Cocoa はやっぱり! 出張版

2001.10

4. Mac OS X 10.1 の新機能

■ 今回のテーマ

Mac OS X 10.1 がついに 2001.9.29 にリリースされました。既に手にされている方も多いかと思います。そこで今回は、Mac OS X 10.1 のパッケージに添付の新しい Developer Tools や Foundation や Application Kit フレームワークの新機能について解説をしたいと思います。細かいところまで含めるとかなりの量になりますので、全部はフォローできませんが、重要だと思われる箇所や興味深い箇所についてピックアップして説明を行っていきます。詳細は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/

■ Project Builder の新機能

クラスブラウザー

Project Builder の見た目で一番変化が大きいのは、プロジェクトウィンドウの左側に追加された**クラスタブ**です。ここを開くと**クラスブラウザー**が表示されます。

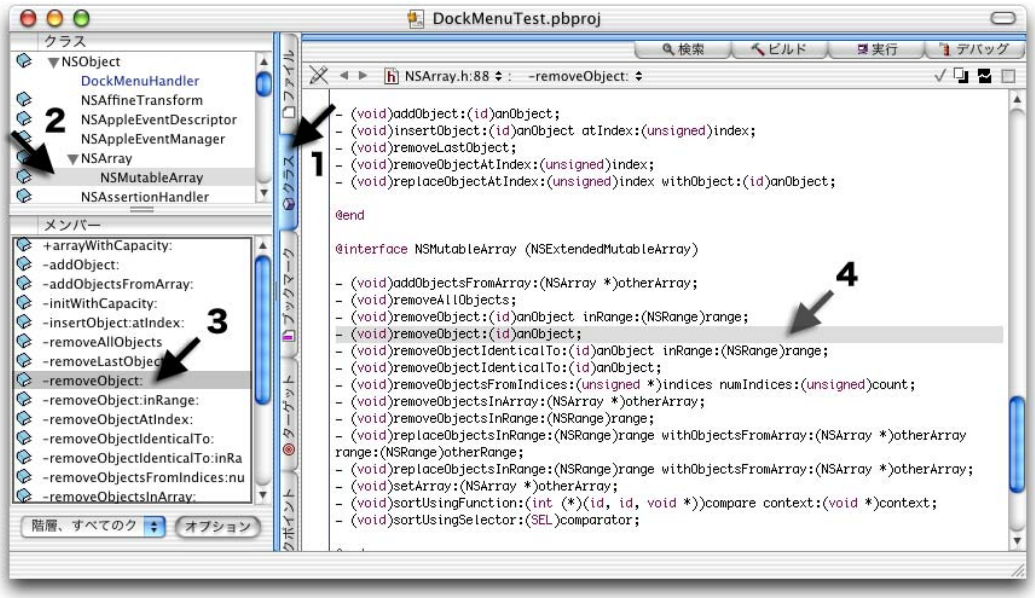

図. Project Builder のクラスブラウザーの操作手順

クラスタブをクリックすると、クラスブラウザーが表示されます…**1**。上部には、クラスツリーが表示され、階層構造を見ることが出来ます。一番下にあるポップアップメニューやオプションボタンで表示形式を変えることが出来ます。フレームワークのクラスだけにしたり、プロジェクトで定義しているクラスだけにしたりということが可能です。そして、クラス名をクリックすると下部には、そのクラスのメソッドが表示されます…**2**。さらに、メソッド名をクリックすることで、フレームワーク内のクラスならば、ヘッダーファイルの宣言の箇所が、自前のクラスの場合は、ソースファイルの実装箇所が表示されます…**3,4**。ダブルクリックすると、単独のウィンドウで開くことができます。

クラスブラウザーの中に小さい本の形をしたアイコンがありますが、これをクリックすると…**1**、リファレンスが表示されます…**2**。

図. クラスブラウザーからリファレンスを呼び出したところ

メソッド一覧メニュー

ソースを表示するウィンドウの上部に、**メソッド等の一覧がメニューで表示される**ようになりました。選択すると、定義してある場所へスクロールします。ヘッダーファイルでも同様に使用できます。

