Cocoaはやっぱり! 出張版

2001.9

3. ウィンドウの形状変更と透明化

■ 今回のテーマ

今回は、前回に引き続いて、Appleから配付されているサンプルコードの解説です。**RoundTransparentWindow**というサンプルで、これは、**ウィンドウを特殊な形状にしたり、半透明にしたりするもの**です。このサンプルコードを読む上で、ポイントになるところをピックアップして解説していきます。まずは、以下のURLからダウンロードして解凍しておいてください。

http://developer.apple.com/samplecode/Sample_Code/Cocoa/RoundTransparentWindow.htm

■ このサンプルから学べること

このサンプルを読むことで何が学べるのか、まず、そのポイントを列挙してみました。

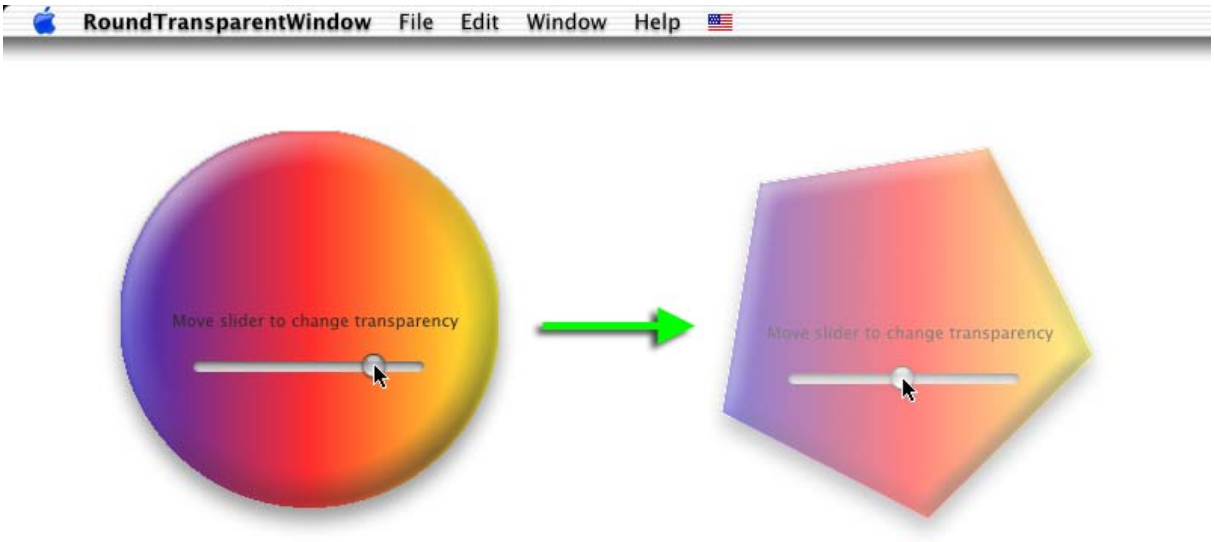
- **ウィンドウの形状を自由に変える方法**
- **ウィンドウ全体の透明度を変える方法**
- **ウィンドウを自分でドラッグする方法**
- **メニューバーより手前にウィンドウを表示する方法**

Mac OS Xには、Clockアプリケーションという時計のアプリケーションが添付されています。このアプリケーションは、**通常のウィンドウとは異なり、丸い形をしています**。また、環境設定を開くと分かりますが、**ウィンドウ全体の透明度を変えることが出来て**、背景が透けて見えるように出来ます。

