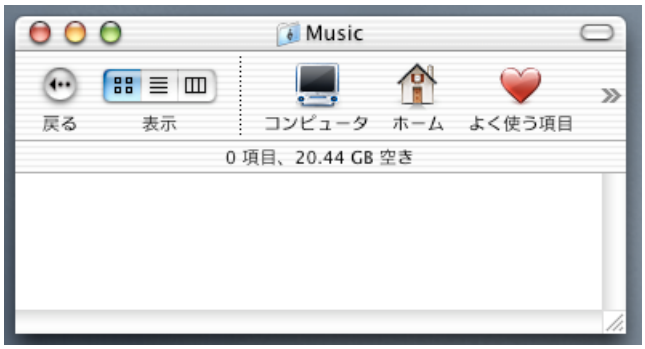
小池邦人のプログラミング日記》2002/3/4
ウィンドウのグループ化　その２

今回は、ウィンドウのグループ化の話の続きとなります。ウィンドウタイトルに配置できるようになった ToolBar ボタンの利用方法についても解説します。

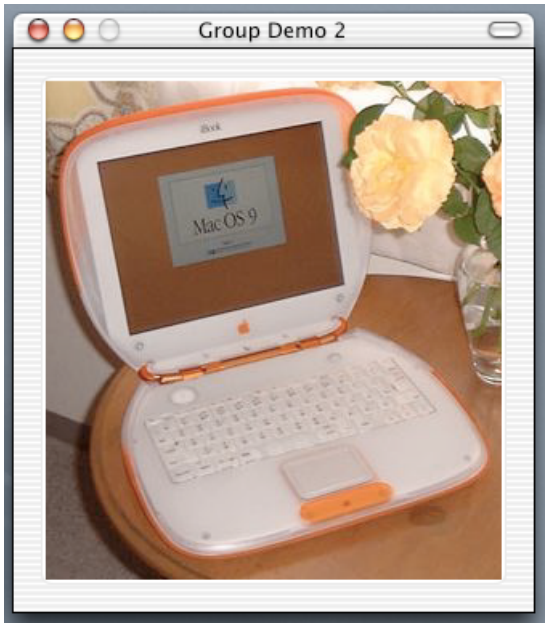
Carbon アプリケーションの開発で、すべてのリソースと縁を切ることは無理なのですが、Nib ファイルのおかげで、ダイアログとメニューについてはリソース編集をする機会はなくなりました。ただし、大きな問題は、Nib ファイルを編集するツールが現状では Interface Builder しかないことです。このアプリケーション、使っていると操作に対する反応速度がどんどん鈍くなっていきます。ポップアップメニューやグループ・ラジオボタンの編集では、オブジェクトをちょっと移動するだけでレインボーカーソルが 10 秒ぐらい回りっぱなしになります。21 世紀の世の中、如何様にするとこんなに反応が遅いアプリケーションが開発できるのか？大きな疑問です。うちだけなのか？日本語環境との相性なのか？ガーベージコレクションをしてるのか？Cocoa フレームワークにボトルネック API があるのか？Nib ファイルのデータ構造がバカなのか？Interface Builder のデータハンドリングがタコなのか？とにかく、最近のストレスの大部分は、Interface Builder での作業に集中しているわけです（涙）。

さて、Mac OS X から新しく採用された ToolBar ボタンを利用する「Group_Demo2」サンプルアプリケーションを紹介しましょう。ToolBar ボタンはウィンドウの右上に表示されます。Mac OS X 10.1 の Finder では、このボタンをクリックすることで、ウィンドウ上部のツールバーを出し入れすることが可能です。



ただ残念なことに、現在の Carebon フレームワークではツールバーを操作するためのシステムモジュール（ToolBar Manager？）は提供されていません。Cocoa フレームワークには、これを操作するためのクラスライブラリが用意されています。興味がある方は、Developer/Examples/AppKit フォルダの「SimpleToolbar」とういう Cocoa ベースのプロジェクトを参考にしてみてください。そんなわけで、Carbon アプリケーションにとって ToolBar ボタンは「宝の持ち腐れ」に近い状態なのですが、それ自体をハンドリングすることは可能です。ToolBar ボタンは、Close ボックスや Zoom ボックスと同じように Carbon Event Handler ルーチンでハンドリングします。

サンプルアプリケーションを起動すると、ウィンドウがひとつオープンします。ここで、ToolBar ボタンをクリックすると、画像表示の部分が持ち上がり、スライダーとボタンを表示している領域を隠します。