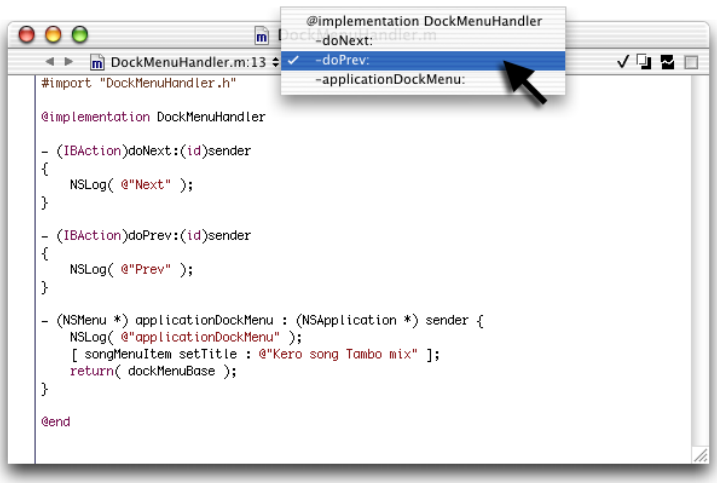

図. メソッド一覧メニューを表示したところ

#pragma mark を使うと、メソッドの先頭だけではなく、ソース中の特定の場所にジャンプすることができ

るようになります。

```
#pragma mark LABELNAME
```

のように書いておくと…**1**、mark 以降の文字がメニューに表示されるようになり、ジャンプできるようになります…**2**。

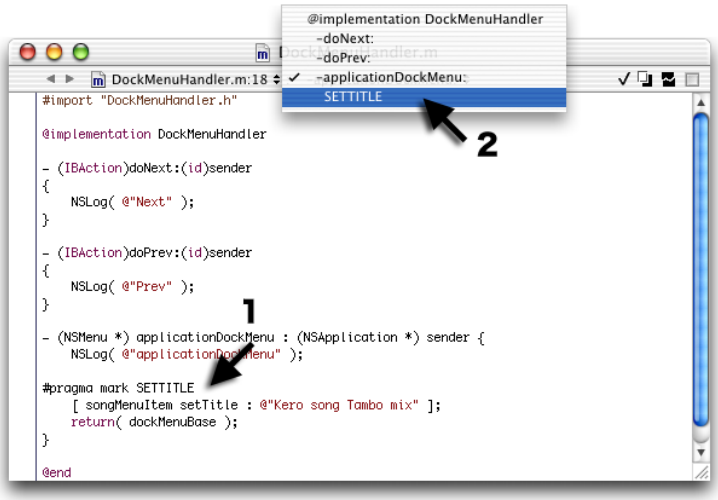

図. #pragma mark を使ってメニューに SETTITLE を登録したところ

ソースコードからコマンド＋ダブルクリックで検索

Cocoa のフレームワークのメソッド名は、かなり長いものが多くあります。それを記憶するのは大変なので、よくヘッダーからコピーをしてきますが、ヘッダーの該当個所まで行くのもちょっと大変です。新しい Project Builder では、途中まで入力して、**コマンドキーを押したままその文字列をダブルクリックしますと、その箇所を検索文字列として検索を行います。検索結果の数によって表示の仕方が変わりますが、数が少なめの場合は、メニューがポップアップして一覧が表示**されます。これを選ぶことで定義箇所にジャンプできますので、そこからコピーをするとよいでしょう。

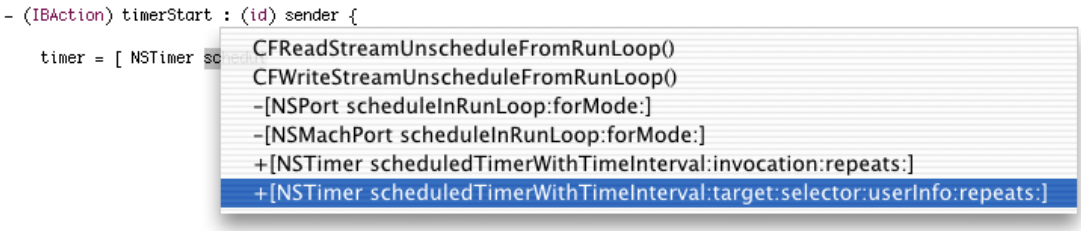

図. sched と入力して、その文字列をコマンド+ダブルクリックしたところ

Objective-C++対応

コンパイラは **Objective-C++に対応**しました。これで、Objective-C と C++を混ぜて書くことが出来るようになりました。C++の資産がある方は、ライブラリなどを流用できるようになります。

■ Interface Builder の新機能

未接続の警告マーク

Interface Builder は、ドキュメントウィンドウ内にあるインスタンスで、**アクションやアウトレットに未接続のものがあると警告マークが表示される**ようになりました。ウィンドウの中に配置されているビューに未接続のものがある場合は、ウィンドウのインスタンスに警告マークが表示されます。この機能によって、接続忘れが減ることでしょう。