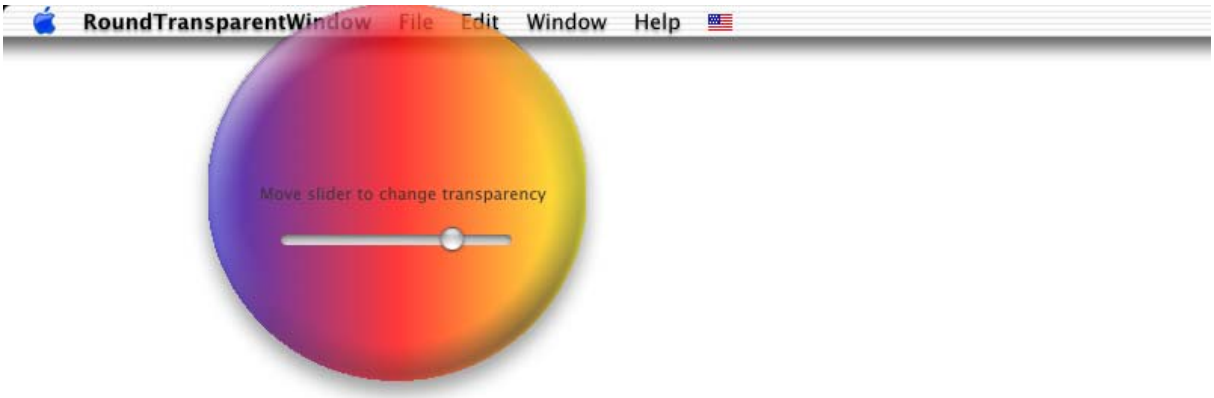
また、このウィンドウにはウィンドウの**上部にタイトルバーがありません**。そのため、ウィンドウをドラッグして移動する処理は、自分で処理をしているのです。さらに、一般にウィンドウはメニューバーの上にかぶさることは出来ませんが、**Clockアプリケーションはメニューバーの上にまでドラッグできます**。このサンプルアプリケーションは、これらの実現方法が全て分かるようになっています。

■ アプリケーション概要

さて、このサンプルのアプリケーションがどんな機能を持っているのか、まずは実際に動かして確かめてみましょう。ダウンロードしたファイルの中には、ビルドされたアプリケーションが入っているので、ダブルクリックするだけで、すぐに動かすことが出来ます。



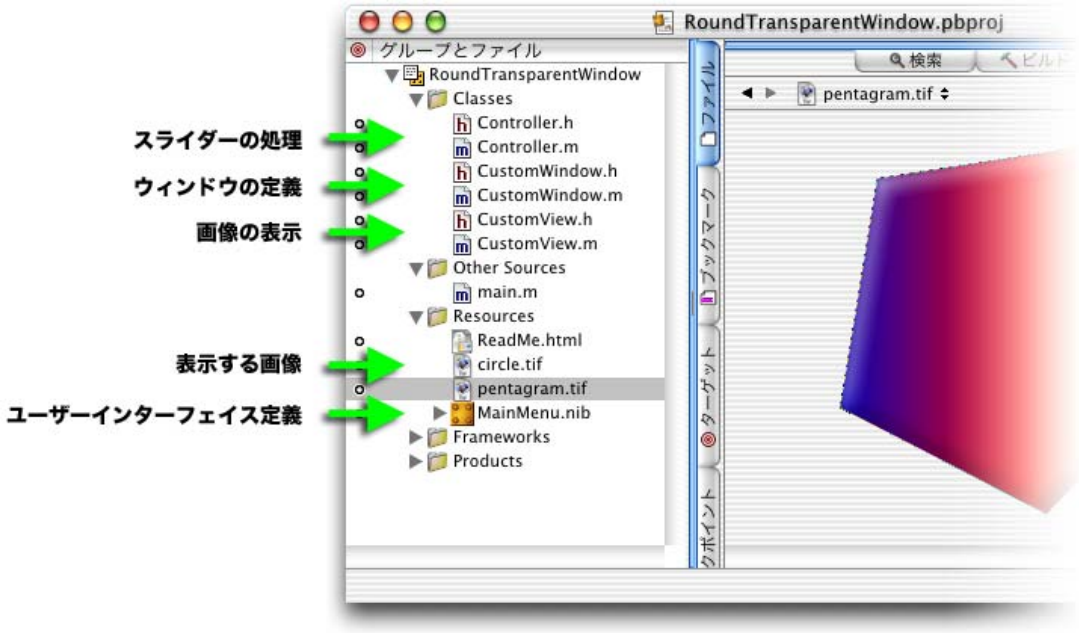
起動すると、上図のようにスライダーを1つ持っている丸いウィンドウが表示されます。このスライダーを動かすと、右のように全体の透明度が増していき、また、途中でウィンドウの形状が五角形に変わります。スライダーを戻すと、ウィンドウも最初の状態に戻ります。