もう一度クリックすると、上がった領域が元に戻り、スライダーとボタンが復活しま

す。「終了」ボタンをクリックするとアプリケーションを終了できますが、スライダーの方はダミーであり、機能は何も割り付けてありません。

実は、ウィンドウのコントロールを表示している部分（ツールバー）と画像表示の部分は、別々のウィンドウで構成されています。両ウィンドウをグループ化することにより、あたかもひとつのウィンドウのように見せているわけです。それでは、このサンプルアプリケーションの main()ルーチンを見てみます。

```
WindowGroupRef  sys_group;      /*  Group Window References */

int main(int argc, char* argv[])
{
    WindowRef   window,wptr;
    IBNibRef    nibRef;
    long        flag;

    if( ! CreateNibReference( CFSTR("main"),&nibRef ) )
    {
        SetMenuBarFromNib( nibRef,CFSTR("MenuBar") );
        if( ! CreateWindowFromNib( nibRef,CFSTR("ToolBarWindow"),&window ) )
        {
            flag=kWindowGroupAttrMoveTogether+kWindowGroupAttrSharedActivation+kWindowGroupAttrHideOnCollapse;
            CreateWindowGroup( flag,&sys_group );
            SetWindowGroup( window,sys_group );
            if( ! CreateWindowFromNib( nibRef, CFSTR("DisplayWindow"),&wptr ) )
            {
                setUpWindowEvent( window,wptr );
                SetWindowGroup( wptr,sys_group );
                SelectWindow( wptr );
                ShowWindow( window );
                ShowWindow( wptr );
                ChangeWindowGroupAttributes( sys_group,kWindowGroupAttrLayerTogether,kWindowGroupAttrSelectAsLayer );
            }
        }
        DisposeNibReference( nibRef );
        RunApplicationEventLoop();
    }
    return( noErr );
}
```

Nib ファイルから、ツールバー部分のウィンドウ（ToolBarWindow）と、画像表示部

分のウィンド（DisplayWindow）の両方を読み込んでいます。どちらのウィンドウもCreateWindowGroup()で作られたウィンドウグループに属します。DisplayWindow の方を SelectWindow()で選択したら、ChangeWindowGroupAttributes()でグループのアトリビュート（性質）を変更し、グループ間の上下関係（レイアー）がマウスクリックで変わらないように設定します。ToolBar ボックスがクリックされた時の処理は、setUpWindowEvent()でインストールされた Carbon Event Handler が行います。この時、setUpWindowEvent()には DisplayWindow の WindowRef も渡しておきます。これにより、この WindowRef はユーザデータ（UserData）として Handler ルーチンでも参照できるようになります。

```
void setUpWindowEvent( WindowRef window,WindowRef wptr )
{
    EventTypeSpec   list1[]={ { kEventClassWindow, kEventWindowToolbarSwitchMode },
                              { kEventClassWindow, kEventWindowClose }
                            };
    EventHandlerRef ref;

    InstallWindowEventHandler( window,NewEventHandlerUPP( myWindowEventHandler ),2,list1,(void *)wptr,&ref );
}
```

ふたつの EventTypeSpec のうち、イベントクラスが kEventClassWindow でイベント種類が kEventWindowToolbarSwitchMode の方が、ToolBar ボックスのクリックに対応した物です。処理を担当する myWindowEventHandler()は以下のようになります。

```
static pascal OSStatus myWindowEventHandler(EventHandlerCallRef myHandler, EventRef event, void* userData)
{
    short           ret=eventNotHandledErr;
    WindowRef       wptr=NULL;
    static long     w_dir=-1;
    WindowRef       wptr1;
    Rect            drt;

    if( GetEventClass( event )==kEventClassWindow )
    {
        GetEventParameter( event,kEventParamDirectObject,typeWindowRef,NULL,sizeof(WindowRef),NULL,&wptr );
        switch( GetEventKind(event) )
        {
            case kEventWindowToolbarSwitchMode:

                wptr1=(WindowRef)userData;
                GetWindowBounds( wptr1,kWindowStructureRgn,&drt );
                OffsetRect( &drt,0,40*w_dir );
                ChangeWindowGroupAttributes( sys_group,0,kWindowGroupAttrMoveTogether );
                TransitionWindow( wptr1,kWindowSlideTransitionEffect,kWindowMoveTransitionAction,&drt );
                ChangeWindowGroupAttributes( sys_group,kWindowGroupAttrMoveTogether,0 );
                if( w_dir==1 )
                    w_dir=-1;
                else
                    w_dir=1;
                ret=noErr;
                break;

            case kEventWindowClose:

                QuitApplicationEventLoop();
                ret=noErr;
                break;
        }
    }
    return( ret );
}
```

DisplayWindow の WindowRef はユーザデータ（userData）に代入されてきます。ToolBar ボタンのクリックで DisplayWindow を上下移動させには TransitionWindow()を使います。移動後の状態は、static 変数の w_dir に保存されますので、次回のクリックでは逆方向への移動が可能になるわけです。DisplayWindow を移動させる時には、ChangeWindowGroupAttributes()で「すべてを同時に移動させる」アトリビュート（kWindowGroupAttrMoveTogether）を解除しておくことを忘れないでください。そうしないと、TransitionWindow()により両方のウィンドウが同時に移動してしまいます。解除したアトリビュートは、移動が終了した時点で再度設定し直します。この処理を忘れると、タイトルバーのドラッグでウィンドウを移動させた時に、DisplayWindow だけ移動せずに、その場に置き去りにされてしまいます。

ここで解説した「Group_Demo2」サンプルアプリケーションは、以下のサイトに登録されていますので試してみてください。Mac OS X 10.1 と最新版の Developer Tools が必要です。

http://www.ottimo.co.jp/library/

次回は、Overlay ウィンドウの使い方を解説します。Overlay ウィンドウとは、Mac OS X から採用された機能で、Quartz 2D を使い半透明のオブジェクトをモニタ上に表示することが可能です。