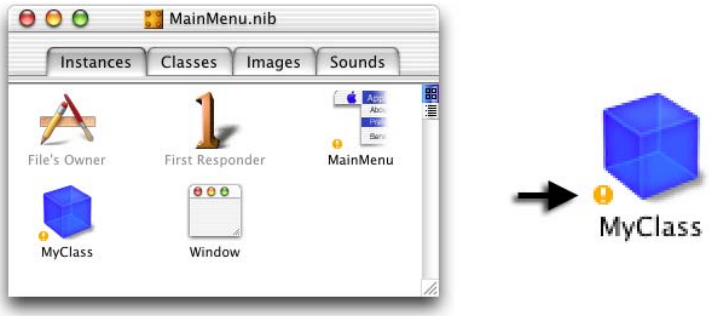

図. 未接続のアクションやアウトレットがあるときの警告マーク

クラスツリーのカラム表示

クラスタブには、従来のリスト表示に加えて、**Finder と同様なカラム表示が追加**されました。クラスの数が多いこともあり、リスト表示よりもスクロールの量が減るため、こちらの方が使いやすいのではないかと思います。

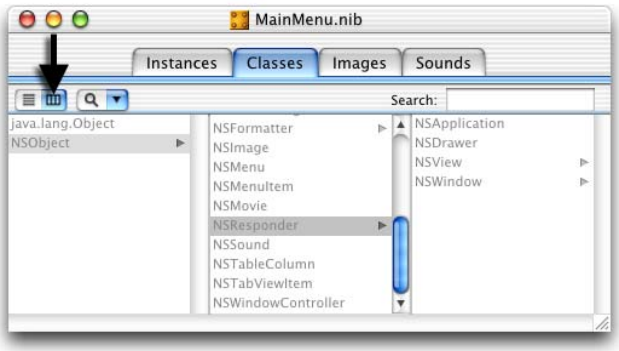

図. クラスツリーのカラム表示

クラスのサーチ機能

さらに、クラス表示画面の右上の Search と書かれているところに文字列を入力すると、**入力した文字列と前方一致するクラスへジャンプ**してくれるようになりました。例えば、「 nso 」と入力すると「 NSObject 」が選択されるようになります。大文字／小文字の違いは無視されます。

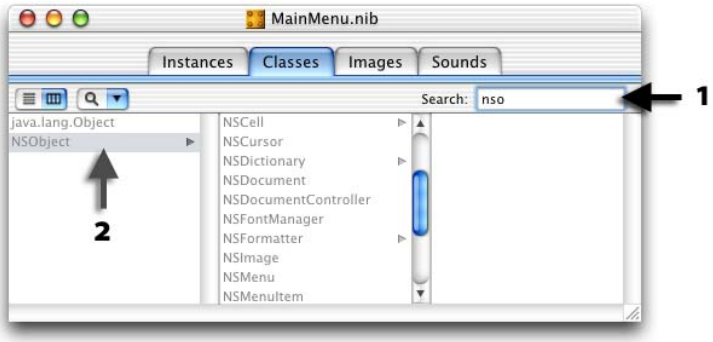

図. クラスのサーチを行ったところ

インフォパネルでのクラス編集

クラスの編集は、インフォパネルで行うようになりました。今まではクラスツリーの中でクラスの中身を編集するということで、ややごちゃごちゃとした中での編集でした。別ウィンドウになりましたので、こちらの方がシンプルで使いやすいのではないかと思います。アクションやアウトレットを追加するには、「 + 」と書かれているボタンをクリックします。削除は、「 - 」ボタンで行います。

