# Cocoa はやっぱり! 出張版 補足

## 2001.11

# 5. Mac OS X 10.1 の新機能 2

## ■ 今回のテーマ

今回は、第 4 回の「 Mac OS X 10.1 の新機能 」で書いていなかった機能について説明を行っていきます。Mac OS X 10.1 の新機能や変更点の詳細は、Apple からのリリースノートに記述されていますので、必要に応じて参照してください。

Apple サイト
　http://developer.apple.com/techpubs/macosx/ReleaseNotes/index.html
ハードディスク上 ( Developer Tools をインストールした場合 )
　/Developer/Documentation/ReleaseNotes/

また、10.0.4 から 10.1 での詳細な変更点が書かれたテクニカルノートも Apple のサイトにアップされましたので、こちらも参考にされるとよいと思います。

　http://developer.apple.com/technotes/tn/tn2029.html

## ■ ライブリサイズ

NSView には、**「 ユーザーが現在ウィンドウのリサイズしているかどうか 」**を知るためのメソッドが追加されました。**inLiveResize** というメソッドです。ウィンドウがリサイズされている最中には、ウィンドウのサイズが変わるたびに、NSView 内を描画するメソッドである drawRect : が繰り返し呼ばれます。

しかしながら、描画処理が重たい場合は、リサイズも重くなり操作性に影響します。そのため、**「 ウィンドウのサイズが確定したときのみ描画したい 」**とか、**「 ライブリサイズ中はラフな描画に切り替えたい 」**ということがあると思います。そういうときには、drawRect : の中で inLiveResize を呼んで判定すればよいのです。

また、ライブリサイズ開始直前と終了直後には、NSView の **viewWillStartLiveResize** と **viewDidEndLiveResize** が呼ばれるようになりました。ライブリサイズ時に別の描画を行う場合、初期化処理や終了処理が必要であればここに記述します。さらに、viewDidEndLiveResize から、[ self setNeedsDisplay : YES ] を実行して、通常の描画を行わせるようにしておきます。

では、ライブリサイズ中には画像を描画しない NSImageView を作ってみましょう。NSImageView のサブクラスとして、NSQuickImageView を作ったとします。このクラスに以下の 2 つのメソッドを実装することで、実現できます。

**NSQuickImageView.m**

```
// 描画が必要なときに呼ばれる

- (void) drawRect : (NSRect) rect {

      if ( [ self inLiveResize ] ) {    // ライブリサイズ中か判定
          [ [ NSColor grayColor ] set ];    // グレーで
          [ NSBezierPath fillRect : rect ]; // 矩形の塗りつぶし
      }
      else
          [ super drawRect : rect ];        // NSImageViewにおまかせ

}

// ライブリサイズ終了直後に呼ばれる

- (void) viewDidEndLiveResize {
    [ self setNeedsDisplay : YES ];  // 通常の再描画を行わせる
}
```

このようにすると、リサイズ中は、グレーに塗りつぶすだけになり、ウィンドウのサイズが確定したところで、通常の画像描画が行われます。実行結果は以下のようになります。

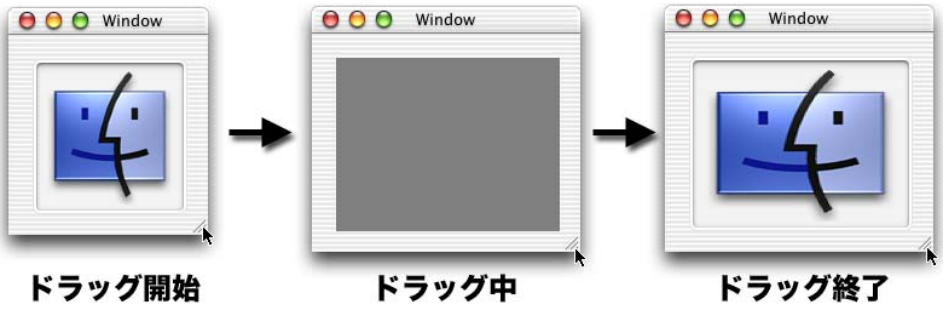


図. ライブリサイズ中の描画を軽くしたときの実行結果

**ライブリサイズ中かを取得**

**書式**
```
- (BOOL) inLiveResize
```
**出力**
返り値 : ライブリサイズ中なら - YES、そうでないなら - NO

**ライブリサイズ開始直前の処理を行う**

**書式**
```
- (void) viewWillStartLiveResize
```

**ライブリサイズ終了直後の処理を行う**

**書式**
```
- (void) viewDidEndLiveResize
```

## ■ 3つ以上のボタンがあるデバイスのサポート

最近のマウスは、色々な種類があり、ボタンが 3 つ以上付いているものも多くなりましたが、これらのサポートも簡単に行えるようになりました。

3 つ目以降のボタンのイベントを受け付けるために、NSResponder クラス ( イベントを受け付けるクラス ) に、**otherMouseDown :** 、**otherMouseUp :** 、**otherMouseDragged :** というメソッドが追加されました。これらのメソッドは、3 つ目以降のボタンの操作が行われたときに呼び出されます。通常は、NSView のサブクラスにこれらのメソッドを書いてイベント処理を行うことになります。