また、このウィンドウにはタイトルバーがありませんが、ウィンドウ本体をつかんで移動が出来ます。そして、メニューバーの上にも移動させることが出来ます。

■ プロジェクトのファイル構成

では、プロジェクトを構成しているファイルを見ていきます。まず、Project Builderでプロジェクトファイル（`RoundTransparentWindow.pbproj`）を開きます。

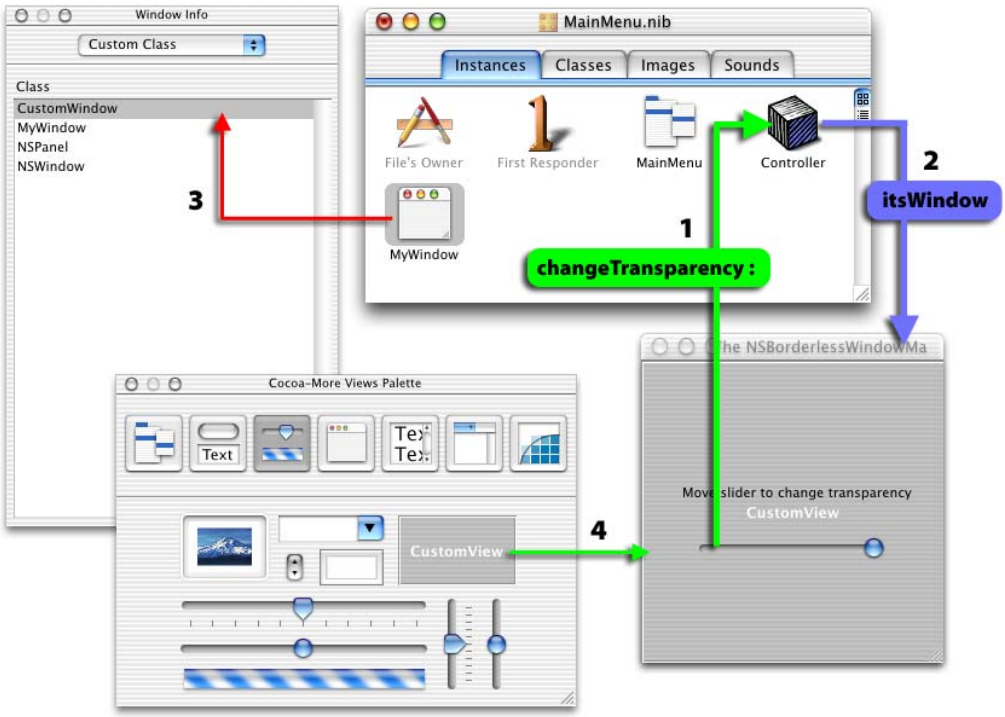


Classesのところに、「 Controller.h 」と「 Controller.m 」がありますが、これはスライダーを動かした時の動作を受け持つ**Controller**という名前のクラスの定義をしています (NSObjectのサブクラス)。「 CustomWindow.h 」と「 CustomWindow.m 」は、このウィンドウを実現するための**CustomWindow**という名前のクラスを定義しているものです (NSWindowのサブクラス)。「 CustomView.h 」と「 CustomView.m 」では、ウィンドウの中に円や五角形を描画するための**CustomView**クラスを定義しています (NSViewのサブクラス)。

Resourcesのところには、「 circle.tif 」と「 pentagram.tif 」という2つの画像がありますが、これがウィンドウの中に表示されるものです。名前のとおりですが、前者が円形で後者が五角形の画像です。どちらの画像も、図形の外側が透明になっているところがポイントです。「 MainMenu.nib 」には、ユーザーインターフェイス等の定義がなされています。

