する。

// WS_DISABLED | // ｳｨﾝﾄﾞｳを無効にする。
// WS_MAXIMIZE | WS_MINIMIZE; // 最大or最小表示する。
// WS_POPUP // ﾎﾟｯﾌﾟｱｯﾌﾟ。( ※WS_CHILD では使えない。ﾎﾟｯﾌﾟｱｯﾌﾟの親はﾃﾞｽｸﾄｯﾌﾟ画面 )
// WS_TABSTOP // ﾀﾌﾞｽﾄｯﾌﾟを許可する。 ( 子ｺﾝﾄﾛｰﾙ向け )
// WS_DLGFRAME // 二重境界を持ち、ﾀｲﾄﾙ無し。
// WS_GROUP // ｺﾝﾄﾛｰﾙｸﾞﾙｰﾌﾟの最初のｺﾝﾄﾛｰﾙ。(方向ｷｰで移動できる)
            // 次に指定したｺﾝﾄﾛｰﾙまでが同じｸﾞﾙｰﾌﾟとなる。

// ------------------------
// ｳｨﾝﾄﾞｳを作成する。

h_window = ::CreateWindowExW(
                    w_ex_style, // 拡張ｳｨﾝﾄﾞｳｽﾀｲﾙを指定 ( これは0でもいい )
                    p_name_, // ｳｨﾝﾄﾞｳｸﾗｽ名
                    p_text_, // ﾀｲﾄﾙﾊﾞｰに表示される文字列
                    w_style, // ｳｨﾝﾄﾞｳｽﾀｲﾙ
                    x_, // ｳｨﾝﾄﾞｳの左端 (x座標)
                    y_, // ｳｨﾝﾄﾞｳの上端 (y座標)
                    width_, // ｳｨﾝﾄﾞｳの横幅
                    height_, // ｳｨﾝﾄﾞｳの縦幅
                    h_parent_, // 親ｳｨﾝﾄﾞｳのﾊﾝﾄﾞﾙ HWND
                NULL, // ﾒﾆｭｰのﾊﾝﾄﾞﾙ HMENU
                    wc.hInstance , // ｱﾌﾟﾘｹｰｼｮﾝｲﾝｽﾀﾝｽのﾊﾝﾄﾞﾗ HINSTANCE
                NULL ); // CREATESTRUCT構造体へのﾎﾟｲﾝﾀ。( MDIｸﾗｲｱﾝﾄ作成時には必須 )

if ( h_window == NULL ) // 作成に失敗したら、
{
  ::UnregisterClassW( p_name_, wc.hInstance ); // ｳｨﾝﾄﾞｳｸﾗｽを抹消する。

  return FALSE;
}

// ------------------------
// ★thisﾎﾟｲﾝﾀを、格納しておく。( ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬから、ﾒｯｾｰｼﾞﾊﾝﾄﾞﾗを呼ぶのに使う )
```

```
::SetWindowLongPtrW( h_window, GWLP_USERDATA, (LONG) this );
// ----------------------
// ★ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬを、本番用にすり替える。

::SetWindowLongPtrW( h_window, GWLP_WNDPROC, (LONG) WindowProc );
// ----------------------
::UpdateWindow( h_window ); // ｳｨﾝﾄﾞｳを更新する。
::ShowWindow( h_window, SW_SHOWNORMAL ); // ｳｨﾝﾄﾞｳを表示する。
// ----------------------

return TRUE; // 作成に成功した。

}
```

```
// 初期化時にだけ使用するｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬ。

LRESULT CALLBACK WindowProc_Init(
    HWND h_wnd_, UINT msg_, WPARAM wparam_, LPARAM lparam_ )
{

  // -----------------
  // ｳｨﾝﾄﾞｳが生成されたら、( CreateWindow関数が呼ばれたら )

  if ( msg_ == WM_CREATE )
  {
    // その他のﾒｯｾｰｼﾞは、ｼｽﾃﾑに処理してもらう。( 親への通知ｺｰﾄﾞ送信や標準描画処理 )
     ::DefWindowProcW( h_wnd_, msg_, wparam_, lparam_ );
     return 0; // 0を返すと、CreateWindow関数が成功する。
   }