**3つ目以降のボタンが押された**

**書式**
```
- (void) otherMouseDown : (NSEvent *) theEvent
```
**入力**
```
theEvent : イベント情報
```

**3つ目以降のボタンが放された**

**書式**
```
- (void) otherMouseUp : (NSEvent *) theEvent
```
**入力**
```
theEvent : イベント情報
```

**3つ目以降のボタンでドラッグされた**

**書式**
```
- (void) otherMouseDragged : (NSEvent *) theEvent
```
**入力**
```
theEvent : イベント情報
```

3 つ目以降のどのボタンの操作なのかは、パラメータで受け取った NSEvent の **buttonNumber** というメソッドを使うことで判別できます。左のボタンが 0 で、右のボタンが 1、3 番目のボタンが 2 というように値が増えていきます。

マウスの種類に依存するかもしれませんが、最近よくあるホイールがクリックできるマウスの場合は、これが 3 つ目のボタンになるようです。

**どのボタンのイベントかを取得**

**書式**
```
- (int) buttonNumber
```
**出力**
```
int : ボタンの番号 ( 左ボタン – 0、右ボタン – 1、3つ目のボタン – 2、... )
```

これにともなって、イベントの種類を表す定数や、イベントのマスクを行うための定数も増えています。

**イベントの種類を表す定数**

```
typedef enum _NSEventType {
      NSLeftMouseDown         = 1,
      NSLeftMouseUp           = 2,
        : 省略
      NSScrollWheel           = 22,
      NSOtherMouseDown        = 25,
      NSOtherMouseUp          = 26,
      NSOtherMouseDragged     = 27
} NSEventType;
```

**受け付けるイベントを制限するための定数**

```
enum {
    NSLeftMouseDownMask      = 1 << NSLeftMouseDown,
    NSLeftMouseUpMask        = 1 << NSLeftMouseUp,
        : 省略
    NSScrollWheelMask        = 1 << NSScrollWheel,
    NSOtherMouseDownMask     = 1 << NSOtherMouseDown,
    NSOtherMouseUpMask        = 1 << NSOtherMouseUp,
    NSOtherMouseDraggedMask  = 1 << NSOtherMouseDragged,
    NSAnyEventMask           = 0xffffffffU
};
```

## ■ 直前のイベントからのマウスの移動量

NSEvent の deltaX と deltaY では、マウス移動やマウスドラッグのイベントで直前のイベントからの移動量が取得できるようになりました。NSView のサブクラスに以下のようなコードを書くことで移動量を得ることが出来ます。

**直前のイベント処理からのドラッグ量を取得するサンプル**

```
- (void) mouseDragged : (NSEvent *) theEvent {

    NSLog( @"deltaX = %f", [ theEvent deltaX ] );
    NSLog( @"deltaY = %f", [ theEvent deltaY ] );

}
```

## ■ アバウトパネル

Cocoa アプリケーションでは、著作権等を表示するアバウトパネルが予め用意されていて、何もしなくてもプロジェクトの設定などにしたがって以下のように表示されます。


図. 通常のアバウトパネル

Mac OS X 10.1 では、**「 Credits.html 」**というファイルをリソースとして組み込んでおくと、このファイルの中身がアバウトパネル内に表示されます ( **「 Credits.rtfd 」**や**「 Credits.rtf 」**も可能 )。

アバウト画面に**サポートページへのリンクを付ける**のもウェブページを作るのと同じ要領で**リンクを作っておくだけ**です。**「 mailto: 」も使えます**ので、メールアドレスを組み込むのも簡単です。クリックするとデフォルトのメーラーが起動して、メール作成画面になります。また、この HTML には、画像を貼り付けておけば、それも表示されます。

アプリケーションへの組み込みは、Project Builder で、プロジェクトウィンドウの Resources の下に画像を含めた全てのファイルをドラッグ&ドロップしてビルドするだけです ( 言語毎にファイルを組み込むこともちろん可能です )。