■ MainMenu.nibの構成

次に、MainMenu.nibをダブルクリックして、Interface Builderでその中身を見てみましょう。



MainMenu.nibのウィンドウを見ると、先程出てきたControllerクラスのインスタンスがあります。これを作るには、NSObjectのサブクラスを作ってインスタンス化します。このControllerには、スライダー(NSSlider) から**changeTransparency :** というメッセージが送られます…**1**。このメッセージを受けると、ウィンドウの透明度を変えます。そのためにウィンドウを参照するアウトレットの**itsWindow**を持っています…**2**。

このサンプルではウィンドウの振る舞いを変えるので、NSWindowのサブクラスとしてCustomWindowというクラスを作っておきます。そうするとウィンドウのインスタンスをInfoウィンドウでCustomWindowクラスに切り替えることが出来るのです…**3**。

そして、ウィンドウ内に描画を行うための独自のビューであるCustomViewをウィンドウ全体を覆うように配置しています。これも、予めCustomViewクラスをNSViewのサブクラスとして予め作っておいて、Infoウィンドウで CustomViewクラスに切り替えます…**4**。

■ 形状変更の仕組み

ウィンドウ上の各ピクセルは、描画を行うことで色を変えることが出来ますが、**色だけでなく透明度も変えることが出来ます**。透明色を指定したピクセルは、後ろに隠れているウィンドウやデスクトップが透けて見えるようになります。丸いウィンドウにしたい場合は、正方形のウィンドウを用意しておいて、丸の内側を不透明の色で塗り、丸の外側を透明色で塗ってしまえばよいわけです。このように、ウィンドウ自体は四角形なのですが、**透明色で塗ることによってウィンドウの形状を変えることが出来る**わけです。さらに、**完全に透明になっているピクセルはクリックできなくなるため**、背後にあるものをクリックすることになります。つまり、クリックされた時に、そこが透明になっているかどうかをアプリケーションが判断してイベント処理を分ける必要はありません。

■ 形状変更の手順

仕組みは分かりましたが、それを実際にコード上でどう書くのかを見ていきます。ウィンドウを変形するには、以下の2段階の処理を行うことになります。

> **1. ウィンドウ全体を完全に透明にする**
> **2. 不透明にしたい個所を描画する**

まず、ウィンドウ上の全てのピクセルを完全に透明にしておいて、その後から描画をすることで、描画したところのみがウィンドウとして見えてくることになります。

■ ウィンドウ全体の透明化手順

ウィンドウ全体を透明化する方法を、さらに詳しくみていきましょう。以下の3段階の処理を行います。

> **1. ウィンドウの背景色を透明色にすること**
> **2. ウィンドウの不透明フラグをNOにすること**
> **3. ウィンドウのスタイルマスクをNSBorderlessWindowMaskにすること**

1.の「 ウィンドウの背景色を透明色にする 」は、以下の1行で行えます。NSWindowのインスタンスに対して、**setBackgroundColor :** を送って透明色である **clearColor** を指定します。下のコードの「window」は存在しているNSWindowのインスタンスです。

```
[ window setBackgroundColor : [ NSColor clearColor ] ];
```

NSWindow : 背景色を指定する

書式
```
- (void) setBackgroundColor : (NSColor *) color
```
入力
```
color : 背景色
```

2.の「 ウィンドウの不透明フラグをNOにする 」は、以下の1行で行えます。**setOpaque :** にNOをセットします。

```
[ window setOpaque : NO ];
```

3.のウィンドウの**スタイルマスクというのは、ウィンドウの大まかな形状を決めるもの**で、以下の値を足し合わせて指定します。**完全に透明にするときには、NSBorderlessWindowMaskを使用**します。

ウィンドウの形状指定

```
enum {
  NSBorderlessWindowMask      = 0,  // 枠が無い
  NSTitledWindowMask          = 1,  // タイトル部分を持つ
  NSClosableWindowMask        = 2,  // クローズボックスを持つ
  NSMiniaturizableWindowMask  = 4,  // 最小化ボタンを持つ
  NSResizableWindowMask       = 8   // リサイズボックスを持つ
};
```