   // -----------------
   // その他のﾒｯｾｰｼﾞは、ｼｽﾃﾑに処理してもらう。( 親への通知ｺｰﾄﾞ送信や標準描画処理 )
   return ::DefWindowProcW( h_wnd_, msg_, wparam_, lparam_ );
   // -----------------
}
```

```
// ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬ ( すべてのｳｨﾝﾄﾞｳｲﾝｽﾀﾝｽで共有する )

LRESULT CALLBACK WindowProc(
      HWND h_wnd_, UINT msg_, WPARAM wparam_, LPARAM lparam_ )
{

  // -----------------
  // 送信先ｳｨﾝﾄﾞｳのﾊﾝﾄﾞﾙから、ｺﾝﾄﾛｰﾙｲﾝｽﾀﾝｽへのﾎﾟｲﾝﾀを取得する。

  CControl* p_control = (CControl*) ::GetWindowLongPtrW(
                                          h_wnd_, GWLP_USERDATA );
  // -----------------
  // ｲﾝｽﾀﾝｽごとのﾒｯｾｰｼﾞ処理を呼び出す。

  // 送信先ｲﾝｽﾀﾝｽのﾊﾝﾄﾞﾗを呼ぶ。( 処理した場合は、0を返す。 )
  return p_control->MessageHandler( h_wnd_, msg_, wparam_, lparam_ );