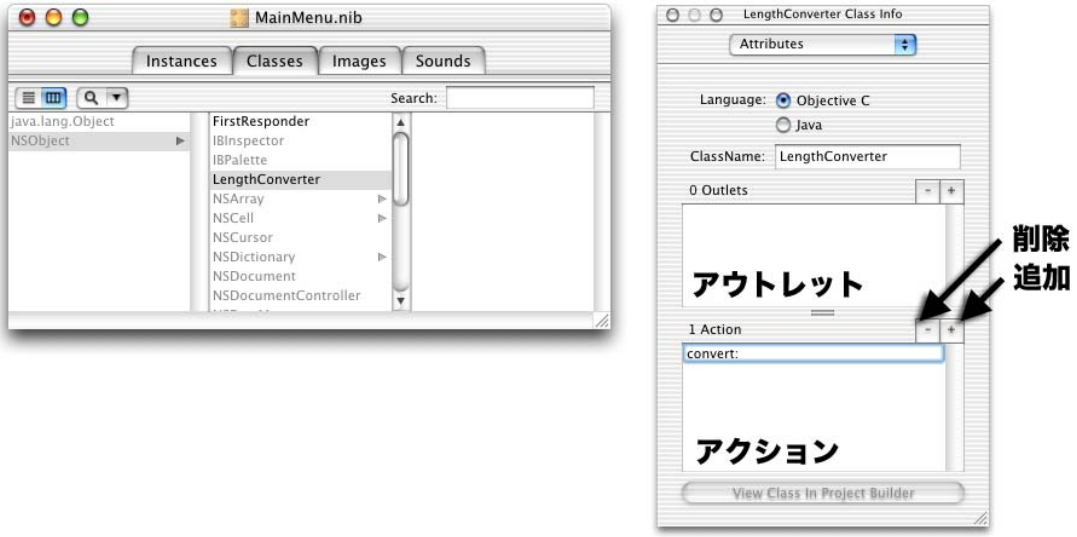

図. インフォパネルでのクラスの編集画面

ソース出力時のファイルマージ

Interface Builder には、作成したクラスのヘッダーとソースの雛形を出力する機能がありますが、クラスに変更が加わって、再度出力をするとき、今までは完全な上書きしか出来ませんでした。出力した雛形に対して手で編集を加えてしまっている場合は、上書きされると困るため面倒なことになっていました。新しい Interface Builder では、出力時に以下のアラートが表示されて、上書きだけでなく、**ファイルマージ**が選べるようになりました。

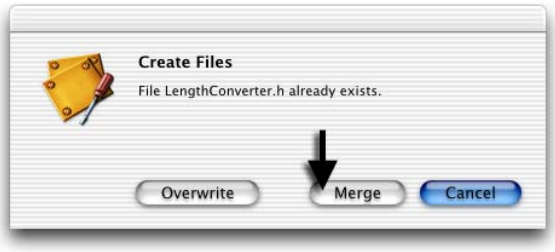

図. ファイル出力時の確認アラート

Merge ボタンをクリックすると、FileMerge アプリケーションが起動します。古いファイルとの相違点を表示してくれて、違っている箇所毎に、どちらのソースを新しいソースへ引用するかを指定できるようになっています。

■ Application Kit の新機能

ドックによる通知

バックグラウンドで動いているアプリケーションから「 エラーが発生した 」や「 メールが着信した 」などのユーザーへの通知がある場合に、**ドック内のアプリケーションアイコンをバウンドさせる**という通知方法が新たに追加されました。以下のような感じになります。

図. ドックの通知でアプリケーションアイコンがバウンドしたところ

NSApplication の **requestUserAttention** : メソッドで、通知を行うことが出来ます。通知の種類には、**一度だけジャンプするもの**と**繰り返しジャンプするもの**の 2 種類があります。

NSApplication : ドックによる通知を行う

```
書式
  - (int) requestUserAttention : (NSRequestUserAttentionType) reqType
入力
  reqType : 通知の種類
出力
  返り値    : 通知の識別番号 ( キャンセルするときに使用する )
```

NSRequestUserAttentionType: ドックによる通知の種類

```
enum {
   NSInformationalRequest = 0,  // 一度だけジャンプ
   NSCriticalRequest      = 1   // 繰り返しジャンプ
} NSRequestUserAttentionType;
```

この通知は、通知を出したアプリケーションがアクティブなった時点で自動的に止まりますが、「 1 分経過したら通知をやめる 」など、強制的に止めたい場合は、**cancelUserAttentionRequest** : メソッドを使います。同時に複数の通知が発生することも考えられますので、そのために、どの通知を止めるのかを指定しなければなりません。requestUserAttention : の返り値が通知の識別番号になりますので、それを記憶しておいて、中止するときに使います。

NSApplication : ドックによる通知を中止する

書式

```
- (void) cancelUserAttentionRequest : (int) request
```

入力

request : 中止したい通知の識別番号

使用例を以下に書きます。**NSApp というのは、NSApplication クラスのインスタンスでアプリケーション自身を指しています**。Application Kit のグローバル変数なので、アプリケーションのどこからでも使えます。

```
int iReqNo = [ NSApp requestUserAttention : NSCriticalRequest ]; // 通知開始
  : 何か処理をする
[ NSApp cancelUserAttentionRequest : iReqNo ]; // 通知中止
```

また、バックグラウンドアプリケーションが、**アラートなどのモーダルパネルを表示した場合は、自動的に requestUserAttention : が呼ばれます**ので、通知の処理は不要です。

アプリケーション独自のドックメニュー

Mac OS X 10.0.4 までは、ドックのアプリケーションのアイコンからポップアップされるメニューの中身は、予め決められたものしか表示できず、独自のメニューを表示することは出来ませんでした。Mac OS X 10.1 では、アプリケーションが独自のメニューを追加することが出来るようになり、ドックからアプリケーションをコントロールできるようになりました。