このスタイルマスクは、**ウィンドウを表示した後から変更することはできません**。ウィンドウの初期化段階で指定したものに固定されるためです。初期化の段階でスタイルマスクを指定するためには、NSWindowのサブクラスを作成して、初期化メソッドをオーバーライドすることになります。NSWindowのサブクラスを作っているのはこのためです。

CustomWindowクラスのソースを書き出したら、Project Builderで初期化メソッドを以下のように書き換えます。初期化メソッドの名前は、長いですが、「 **initWithContentRect : styleMask : backing : defer :** 」といいます。

```
- (id) initWithContentRect : (NSRect            ) contentRect
                 styleMask : (unsigned int      ) aStyle
                   backing : (NSBackingStoreType) bufferingType
                     defer : (BOOL              ) flag
{

   NSWindow* win = [ super initWithContentRect : contentRect
                                     styleMask : NSBorderlessWindowMask
                                       backing : NSBackingStoreBuffered
                                         defer : NO ];

   return win;
}
```

ウィンドウ初期化時には、このメソッドが呼ばれます。MainMenu.nibの中のウィンドウをCustomWindowクラスにしたので、このメソッドが呼ばれるようになります。この中で、スーパークラス、つまりNSWindowの同じ初期化メソッドを呼んでいますが、ここでstyleMaskの箇所のパラメーターをNSBorderlessWindowMaskにすりかえているわけです。これで、スタイルマスクを変えることが出来ます。

このコードに、最初の2段階の処理もこの中に入れてしまいましょう。これで透明ウィンドウを作る初期化メソッドの出来上がりです。

```
- (id) initWithContentRect : (NSRect            ) contentRect
                 styleMask : (unsigned int      ) aStyle
                   backing : (NSBackingStoreType) bufferingType
                     defer : (BOOL              ) flag
{
```

```
   NSWindow* win = [ super initWithContentRect : contentRect
                                     styleMask : NSBorderlessWindowMask
                                       backing : NSBackingStoreBuffered
                                         defer : NO ];

   [ win setBackgroundColor : [ NSColor clearColor ] ];
   [ win setOpaque : NO ];

    return win;
}
```

これと、サンプルコードのCustomWindow.mのソースを比較してみてください。いくつか追加の処理は書かれていますが、ほぼ同様のことが書かれているはずです。個人的には、これだけのためにサブクラスを作るのはちょっと面倒な気もしますので、Interface Builder上でスタイルマスクの指定が出来るようになったらいいのになぁと思います。

■ ウィンドウ内の描画の手順

ウィンドウの中への描画については、CustomViewクラスのdrawRect :メソッドで行います。ウィンドウ内の描画と今まで書いてきましたが、実際には、ウィンドウ上に配置されているビューへ描けばよいのです。まずは、このCustomViewの初期化のところから見ていきます。初期化コードはawakeFromNibに書きます。awakeFromNibについては前回説明しましたので、詳しくは前回の記事を見てください。アプリケーション起動時にMainMenu.nibが読み込まれますが、その後にnibファイル内のインスタンスのawakeFromNibが呼ばれます。

CustomView.m > awakeFromNib

```
  circleImage = [ NSImage imageNamed : @"circle"    ];  // 円の画像
  pentaImage  = [ NSImage imageNamed : @"pentagram" ];  // 五角形の画像
  [ self setNeedsDisplay : YES ];                       // 描画要求
```

ここで、ウィンドウの中に表示する画像を２つ読み込んでいます。**imageNamed :** というメソッドは、プロジェクトのResourcesに登録されている画像を読み込んでNSImageを作成するものです。最後に、**setNeedsDisplay :** で自分自身に描画要求を行っていますが、これで、自分自身のdrawRect : が呼ばれて描画が行われることになります。その描画メソッドであるdrawRect :は以下のようになります。