// -----------------
}
```

・通常は、ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬの中に、処理を書いていくんですが、
ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬはｸﾞﾛｰﾊﾞﾙ関数でないといけないので、
ｳｨﾝﾄﾞｳをｸﾗｽ化した場合に、ﾒｿｯﾄﾞを指定することができません。

・しかし、ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬには、
送信先ｳｨﾝﾄﾞｳのﾊﾝﾄﾞﾙが渡されますので、
これを使って、ｳｨﾝﾄﾞｳｲﾝｽﾀﾝｽのthisﾎﾟｲﾝﾀを取り出すことができ、
ｲﾝｽﾀﾝｽごとの処理ﾒｿｯﾄﾞを呼ぶことができます。

・ただし、ここでthisﾎﾟｲﾝﾀを受け取るには、あらかじめ、
ｳｨﾝﾄﾞｳの GWLP_USERDATA に、格納しておく必要があります。

・MessageHandlerﾒｿｯﾄﾞは、CControlｸﾗｽのﾒﾝﾊﾞです。

・「仮想関数」として定義されていますので、上書きして使うこともできます。

・CFormｸﾗｽは、CControlｸﾗｽを派生させたものです。

・CControlｸﾗｽというのは、ｳｨﾝﾄﾞｳのﾊﾝﾄﾞﾙをくるんだｸﾗｽで、
すべてのｺﾝﾄﾛｰﾙｸﾗｽの親ｸﾗｽです。

・ｳｨﾝﾄﾞｳのﾊﾝﾄﾞﾙは、CreateWindow関数の戻り値で、
これがあれば、GetWindowLongPtr関数などから、
そのｳｨﾝﾄﾞｳに関する情報を取得したり、
SetWindowLongPtr関数などから
変更し直したりすることができます。

・ｳｨﾝﾄﾞｳのﾊﾝﾄﾞﾙは、「HWND型」です。

・**Windows** は、
ﾊｰﾄﾞﾃﾞｨｽｸ上の領域や、
ﾒﾓﾘ上の領域など、
すべてを管理しています。

・ｱﾌﾟﾘｹｰｼｮﾝｿﾌﾄが、
ﾊｰﾄﾞﾃﾞｨｽｸ上のﾌｧｲﾙや、
ｳｨﾝﾄﾞｳやｺﾝﾄﾛｰﾙなどを使用するときは、
Windowsからﾚﾝﾀﾙするような形を取り、
不要になったら返却します。

・ｱﾌﾟﾘｹｰｼｮﾝｿﾌﾄは、**Windows**から
「ｼｽﾃﾑﾘｿｰｽ」 (ｼｽﾃﾑ資源) をﾚﾝﾀﾙするときに、
それらを識別するための「ﾊﾝﾄﾞﾙ」というものを取得します。

・たとえば、ﾒﾆｭｰｺﾝﾄﾛｰﾙのﾊﾝﾄﾞﾙは「HMENU型」で、
CreateMenu関数の戻り値として取得することができます。

・WindowsAPIの関数には、
ﾘｿｰｽを操作するための関数が多数ありますが、
操作対象となるﾘｿｰｽを指定するために、
第一引数に、この「ﾊﾝﾄﾞﾙ」を渡します。

・まるで部屋を借りるような感覚で
Windowsからウィンドゥを貸してもらい、
料理を注文するような感覚で、
Windowsに位置を移動させてもらうのです。

・そして、使い終わったら、ウィンドゥを返却します。

・アプリケーションソフトは、
こうした指示を**Windows**へと送るために、
WindpwsAPIの関数を呼び出しているのです。

同期ｵﾌﾞｼﾞｪｸﾄ

・「同期ｵﾌﾞｼﾞｪｸﾄ」は、
　複数のｽﾚｯﾄﾞが並走している中で、
　ｽﾚｯﾄﾞを1つずつ動かしたい場合に使用します。

・複数のｽﾚｯﾄﾞが並走している場合は、
　同じﾃﾞｰﾀに対して同時に書き込みを行うことで、
　ﾃﾞｰﾀが破損してしまうことがあります。

・「同期ｵﾌﾞｼﾞｪｸﾄ」は、複数のｽﾚｯﾄﾞから
　同時にｱｸｾｽされるﾃﾞｰﾀに対して設定され、
　複数のｽﾚｯﾄが、同時にｱｸｾｽできなくします。

・WaitForSingleObject関数を呼ぶと、
　そのｽﾚｯﾄﾞは順番待ちに加わります。

・この時点で、動作中のｽﾚｯﾄﾞがなければ、
　このｽﾚｯﾄﾞが動作を開始します。（ｽﾚｯﾄﾞ1のｹｰｽ）

・そうでない場合は、そのｽﾚｯﾄﾞの処理が

終わるまで待機します。(ｽﾚｯﾄﾞ2のｹｰｽ)

ﾘｿｰｽﾌｧｲﾙを編集する

・ｺﾝﾄﾛｰﾙやﾒﾆｭｰなど、一部のｼｽﾃﾑﾘｿｰｽは、
　実行ﾌｧｲﾙの内部に埋め込むことができます。

・埋め込むﾘｿｰｽの情報は、「ﾘｿｰｽﾌｧｲﾙ」(***.rc**) に書きます。

(「ﾘｿｰｽｽｸﾘﾌﾟﾄ」という簡易ｽｸﾘﾌﾟﾄ言語で記述します)

・「**Express Edition**」では、ﾘｿｰｽﾌｧｲﾙの編集ができません。

(ﾌｧｲﾙ上で右ｸﾘｯｸして、「開く」ではなく、「ｺｰﾄﾞの表示」を選べば、表示はされます。)

・編集するには、「**ResEdit**」というｴﾃﾞｨﾀｿﾌﾄを使用します。↓

　http://gurigumi.s349.xrea.com/programming/visualcpp/resedit.html

---

・ｲﾝｽﾄｰﾙした時点では、英語で表示されていますが、
　日本語で表示することもできます。↓

　[Option]→[Preferences]→[General]→[Language] を、
　[Japanese] に切り替える。

・ﾌﾟﾛｼﾞｪｸﾄ側には、上記のｴﾃﾞｨﾀで作成した「ﾘｿｰｽﾌｧｲﾙ」と、
　「ﾘｿｰｽ**ID**」定数を**#define**した「ﾍｯﾀﾞﾌｧｲﾙ」を追加します。↓

// 《 **dialog_test.dlg** 》

MyDialog DIALOG 21, 16, 146, 55

```
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "タイトル"
FONT 12, "System"

BEGIN
  LTEXT "ﾗﾍﾞﾙﾃｷｽﾄ1", ID_LABEL1, 47, 6, 82, 8
  EDITTEXT ID_EDIT1, 44, 18, 94, 12, ES_AUTOHSCROLL
  PUSHBUTTON "OK", ID_OK, 44, 35, 40, 14
  PUSHBUTTON "ｷｬﾝｾﾙ", ID_CANCEL, 96, 35, 40, 14
END
```