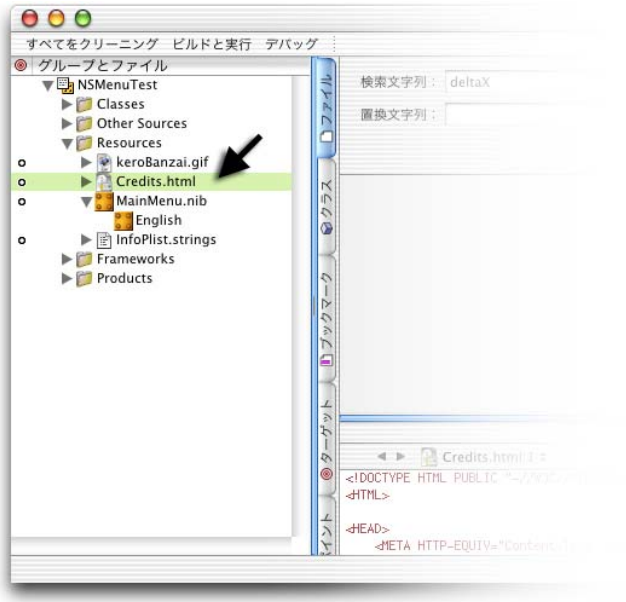


図. アバウトパネル用の HTML ファイルのプロジェクトへの組み込み

組み込んだ結果ですが、中身が短い場合は左のようになり、長い場合はスクロール表示されるようになります。

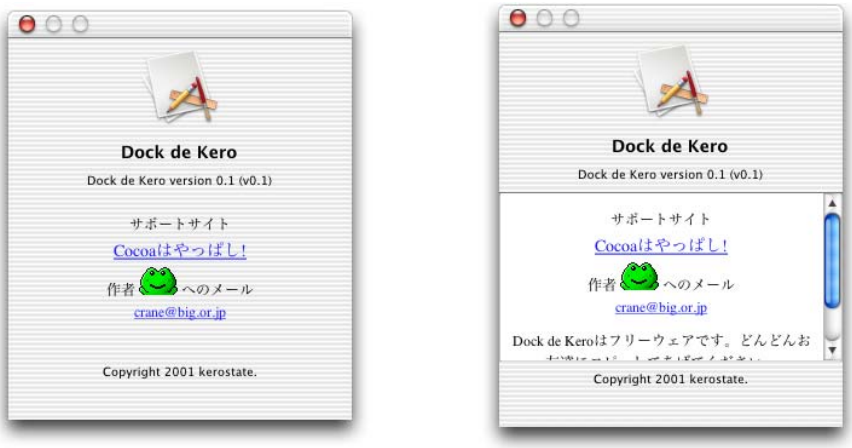


図. HTML を組み込んだ場合のアバウトパネル

## ■ NSURL / NSURLHandle

NSURL/NSURLHandle は、**HTTPS をサポート**したので、暗号化を行うサーバーとのコネクションが可能になりました。

また、サーバーへ送信する HTTP ヘッダーを加工することができるようになっています。NSURLHandle の **writeProperty : forKey :** でユーザーエージェントの指定などを追加できますし、Basic 認証のかかっているページへのアクセスも可能になります。

**HTTPヘッダーを加工するサンプル**

```
NSURL       *url    = [ NSURL URLWithString : @"http://www.apple.com/" ];
NSURLHandle *handle = [ url URLHandleUsingCache : NO ];
NSData      *pageData;

[ handle writeProperty : @"MyApp/1.0"
                forKey : @"User-Agent" ];     // ユーザーエージェント
[ handle writeProperty : @"Basic xxxx"
                forKey : @"Authorization" ];  // Basic認証のページへのアクセス

pageData = [ handle resourceData ];
```

ただし、私の実験では、あいかわらず HTTP のプロキシーサーバーはサポートを行っていないようです。システム環境設定のネットワークの設定には従いますが、プロキシーの設定は無効ですので注意が必要です。プロキシーも考慮する場合は、NSURL は現状使えません。

## ■ マルチスクリーンサポート改善

スクリーンの情報を扱う NSScreen は、マルチスクリーンのサポートが改善されました。スリープから復帰した際に、スクリーンが追加されたり削除されたりすることがありますが、これを正しく扱うようになっています。このため、NSScreen から得た情報をアプリケーション側に長期的にコピーを持つことはお奨めできません。画面の情報が必要な場合は、その時点で NSScreen のメソッドを使って参照するという方法にしなければなりません。

## ■ クリップボード ( ペーストボード ) への画像の出力

私の見落としでなければ、リリースノートには書いていないのですが、クリップボードへの画像の出力に関しても、Mac OS X 10.1 では改善が行われています。ただし、これはフレームワーク側の改善ではなく、OS 側の改善なのかもしれません。

Cocoa では、以下のコードで画像データをクリップボードへ出力できます。

**画像をクリップボードに出力するサンプル**

```
NSImage      *img = [ 画像を生成するメソッド ];
NSPasteboard *pb  = [ NSPasteboard generalPasteboard ];

[ pb declareTypes : [ NSArray arrayWithObjects : NSTIFFPboardType, nil ]
            owner : nil ];
[ pb setData: [ img TIFFRepresentation ] forType : NSTIFFPboardType ];
```

このコードでは、NSImage から TIFF 形式で画像データを取得して、クリップボードに出力しています。しかしながら、Carbon や Classic アプリケーションは、PICT 形式でないと画像を受け取れないため、Mac OS X 10.0.4 まででは、Cocoa アプリケーション間でしか画像のやり取りはできませんでした ( Cocoa アプリケーションで、PICT 形式のデータを作ることは不可能ではありませんが、やや面倒なのです )。

それが、Mac OS X 10.1 では、TIFF で書き出すだけで、クリップボードには PICT 形式のデータも自動的に組み込まれるようになりました。そのため、先程のコードのままで、Cocoa アプリケーションから Carbon や Classic アプリケーションへ画像をクリップボード経由で渡すことはができるようになりました。

NSImage クラスに PICTRepresentation というメソッドがあってもいいのではないかと思うんですけどね。

## ■ まとめ

このように Cocoa のフレームワークは、小幅ながらも役に立つ改良が行われています。是非あなたのアプリケーションでも使ってみてください。