CustomView.m > drawRect :

```
- (void) drawRect : (NSRect) rect
{
      : 略
    if ( [ [ self window ] alphaValue ] > 0.7 ) // ウィンドウの透明度で振り分け
      [ circleImage compositeToPoint : NSZeroPoint
                           operation : NSCompositeSourceOver ]; // 円を描画
```

```
    else
      [ pentaImage  compositeToPoint : NSZeroPoint
                          operation : NSCompositeSourceOver ]; // 五角形を描画

    [ [ self window ] setHasShadow : NO  ]; // ウィンドウの影の再計算のため
    [ [ self window ] setHasShadow : YES ];
}
```

最初にウィンドウの透明度によって円を描くのか、五角形を描くのかを振り分けています。後で出てきますが、スライダーを動かすとウィンドウの透明度が変わるようになっています。ウィンドウの透明度を取得するのが**alphaValue**です。それから、「 self window 」と書いてありますが、windowメソッドで、このCustomViewが配置されている**親のウィンドウ**を取得できるのです。

```
    if ( [ [ self window ] alphaValue ] > 0.7 ) // ウィンドウの透明度で振り分け
```

NSWindow : ウィンドウの透明度を取得

書式
```
  - (float) alphaValue
```
出力
```
  返り値 : 透明度 (不透明 0.0 〜 1.0 透明 )
```

さて、描画に戻ります。画像 (NSImage) の描画には、**compositeToPoint : operation :** を使っています。このメソッドは、描画の場所と背景との画像の演算の種類を指定できます。**NSZeroPoint**は、原点(0, 0)を表します。**NSCompositeSourceOver**は、描画先との演算指定ではもっともよく使用されるもので「 **不透明の部分は上書きして、透明部分は何もしない 」**という演算です。

```
      [ circleImage compositeToPoint : NSZeroPoint
                          operation : NSCompositeSourceOver ]; // 円を描画
```

NSImage : 指定位置に指定透明度、指定合成方法で画像を表示

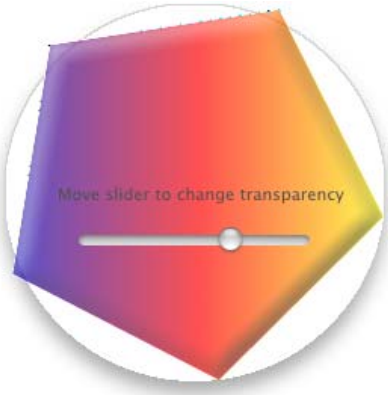
書式
```
  - (void) compositeToPoint : (NSPoint) point
                  operation : (NSCompositingOperation) op
```
入力
```
  point : 表示位置
  op    : 合成方法
```

最後に、ウィンドウの影のありなしを切り替えています。

```
    [ [ self window ] setHasShadow : NO  ]; // ウィンドウの影の再計算のため
    [ [ self window ] setHasShadow : YES ];
```

意味のないことをしているようですが、透明なウィンドウでは、ウィンドウの中に描画を行うと、ウィンドウ

の形が変わり、それに連動してウィンドウの影の形も変わらなければなりません。しかし、ウィンドウの影の形は自動的には更新されません。更新させるために、ウィンドウの影をオフ／オンを行っています。この処理をしないと、以下のように直前の影が残った状態になってしまいます。



■ ウィンドウの透明度変更

次は、スライダーの処理を見てみましょう。Controller.mを見てください。

Controller.m > changeTransparency :

```
- (IBAction) changeTransparency : (id) sender
{
   [ itsWindow setAlphaValue : [ sender floatValue ] ];  // 透明度変更
   [ itsWindow display ];                                // 再描画
}
```

先程のdrawRect : の中で、ウィンドウの透明度を参照していましたが、設定をしているのはここです。**setAlphaValue :** で透明度を変えることが出来ます。そして、displayメソッドで、ウィンドウを再描画させています。つまり、そのウィンドウの中にある、CustomViewも再描画されますので、先程のdrawRect :も呼ばれることになります。

NSWindow : ウィンドウの透明度を設定

書式

```
- (void) setAlphaValue : (float) windowAlpha
```

入力

windowAlpha : 透明度 (不透明 0.0 〜 1.0 透明)