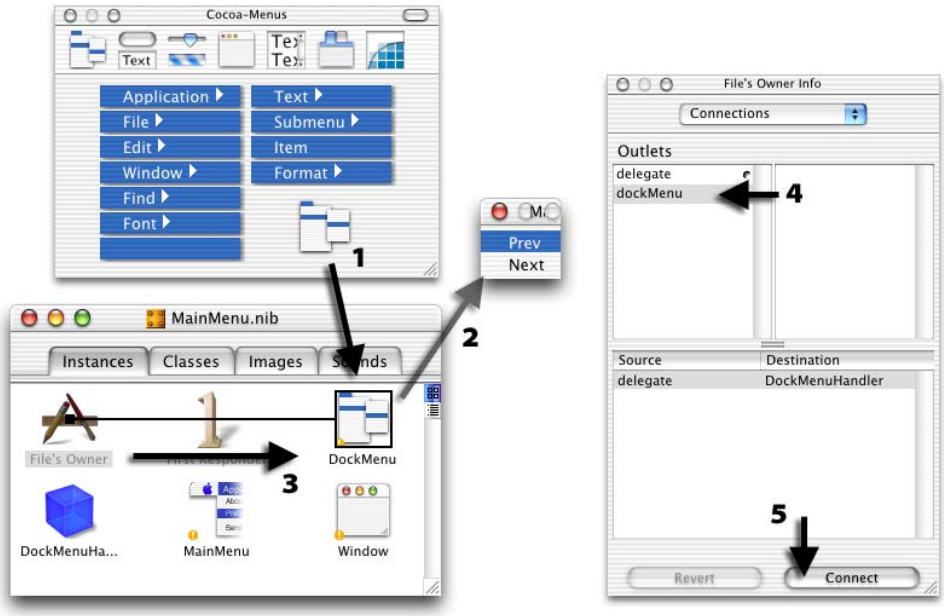

図. メニューをドックに登録する手順

パレットウィンドウから MainMenu.nib へメニューをドロップします…**1**。すると、メニュー編集ウィンドウが表示されますので、メニュー項目のタイトルを「 Prev 」と「 Next 」と変えます…**2**。File's owner (この場合は `NSApplication`) には dockMenu というアウトレットが増えていますので、ここに今作ったメニューを接続します…**3,4,5**。これで、ドックのメニューに割り当てられます。それぞれのメニュー項目に、実行したいアクションを接続すれば、準備完了です。実行すると以下のようになります。Prev と Next がメニューに組み込まれているのが分かります。

図. ドックのメニューを表示したところ

メニューの中身が常に変わらない場合は、これで終わりですが、ダイナミックにメニューの中身を変えたい場合もあります。この場合は、アプリケーションのデリゲートのインスタンスに **`applicationDockMenu :`** メソッドを書いておきます。このメソッドは、**ドックのメニューがポップアップする直前に呼ばれます**ので、表示したいメニューを作成して返り値として返します。そうすると、返したメニューがドックに表示されます。

```
- (NSMenu *) applicationDockMenu : (NSApplication *) sender {
    [ songMenuItem setTitle : @"Kero song Tambo mix" ];
    return( dockMenuBase );
}
```

このサンプルでは、メニューをゼロからは作成していません。予め以下のようなメニューを Interface Builder 上で作って、dockMenuBase というアウトレットで参照できるようにしておきます。さらに、一番下のメニュー項目を songMenuItem という名前のアウトレットとして参照できるようにして、このメニュー項目のタイトルを書き換えて、dockMenuBase を返り値としています。メニューの項目数が変わらない場合は、こういうやり方がお手軽でしょう。

図. ダイナミックなメニューのための雛形とそのアウトレットの名前

実行結果は以下のようになります。Mac OS X 10.1 に付属の iTunes にもこのようなメニューが表示されますが、このような方法で実現できるわけです。

図. ダイナミックなドックのメニューを表示したところ

音声合成

NSTextView には、音声合成の機能がつきました。テキストを選択してコンテキストメニューを表示すると、「 スピーチ 」メニューが表示されます。

図. コンテキストメニューから音声合成を行うところ

この音声合成の機能をアプリケーション側から利用する方法を説明します。やり方としては、かなり邪道かもしれませんが、手軽にできる方法です。画面に表示しないウィンドウを 1 つ作って NSTextView を配置して、しゃべって欲しい文字列をセットしてから、「 **startSpeaking :** 」メソッドを呼び出すのです。

textView というアウトレットを作って、その NSTextView を接続して、以下のようにするだけです。「 My name is Kero. 」としゃべってくれるはずです。

NSTextViewに文字列をしゃべらせるサンプル

```
[ textView setString : @”My name is Kero.” ];
[ textView startSpeaking : self ];
```