```
// 《 dialog_test.h 》

#define ID_LABEL1 102
#define ID_EDIT1 103
#define ID_OK 104
#define ID_CANCEL 105
```

```
// 《 dialog_test.rc 》

#include "test_dialog.h"
#include "test_dialog.dlg"
```

```
// 《 dialog_test.cpp 》

#include "test_dialog.h"

int APIENTRY _tWinMain( HINSTANCE h_instance_ , /* 省略 */ )
{
   ::DialogBox( h_instance_, "MyDialog", NULL, MyDialogProc );

   return 0;
}
```

・ﾀﾞｲｱﾛｸﾞの場合は、「ﾀﾞｲｱﾛｸﾞﾌｧｲﾙ」 (***.dlg**) に、ﾘｿｰｽｽｸﾘﾌﾟﾄを書きます。

・この中で、「**ID_OK**」といった定数が使われていますが、
これは、ｺﾝﾄﾛｰﾙﾘｿｰｽの**ID**で、ｿｰｽﾌｧｲﾙ上でも使用しますので、
ﾍｯﾀﾞﾌｧｲﾙ上で定義しておきます。

ﾋﾞｼﾞｭｱﾙｽﾀｲﾙ

・**WindowsAPI**でﾎﾞﾀﾝを作ると、
「ｸﾗｼｯｸｽﾀｲﾙ」で表示されます。

・これを今風のﾃﾞｻﾞｲﾝにするには、
次のようにして、最新の「ﾋﾞｼﾞｭｱﾙｽﾀｲﾙ」を適用させます。↓

```
#pragma comment( lib,"UxTheme.lib")

// ｺﾓﾝｺﾝﾄﾛｰﾙ6.0を使うように指示する。
#pragma comment(linker,"\"/manifestdependency:type='win32' \
name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \
processorArchitecture='*' publicKeyToken='6595b64144ccf1df' language='*'\"")
// comctl32.lib を使う。(ﾍｯﾀﾞﾌｧｲﾙは「Comctl.h」のまま)
// これはXP以降であれば、Windowsのﾊﾞｰｼﾞｮﾝと同じｽﾀｲﾙが適用される。

#include "Uxtheme.h"

int APIENTRY _tWinMain(
                    HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPTSTR lpCmdLine, int nCmdShow )
{

  // 省略

  //ﾎﾞﾀﾝを作成する。
  CButton* p_button1 = new CButton();
  p_button1->Init( h_main_window, 0, L"ぼたん", 0, 0, 100, 40 );
  p_button1->Show();

   //ﾋﾞｼﾞｭｱﾙｽﾀｲﾙを適用する。
  ::SetWindowTheme( p_button1->GetWindowHandle(), L"BUTTON", 0 );

  // 省略
```

}

・引数**2**は、ｳｨﾝﾄﾞｳｸﾗｽ名で、 下記のﾍﾟｰｼﾞに一覧があります。 ↓
http://msdn.microsoft.com/en-us/library/windows/desktop/bb773210(v=vs.85).aspx

・**msdn**の説明。 ↓
http://msdn.microsoft.com/ja-jp/library/ms997646.aspx

・次のようにして適用することもできます。 ↓

```
// ﾃｰﾏを開く。
HTHEME h_theme = ::OpenThemeData( p_button1->GetWindowHandle(), L"BUTTON" );
```