これを使うと、ウィンドウを画面に表示させる時にフェードインさせたり、ウィンドウを閉じる時にフェードアウトさせるような遊びをすることも可能になります。

■ ウィンドウの自前ドラッグ処理

ウィンドウのスタイルマスクでタイトルバーを無くしてしまったので、ウィンドウを移動させる手段がありません。そのため、ウィンドウ (CustomWindow) にマウスダウンイベントとマウスドラッグイベントに対して、ウィンドウを移動する処理を書いておく必要があります。そのためには、**NSWindowのmouseDown : メソッドとmouseDragged : メソッドをオーバーライドします**。ウィンドウやビューは、マウスボタンが自分自身の上で押されると、 mouseDown : メソッドが、そのままドラッグされるとmouseDragged : メソッドが呼ばれます。

■ マウスダウン処理

サンプルのmouseDown : メソッドでは、CustomWindowのinitialLocationというNSPoint型のインスタンスに、ウィンドウのどこでマウスボタンが押されたかの座標を、ウィンドウの座標系で覚えています。

CustomWindow.m > mouseDown :

```
- (void) mouseDown : (NSEvent *) theEvent
{
      : 省略
   initialLocation = [ self convertBaseToScreen :
                          [ theEvent locationInWindow ] ];  // クリック座標を取得
      : 省略
}
```

mouseDown : のメソッドには**NSEvent**クラスのパラメータが1つ渡ってきますが、ここにイベントに関する情報が入っています。**マウスボタンが押された座標はNSEventのlocationInWindowメソッドで得ることが出来ます**。これは、ウィンドウの左下を原点とする座標系の値になっています。**ConvertBaseToScreen : メソッドで画面の左下を原点とする座標系に変換しています**。左下というのを不思議に思うかもしれませんが、Cocoaでは、左上ではなく、左下が座標原点になっています。数学で使っていた上に向かってY座標の値が増える座標系を使っています。

NSEvent : イベント発生時のマウスの位置を取得

書式
```
- (NSPoint) locationInWindow
```
出力
返り値 : マウスの位置

NSWindow : ウィンドウ内の座標からスクリーンの座標に変換

書式
```
- (NSPoint) convertBaseToScreen : (NSPoint) aPoint
```
入力
aPoint : ウィンドウ座標での位置
出力
返り値 : スクリーン座標での位置

■ マウスドラッグ処理

CustomWindow.m > mouseDragged :

```
- (void) mouseDragged : (NSEvent *) theEvent
{
      : 省略
   NSPoint newOrigin; // 新しいウィンドウの位置
   NSRect  screenFrame = [ [ NSScreen mainScreen ] frame ]; // 画面の大きさ
   NSRect  windowFrame = [ self frame ];          // ウィンドウの画面での位置
      : 省略
    // クリック座標を取得
    currentLocation = [ self convertBaseToScreen :
                          [ self mouseLocationOutsideOfEventStream ] ];
      : 省略
    [ self setFrameOrigin : newOrigin ];  // ウィンドウを移動
}
```

mouseDragged : メソッドでは、mouseDown : メソッドで記憶したマウスが最初に押された時の座標からのずれを計算して、新しいウィンドウの場所 (左下隅の座標) を計算しています。ここでのマウスの座標の取得方法は、**NSWindow**の**mouseLocationOutsideOfEventStream** を使っています。先程使った、NSEventのlocationInWindowを使っても同じ値になります。イベント (NSEvent) から座標を得る方法と、ウィンドウ (NSWindow) から座標を得る方法があるということを示したかったのでしょうか、あえて、別の方法で実装しています。

NSWindow : マウスの座標を取得

書式

```
 - (NSPoint) mouseLocationOutsideOfEventStream
```

出力

返り値　: ウィンドウ座標でのマウスの位置

計算の中で、ウィンドウの上端が、画面の上端からはみださないような処理もしています。このときのために、画面の大きさを得ていますが、これには、**NSScreen**クラスを使います。**mainScreen**メソッドで、メインスクリーンのインスタンスが得られますので、これの**frame**メソッドで座標を得ることが出来ます。