■ Foundation の新機能

ファイルタイプとクリエータ

Mac OS X 10.0.4 までの二大フレームワークでは、ファイル属性のファイルタイプとクリエータに関しては、サポートがあまりなされていませんでした。そのために、Carbon API を呼ぶ必要がありました。10.1 になってようやく NSFileManager クラスでサポートが行われるようになりました。ファイルの属性を取得するメソッドを使うことで、ファイルタイプとクリエータの情報も同時に取得できます。

NSFileManager : ファイル属性を取得する

書式

```
- (NSDictionary *) fileAttributesAtPath : (NSString *) path
                            traverseLink : (BOOL      ) yorn
```

入力

path : 属性を取得したいファイルのパス
yorn : リンクを辿るなら – YES、辿らないなら - NO

出力

返り値 : ファイルの属性が格納された辞書

このように、返り値が辞書になっていますので、ファイルの属性が沢山詰まった辞書のインスタンスが返ってくることが分かります。以下のようにすると、指定のファイルパスのファイル属性が得られます。

filePathに存在するファイル属性の辞書の表示

```
NSLog( @”%@”, [ [ NSFileManager defaultManager ]
                  fileAttributesAtPath : @” filePath “
                          traverseLink : YES ] );
```

実行結果は以下のようになります。

ファイル属性の辞書のサンプル

```
2001-XX-XX 16:31:55.374 FileTypeCreator[428] {
    NSFileExtensionHidden = 0;
    NSFileGroupOwnerAccountName = staff;
    NSFileHFSCreatorCode = 943868237;
    NSFileHFSTypeCode = 1246774599;
    NSFileModificationDate = 2000-12-08 19:40:46 +0900;
    NSFileOwnerAccountName = crane;
    NSFilePosixPermissions = 511;
    NSFileReferenceCount = 1;
    NSFileSize = 4003;
    NSFileSystemFileNumber = 282823;
    NSFileSystemNumber = 234881033;
    NSFileType = NSFileTypeRegular;
}
```

様々な属性があることが分かりますが、この中でファイルタイプとクリエータに当たるのが、**NSFileHFSTypeCode** と **NSFileHFSCreatorCode** です。この 2 つは、ファイルタイプとクリエータを辞書から取り出したり、辞書へ書き込んだりするときに使用するキーになります。

ファイルタイプとクリエータは、4 文字の文字列として見ることが多いですが、ここでは、unsigned long の形式で数値として表記されています。文字に変換するとファイルタイプが「 JPEG 」で、クリエータが「 8BIM (Adobe Photoshop) 」になります。この辞書からファイルタイプとクリエータを取り出す専用のメソッドも用意されています。

NSDictionary : 辞書からファイルタイプを取得する

書式

```
- (OSType) fileHFSTypeCode
```

出力

返り値 : ファイルタイプ

NSDictionary : 辞書からクリエータを取得する

書式

```
- (OSType) fileHFSCreatorCode
```

出力

返り値 : ファイルタイプ

OSType という型の実体は unsigned long で、そのまま Cocoa で扱う場合は不便なため、文字列との相互変換を行う関数が用意されています。

OSTypeからNSStringに変換

書式
```
NSString *NSFileTypeForHFSTypeCode( OSType hfsFileTypeCode )
```
入力
hfsFileTypeCode : NSStringに変換したいOSType
出力
返り値 : 変換されたOSType。前後にシングルクォートがつく (例'TEXT')。

NSStringからOSTypeに変換

書式
```
OSType NSHFSTypeCodeFromFileType( NSString *fileTypeString )
```
入力
fileTypeString : OSTypeに変換したいNSString
前後にシングルクォートが必要 (例'TEXT')
出力
返り値 : 変換されたNSString

以上で道具がそろいましたので、指定のファイルパスからファイルタイプとクリエータを NSString で取得するサンプルコードを以下に書きます。

ファイルパスからファイルタイプとクリエータをNSStringで取得

```
NSFileManager *fm  = [ NSFileManager defaultManager ];
NSDictionary  *dic = [ fm fileAttributesAtPath : filePath
                                  traverseLink : YES ];

NSLog( @"%@", NSFileTypeForHFSTypeCode( [ dic fileHFSTypeCode ] ) );
NSLog( @"%@", NSFileTypeForHFSTypeCode( [ dic fileHFSCreatorCode ] ) );
```

ファイル属性を書き込む以下のメソッドも、もちろんファイルタイプとクリエータに対応しています。

NSFileManager : ファイル属性を変更する

書式
```
- (BOOL) changeFileAttributes : (NSDictionary *) attributes
                       atPath : (NSString     *) path
```
入力
attributes : 変更したいファイルの属性
path : 属性を変更したいファイルのパス
出力
返り値 : 成功 - YES、失敗- NO