```
::CloseThemeData( h_theme ); // ﾃｰﾏを閉じる。
```

・通常のｳｨﾝﾄﾞｳは、自動的に適用されます。
(※ﾃﾞｽｸﾄｯﾌﾟの設定画面で、ｸﾗｼｯｸｽﾀｲﾙに設定していた場合は、適用されません。)

ｸﾗｽの構成

CObject ... すべてのｸﾗｽの基底ｸﾗｽ。ｺﾚｸｼｮﾝ化するのに必要。

|

+- CAppli ... ｱﾌﾟﾘｹｰｼｮﾝ。

+- CText ... ﾃｷｽﾄ。

+- CBinary ... ﾊﾞｲﾄ配列。

+- CArray ... ｵｰﾄﾘｻｲｽﾞ配列。

+- CSmartArray ... 隙間を埋めてくれる配列。

+- CList ... ﾘｽﾄ。

+- CSmartArrayList ... 隙間を埋めてくれる配列のﾘｽﾄ。

+- CClipboard ... ｸﾘｯﾌﾟﾎﾞｰﾄﾞ。

+- CDateTime ... 日時。

+- CFont ... ﾌｫﾝﾄ。

+- CCharFormat ... 文字書式。

+- CParagraphFormat ... 段落書式。

+- CFileMap ... ﾌｧｲﾙﾏｯﾌﾟ。

+- CDirectory ... ﾃﾞｨﾚｸﾄﾘ。

+- CThread ... ｽﾚｯﾄﾞ。

+- CMath ... 算術系のﾏｸﾛなど。

+- CAclTable ... ｱｸｾﾗﾚｰﾀｰｷｰﾃｰﾌﾞﾙ。

|

+- CFile ... ﾌｧｲﾙの基底ｸﾗｽ。

|　　|

|　　+- CTxtFile ... ﾃｷｽﾄﾌｧｲﾙ。

|　　+- CBinFile ... ﾊﾞｲﾅﾘﾌｧｲﾙ。

|

+- CBmp ... Bitmapの基底ｸﾗｽ。

|　　|

|　　+- CDdb ... DDB。

|　　+- CDib ... DIB。

|

+- CDc ... ﾒﾓﾘﾃﾞﾊﾞｲｽｺﾝﾃｷｽﾄ。

+- CImageList ... ｲﾒｰｼﾞﾘｽﾄ。

+- CTreeNode ... ﾂﾘｰﾉｰﾄﾞ。

+- CAction ... 反応処理。

|　|

```
|    +- CWmAction ... ｳｨﾝﾄﾞｳﾒｯｾｰｼﾞ反応処理。
|    +- CCommandAction ... 通知ｺｰﾄﾞ反応処理。
|    +- CNotifyAction ... 通知ｺｰﾄﾞ反応処理。(ｺﾓﾝｺﾝﾄﾛｰﾙ用)
|
+- CSyncObject ... 同期ｵﾌﾞｼﾞｪｸﾄの基底ｸﾗｽ。
|           |
|           +- CCriticalSection ... ｸﾘﾃｨｶﾙｾｸｼｮﾝ
|           +- CMutex ... ﾐｭｰﾃｯｸｽ
|           +- CSemaphore ... ｾﾏﾌｫ
|           +- CEvent ... ｲﾍﾞﾝﾄ
|
+- CIme ... IME。
|
+- CControl ... ｺﾝﾄﾛｰﾙの基底ｸﾗｽ。ｳｨﾝﾄﾞｳﾊﾝﾄﾞﾙを包んだもの。
          |
          +- CForm ... ｳｨﾝﾄﾞｳ。
          |       |
          |       +- CDialog ... ﾓｰﾄﾞﾚｽﾀﾞｲｱﾛｸﾞ。
          |       +- CMdiFrame ... MDIﾌﾚｰﾑ。
          |       +- CMdiClient ... MDIｸﾗｲｱﾝﾄ。
          |       +- CMdiDoc ... MDIﾄﾞｷｭﾒﾝﾄ。
          |
          +- CModalDialog ... ﾓｰﾀﾞﾙﾀﾞｲｱﾛｸﾞ。
          +- CPanel ... ﾊﾟﾈﾙ。
          +- CGroupBox ... ｸﾞﾙｰﾌﾟﾎﾞｯｸｽ。
          +- CScrollBarH ... 水平ｽｸﾛｰﾙﾊﾞｰ。
          +- CScrollBarV ... 垂直ｽｸﾛｰﾙﾊﾞｰ。
          +- CMenu ... ﾒﾆｭｰ。
          +- CMenuItem ... ﾒﾆｭｰ項目。
          +- CButton ... 押しﾎﾞﾀﾝ。
          |   |
          |   +- CSplitButton ... 分割ﾎﾞﾀﾝ。
          |   +- CCommandLinkButton ... ｺﾏﾝﾄﾞﾘﾝｸﾎﾞﾀﾝ。
          |
          +- CRadioButton ... ﾗｼﾞｵﾎﾞﾀﾝ。
          +- CCheckBox ... ﾁｪｯｸﾎﾞｯｸｽ。
          +- CListBox ... ﾘｽﾄﾎﾞｯｸｽ。
```

+- CComboBox ... ｺﾝﾎﾞﾎﾞｯｸｽ。
| |
| +- CComboBoxEx ... 拡張ｺﾝﾎﾞﾎﾞｯｸｽ。
|
+- CTextBox ... ﾃｷｽﾄﾎﾞｯｸｽ。
+- CLabel ... ﾗﾍﾞﾙ。
+- CImageView ... ﾋﾟｸﾁｬｰﾎﾞｯｸｽ。
| |
| +- CCipGrid ... ｸﾞﾘｯﾄﾞのついた選択ｺﾝﾄﾛｰﾙ。
|
+- CTreeView ... ﾂﾘｰﾋﾞｭｰ。
+- CRichTextBox ... ﾘｯﾁﾃｷｽﾄﾎﾞｯｸｽ。

・「MFC」を簡単にしたような感じなんですが、
例によって、ﾈｰﾐﾝｸﾞが「.Net」寄りで、
ｻﾌﾞ ｸﾗｽ化だとか、ﾒｯｾｰｼﾞﾊﾝﾄﾞﾘﾝｸﾞの実際のところが、
細かく解説されている感じでしょうかね。

CMessageHandler

・MessageHandlerｸﾗｽ。^^ ↓(「.NET」の、「EventHandler」ｸﾗｽと似ている)