NSScreen : メインスクリーンを取得

書式

```
 + (NSScreen *) mainScreen
```

出力

返り値　: メインスクリーンのインスタンス

NSScreen : スクリーンの位置情報を取得

書式
```
- (NSRect) frame
```
出力
返り値　: スクリーンの位置情報

■ メニューバーより手前に表示する

CustomWindow.mのソースの初期化メソッドの中に以下の行があります。

```
[ result setLevel : NSStatusWindowLevel ];
```

ウィンドウのレベルをセットしているメソッドですが、これは、画面の中でウィンドウが表示される高さを決めるものです。高いというのは、より手前に表示されるという意味です。**NSStatusWindowLevel**というのはドキュメントがまだ整備されていませんので厳密なところは分かりませんが、メニューよりもDockよりも手前で、他のアプリケーションをアクティブにしても手前に表示されますので、最も手前の部類に属するレベルであることは間違いないと思われます。ただし、ここまで手前に表示するのは、例えば、画像ビューワーのスライドショーや動画の全画面再生のような用途に限られると思います。

NSWindow : ウィンドウのレベルを変更

書式
```
- (void) setLevel : (int) newLevel
```
入力
```
newLevel : ウィンドウのレベル
```

ヘッダーを見ると、ウィンドウのレベルには以下のようなものがあります。

NSWindow.h > WindowLevel

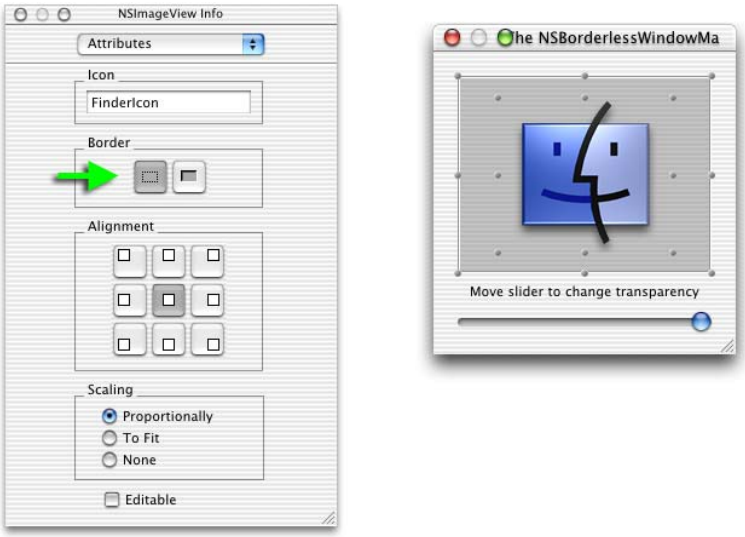
```
NSNormalWindowLevel
NSFloatingWindowLevel
NSSubmenuWindowLevel
NSTornOffMenuWindowLevel
NSMainMenuWindowLevel
NSStatusWindowLevel
NSDockWindowLevel
NSModalPanelWindowLevel
NSPopUpMenuWindowLevel
NSScreenSaverWindowLeve
```

■ 終わりに

前回のムービー再生で動画やMP3も再生出来るようになりましたし、今回のウィンドウの変形でスキンの機能も実現出来るようになりました。これで、面白いプレーヤーソフトが作れるための材料が揃ってきました。その他、時計やカレンダー、デスクトップマスコットなど応用範囲の広いものですので、是非いろいろと試して

みてください。

最後に、1つ手抜きのためのテクニックを紹介します。今回のサンプルでは、CustomViewでウィンドウの中を描画していましたが、NSImageViewを使う手もあります。NSImageViewのBorderを枠なしにすると、NSImageView自身が透明になるため、透明色を含む画像を表示すると、そのままウィンドウも透けるのです。ただし、NSImageViewがイベントを処理してしまってウィンドウがドラッグできなくなるので、その上に、CustomViewを被せておきます。このCustomViewには何もコードは書かなくて構いません。



これを実行すると、以下のようになります。自分で描画をしないのならば、こちらの方が手軽かもしれませんね。