ファイルタイプとクリエータを変更するサンプルは、以下のようになります。

ファイル属性を変更するサンプル

```
// ファイル属性を取得
NSString      *filePath = @”ファイルパス”;
NSFileManager *fm  = [ NSFileManager defaultManager ];
NSDictionary  *dic = [ fm fileAttributesAtPath : filePath
                                  traverseLink : YES ];
// ファイル属性をコピー
NSMutableDictionary *dicNew = [ NSMutableDictionary
                                  dictionaryWithDictionary : dic ];

// OSTypeを準備
OSType fileType = NSHFSTypeCodeFromFileType( @"'ImgV'" );
OSType creator  = NSHFSTypeCodeFromFileType( @"'JPEG'" );

// NSNumberを準備
NSNumber *numFileType = [ NSNumber numberWithUnsignedLong : fileType  ];
NSNumber *numCreator  = [ NSNumber numberWithUnsignedLong : creator ];

// ファイル属性辞書を変更
[ dicNew setObject : numFileType
            forKey : NSFileHFSCreatorCode ];
[ dicNew setObject : numCreator
            forKey : NSFileHFSTypeCode ];

// ファイル属性を変更
[ fm changeFileAttributes : dicNew
                   atPath : filePath ];
```

リソースフォークの扱いの改善

MacOS X 10.0.4 までは、NSFileManager に存在しているファイルコピーなどのメソッドは、リソースフォークを無視していました。コピーするとリソースフォークが落ちてしまい、事実上使いものにならなかったのですが、Mac OS X 10.1 でようやくリソースフォークにも対応してくれました。以下に、ファイル移動のサンプルだけ書いておきます。ファイル操作については、NSFileManger に色々と用意されていますので、リファレンスを参照ください。

ファイル移動のサンプル

```
NSFileManager *fm   = [ NSFileManager defaultManager ];
BOOL bResult;

NSString *srcPath = @”移動元”;
NSString *dstPath = @”移動先”;
bResult = [ fm movePath : srcPath
                 toPath : dstPath
                handler : nil ];  // 移動
```

■ 拡張子の扱い

拡張子に関するガイドライン

Mac OS X 10.1 で、ファイルの拡張子の扱いについてのガイドラインが定義されました。**全てのファイルに拡張子を表示するかどうかのフラグが、ファイル属性として追加されています**。画面にファイル名やファイルパスを表示する際には、このフラグを参照して表示する必要があります。多くのアプリケーションは、このための修正が必要になるでしょう。拡張子のガイドラインは以下のところにありますので、参照ください。

Apple サイト
　http://developer.apple.com/techpubs/macosx/ReleaseNotes/FileExtensionGuidelines.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/FileExtensionGuidelines.html

このガイドラインに沿っていないと、**ファイル名の表示が Finder と異なる**という状況になり、ユーザーに混乱を与えかねませんし、Mac OS X 10.1 対応と謳うためには、必須のガイドラインと言えます。

基本的に**全てのファイルには拡張子を付ける**ことになります。これは、他のプラットホームとのファイル交換のことを考えたためです。しかし、なるべく**ユーザーに拡張子の存在を意識させないようにする**ために、ユーザーがファイル名を入力した時に、拡張子を入力しなかった場合は、拡張子を隠すように属性をコントロールした上で、そのファイルのフォーマットに応じた拡張子を付けます。逆に、ユーザーが意識的に拡張子を入力した場合は、拡張子はそのまま表示します。どちらにしても拡張子は付くわけですが、**入力したままが表示される**のというのが方針で、これを Apple は**「 What you see is what you type 」**と呼んでいます。

ファイル名の表示

What you see is what you type を実現するためには、ファイル名を表示するときに拡張子フラグを参照しないといけませんが、簡単な仕組みが用意されています。**displayNameAtPath :** というメソッドが NSFileManager に用意されていて、ファイルパスを与えると、表示用のファイル名を得ることが出来ます。

```
NSFileManager *fm       = [ NSFileManager defaultManager ];
NSString      *dispName = [ fm displayNameAtPath : @" filepath " ];
```

NSFileManager : 表示用のファイル名を取得

書式
```
- (NSString *) displayNameAtPath : (NSString *) path
```
入力
　path　: 表示用のファイル名を得たいファイルのフルパス
出力
　返り値 : 表示用のファイル名 (パスは含まない)
備考
　"/Users/crane/Pictures/photo.jpg" → "photo.jpg" or "photo"

セーブパネルでの拡張子の扱い

拡張子のガイドラインはセーブパネルでの動作についても取り決めをしてあります。ファイルを保存する際に、ユーザーに拡張子を隠すかどうかを選択できるようにするものです。以下の図のように、セーブパネルにチェックボックスを 1 つ追加します。