```
// ﾒｯｾｰｼﾞﾊﾝﾄﾞﾗ ｸﾗｽの定義 ( ﾒｯｾｰｼﾞごとの反応処理を、ｸﾗｽ化したもの )

class CControl;

class CMessageHandler : public CObject
{

public:

// ------------------------------------------------
// ﾒﾝﾊﾞ ( 呼び出しｺｽﾄを無くすために公開している )

UINT msg_code; // ﾒｯｾｰｼﾞｺｰﾄﾞ。

CMessageHandler* p_next; // 次のﾊﾝﾄﾞﾗ。無ければ終了。

// ------------------------------------------------
// ﾒｿｯﾄﾞ

CMessageHandler(); // ｺﾝｽﾄﾗｸﾀ

virtual BOOL IsEmpty() = 0; // ｺｰﾙﾊﾞｯｸ関数がNULLなら、TRUEを返す。

virtual void Execute( CControl* p_control_,
  HWND h_wnd_, WPARAM wparam_, LPARAM lparam_ ) = 0; // ｺｰﾙﾊﾞｯｸ関数を呼び出す。

~CMessageHandler(); // ﾃﾞｽﾄﾗｸﾀ

// ------------------------------------------------

}; // end class
```

```
// ｵｰﾙﾏｲﾃｨｰなﾒｯｾｰｼﾞﾊﾝﾄﾞﾗｸﾗｽの定義

typedef void (*AnyProc)( CControl* p_control_,
  HWND h_wnd_, WPARAM wparam_, LPARAM lparam_ ); // ｺｰﾙﾊﾞｯｸ関数の型

class CAnyHandler : public CMessageHandler
{

public:

// ---------------------------------------------------------------
// ｺｰﾙﾊﾞｯｸ関数

AnyProc p_proc;

// ---------------------------------------------------------------
// ｺﾝｽﾄﾗｸﾀ

CAnyHandler()
{
  msg_code = 0;
  p_proc = NULL;
  p_next = NULL;
}

// ------------------------------------------------
// ｺﾝｽﾄﾗｸﾀ

CAnyHandler( UINT msg_code_, AnyProc p_proc_ )
{
   msg_code = msg_code_;
   p_proc = p_proc_;
   p_next = NULL;
}

// ------------------------------------------------
```

```
// ｺｰﾙﾊﾞｯｸ関数がNULLなら、TRUEを返す。

BOOL IsEmpty()
{
    return ( p_proc == NULL );
}

// ------------------------------------------------------------
// ｺｰﾙﾊﾞｯｸ関数を呼び出す。

void Execute( CControl* p_control_,
              HWND h_wnd_, WPARAM wparam_, LPARAM lparam_ )
{
    (*p_proc)( p_control_, h_wnd_, wparam_, lparam_ );
}

// ------------------------------------------------

}; // end class
```

・ｺﾝﾄﾛｰﾙ側には、「CMessageHandler」ｸﾗｽのﾘｽﾄを持たせておいて、
ﾒｯｾｰｼﾞﾊﾝﾄﾞﾗの中で呼び出します。↓

```
// ★ｲﾝｽﾀﾝｽごとのﾒｯｾｰｼﾞ処理。（これ自身は、ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬから呼ばれる）

LRESULT CControl::MessageHandler(
  HWND h_wnd_, UINT msg_, WPARAM wparam_, LPARAM lparam_ )
{

  // ----------------
  CMessageHandler* p_handler = p_first_handler; // 最初のﾒｯｾｰｼﾞ処理を取り出す。

  while ( p_handler ) // ﾒｯｾｰｼﾞ処理があれば、
  {
```

```
    if ( msg_ == p_handler->msg_code ) // ﾒｯｾｰｼﾞｺｰﾄﾞが一致していたら、
    {
      p_handler->Execute( this, h_wnd_, wparam_, lparam_ ); // 処理を呼び出す。

        return ::CallWindowProcW( default_proc,
                                  h_wnd_, msg_, wparam_, lparam_ );
    }

    p_handler = p_handler->p_next; // 次のﾒｯｾｰｼﾞ処理があれば、ﾙｰﾌﾟを続ける。

  }

  // ----------------
  // 親ｳｨﾝﾄﾞｳに送信する。(※ｺﾝﾄﾛｰﾙ系のみ)

  return ::CallWindowProcW( default_proc,
                            h_wnd_, msg_, wparam_, lparam_ );

}
```

・速度的には、**switch**文でﾊｰﾄﾞｺｰﾃﾞｨﾝｸﾞした方が速い。

・その場合は、ｲﾝｽﾀﾝｽごとにｳｨﾝﾄﾞｳｸﾗｽを登録して、
別個のｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬを呼ぶようにします。

・ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬは、ｳｨﾝﾄﾞｳを作成した後でも変更できます。↓

```
::SetWindowLongPtrW( h_window, GWLP_WNDPROC,
                     (LONG) WindowProc1 ); // ｳｨﾝﾄﾞｳﾌﾟﾛｼｰｼﾞｬを差し替える。
```

・ﾘｽﾄへの追加は、単方向ﾘｽﾄなので簡単。^^ ↓

```
// ﾒｯｾｰｼﾞ処理を追加する。

BOOL CControl::AddMessageHandler( CMessageHandler* p_handler_ )
{

  if ( p_handler_ == NULL     ) return FALSE; // NULLﾁｪｯｸを、この時点で行っておく。
  if ( p_handler_->IsEmpty() ) return FALSE; // NULLﾁｪｯｸを、この時点で行っておく。

  if ( p_first_handler == NULL ) // 最初のﾒｯｾｰｼﾞ処理がないなら、
  {
    p_first_handler = p_handler_;      // 最初のﾒｯｾｰｼﾞ処理にする。
    p_last_handler = p_first_handler; // この時点では、最初と最後が同じ。
  }
  else // 最初のﾒｯｾｰｼﾞ処理があるなら、
  {
    p_last_event->p_next = p_event_;     // 最後のﾒｯｾｰｼﾞ処理の次に追加する。
    p_last_event = p_last_event->p_next; // 追加されたﾒｯｾｰｼﾞ処理が、最後になる。
  }

  return TRUE; // 登録成功。
}
```

・部分的に削除したいのであれば、双方向ﾘｽﾄにした方がいいでしょうね。

・ﾒｯｾｰｼﾞによっては、ﾊﾟﾗﾒｰﾀの取り出し処理がめんどくさい場合がありますが、
　これを省略したい場合は、次のようにも書けます。↓

```
// Touchﾊﾝﾄﾞﾗ ｸﾗｽ ( Windows SDK ver7.1以降で対応 )

typedef void (*TouchProc)( CControl*, TOUCHINPUT*, POINT*, size_t );