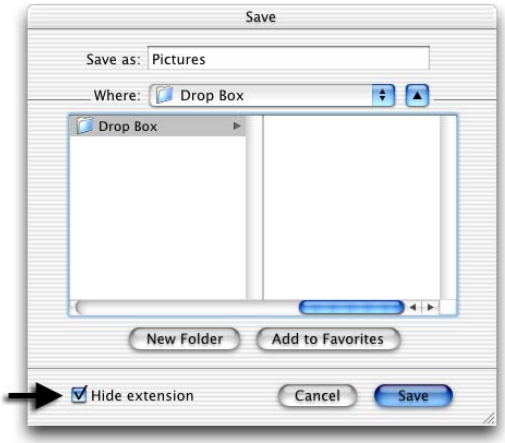

図. セーブパネルの拡張子を隠すためのチェックボックス

このためには、NSSavePanel の **setCanSelectHiddenExtension :** というメソッドを使います。

NSSavePanel : 拡張子表示のチェックボックスの表示状態を変更する

書式

```
- (void) setCanSelectHiddenExtension : (BOOL) flag
```

入力

flag : 拡張子のチェックボックスを表示する – YES、しない - NO

出力

返り値 : ファイルタイプ文字列

そして、セーブパネルを表示したら戻ってきたところで、**isExtensionHidden** メソッドで、チェックボックスの状態がどうなっているかを取得します。

NSSavePanel : 拡張子表示のチェックボックスの状態を取得する

書式

```
- (BOOL) isExtensionHidden
```

出力

返り値 : 拡張子を隠す – YES、隠さない - NO

サンプルは以下のようになります。

```
NSSavePanel *sp = [ NSSavePanel savePanel ];
int         iSts;
BOOL        bHidden;

[ sp setCanSelectHiddenExtension : YES ]; // チェックボックス表示
iSts = [ sp runModalForDirectory : NSHomeDirectory()
                            file : @"Pictures" ];

bHidden = [ sp isExtensionHidden ]; // チェックボックスの状態取得
```

属性の取得と変更

この処理の続きとしてファイル保存があるわけですが、拡張子の属性を保存したファイルにセットする必要があります。要領は、ファイルタイプやクリエータと同様です。先程のファイル属性の辞書の先頭を見ると、**NSFileExtensionHidden** という項目があります。これが、拡張子を隠すかどうかの属性です。値は、論理型なので numberWithBool : を使って NSNumber を生成します。

ファイル属性の辞書のサンプル

```
2001-XX-XX 16:31:55.374 FileTypeCreator[428] {
    NSFileExtensionHidden = 0;
    NSFileGroupOwnerAccountName = staff;
      : 省略
```

拡張子を隠すかどうかの属性を変更する

```
// ファイル属性を取得
NSFileManager *fm  = [ NSFileManager defaultManager ];
NSDictionary  *dic = [ fm fileAttributesAtPath : filePath
                                  traverseLink : YES ];
// ファイル属性をコピー
NSMutableDictionary *dicNew = [ NSMutableDictionary
                                  dictionaryWithDictionary : dic ];

// NSNumberを準備
NSNumber *extHidden = [ NSNumber numberWithBool : YES or NO  ];

// ファイル属性辞書を変更
[ dic setObject : extHidden
         forKey : NSFileExtensionHidden ];

// ファイル属性を変更
[ fm changeFileAttributes : dicNew
                   atPath : filePath  ];
```

拡張子のフラグをファイル属性の辞書から取得するメソッドも用意されています。

NSDictionary : ファイル属性の辞書から拡張子のフラグを取得

書式

```
- (BOOL) fileExtensionHidden
```

出力

返り値 : 拡張子を隠す – YES、隠さない - NO

拡張子を隠すかどうかの属性を取得する

```
// ファイル属性を取得
NSFileManager *fm  = [ NSFileManager defaultManager ];
NSDictionary  *dic = [ fm fileAttributesAtPath : filePath
                                 traverseLink : YES ];
// フラグを取得
BOOL bHidden = [ dic fileExtensionHidden ];
```

■ まとめ

Mac OS X 10.1 において、ドックによる通知や拡張子のガイドラインなど、ユーザーインターフェイスに関する新たな機能が追加されました。その他、パフォーマンスの向上を始めとして、常用 OS として必要なものが随分とそろってきました。ネイティブアプリケーションがまだ少ないというところが、まだ大きな問題として残っていますが、逆に、基盤は整ってきたわけですし、今こそネイティブアプリケーションの開発をスタートするべき時に来ているのではないかと思います。

また、ここに書いた以外にも、さまざまな機能が追加されていますので、リリースノートや Develolper フォルダーを色々と探ってみると新たな発見があると思います。