// この3つのﾒｯｾｰｼﾞは、取り出し処理が同じ。↓
enum TouchMessageType
```

```
{
  TMT_TouchDown = 0x0241,  // WM_TOUCHDOWN
  TMT_TouchUp   = 0x0242,  // WM_TOUCHUP
  TMT_TouchMove  = 0x0240  // WM_TOUCHMOVE == WM_TOUCH
};


class CTouchHandler : public CMessageHandler
{

public:

// ------------------------------------------------
// ｺｰﾙﾊﾞｯｸ関数

TouchProc p_proc;

// ------------------------------------------------
// ｺﾝｽﾄﾗｸﾀ

CTouchHandler()
{
  msg_code = 0;
  p_proc = NULL;
  p_next = NULL;
}

// ------------------------------------------------
// ｺﾝｽﾄﾗｸﾀ

CTouchEvent( TouchMessageType type_, TouchProc p_proc_ )
{
  msg_code = type_;
  p_proc = p_proc_;
  p_next = NULL;
}
```

```
// ------------------------------------------------
// ｺｰﾙﾊﾞｯｸ関数がNULLなら、TRUEを返す。

BOOL IsEmpty()
{
    return ( p_proc == NULL );
}

// ------------------------------------------------
// ｺｰﾙﾊﾞｯｸ関数を呼び出す。

void Execute( CControl* p_control_,
              HWND h_wnd_, WPARAM wparam_, LPARAM lparam_ )
{

 HTOUCHINPUT h_touch = (HTOUCHINPUT) lparam_; // ﾀｯﾁ入力のﾊﾝﾄﾞﾙ
 size_t touch_count = LOWORD( wparam_ ); // ﾀｯﾁ入力点の数
 TOUCHINPUT* p_inputs = new TOUCHINPUT[ touch_count ];
 POINT* p_points = new POINT[ touch_count ];

 // p_inputs に ﾀｯﾁ情報配列を格納する。
 if ( ::GetTouchInputInfo( h_touch, touch_count, p_inputs, sizeof(TOUCHINPUT) ) )
 {
   for ( size_t i = 0 ; i < touch_count; i++ ) // ｲﾝｸﾘﾒﾝﾄしながら処理していく。
   {
     TOUCHINPUT ti = p_inputs[i]; // ﾀｯﾁ入力情報を取り出す。
     p_points[i].x = TOUCH_COORD_TO_PIXEL( ti.x ); // ﾋﾟｸｾﾙ座標値に変換する。
     p_points[i].y = TOUCH_COORD_TO_PIXEL( ti.y ); // ﾋﾟｸｾﾙ座標値に変換する。
     ::ScreenToClient( h_wnd_, &p_points[i] ); // ｸﾗｲｱﾝﾄ座標に変換する。
     // ※ScreenToClient関数を使用するには、高DPIに対応させておく必要がある。
   } // end for

   (*p_proc)( p_control_, p_inputs, p_points, touch_count ); // ｺｰﾙﾊﾞｯｸ関数を呼ぶ。

   ::CloseTouchInputHandle( h_touch ); // ﾊﾝﾄﾞﾙを必ず閉じる。

 } // endif
```

```
  delete [] p_inputs;
  delete [] p_points;

}

// ------------------------------------------------

}; // end class
```

・簡単でいいんですが、なんにもしない場合でも
取り出し処理を行うため、無駄が多い。

・ﾒｯｾｰｼﾞを処理しない時は、対応するﾊﾝﾄﾞﾗを
　ﾘｽﾄから除外しておくなど、工夫が必要です。

奥付

ハム太と作るC++ライブラリ　-ｺﾝﾄﾛｰﾙ編-

http://p.booklog.jp/book/76727

著者 : haseham
著者プロフィール：http://p.booklog.jp/users/haseham/profile

感想はこちらのコメントへ
http://p.booklog.jp/book/76727

ブクログ本棚へ入れる
http://booklog.jp/item/3/76727

電子書籍プラットフォーム：ブクログのパブー（http://p.booklog.jp/）
運営会社：株式会社ブクログ