Cocoa はやっぱり! 出張版

2001.12

6. ツールバー

■ 今回のテーマ

今回のテーマは、ツールバーです。Apple のサイトにある ToolbarSample の説明をしながら、ツールバーの使い方について解説を行っていきます。このサンプルは、Apple のサイトの以下の URL にありますので、まずはダウンロードしておいてください。

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

◎ 推奨環境

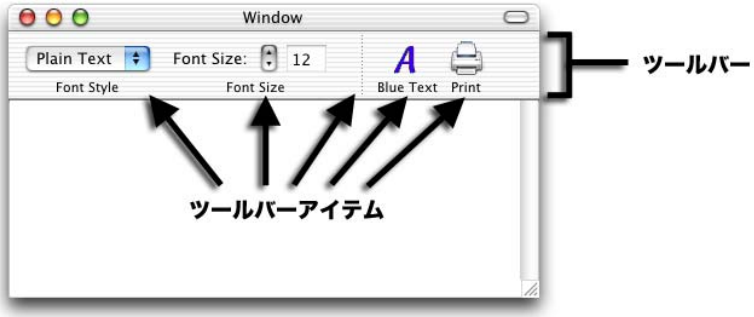
この解説は、以下の環境を前提にしています。これ以外の環境では解説の内容と一致しない部分がありますので、ご確認ください。

・Mac OS X 10.1 以降
・Project Builder 1.1.1 (December 2001 Developer Tools) 以降
・Interface Builder 2.2 (December 2001 Developer Tools) 以降

■ ツールバーの基礎

◎ ツールバーとツールバーアイテム

アプリケーションを起動すると、以下のようなウィンドウが表示されます。ウィンドウの上部にあるのが「**ツールバー (Toolbar)**」で、ツールバーの中にあるボタンなどを「**ツールバーアイテム (Toolbar Item)**」と呼びます。ツールバーは「**`NSToolbar` クラス**」、ツールバーアイテムは「**`NSToolbarItem` クラス**」で実現されています。



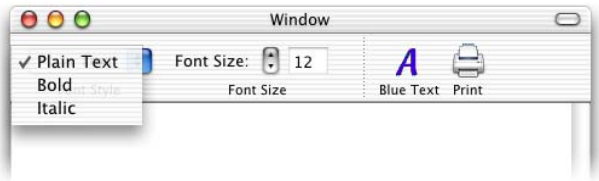
【図】ToolbarSample のウィンドウ

このサンプルの各ツールバーアイテムの機能は、以下のようになっています。Font Size の右にあるセパレータもツールバーアイテムです。

```
Font Style     : 文字修飾 ( Plain Text、Bold、Italic ) をメニューから変更する。
Font Size      : 文字サイズをステッパーで変更する。
セパレータ     : 区切りを表示する。
Blue Text      : 文字色を青と黒に交互に切替える。
Print          : テキストビューを印刷する。
```

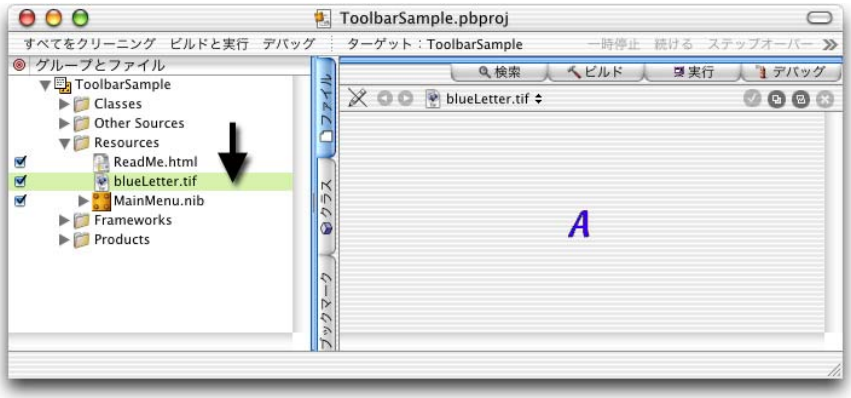
ツールバーの下にあるのは、複数行のテキスト入力ができるテキストビュー (NSTextView) です。このサンプルでは、ツールバーを使ってこの中のテキストを操作します。

ツールバーアイテムには、「 Blue Text 」や「 Print 」のように画像にテキストが付いただけのシンプルなものと、「 Font Style 」や「 Font Size 」のようにちょっと複雑なものがあります。前者を「 **イメージアイテム (Image Item)** 」といい、後者を「 **ビューアイテム (View Item)** 」といいます。



【図】Font Style メニューの中身

イメージアイテムの場合は、アイコン画像のデータファイルをプロジェクトに登録しておきます。このサンプルのプロジェクトファイルを Project Builder で開くと Resources の中に blueLetter.tif というファイルがあります。これが、Blue Text アイテムで使われているアイコンの画像です。**画像は、32×32 ドットサイズ**のものを用意します。これ以外のサイズで作っても強制的にリサイズされて表示されます。



【図】Blue Text のアイコン画像

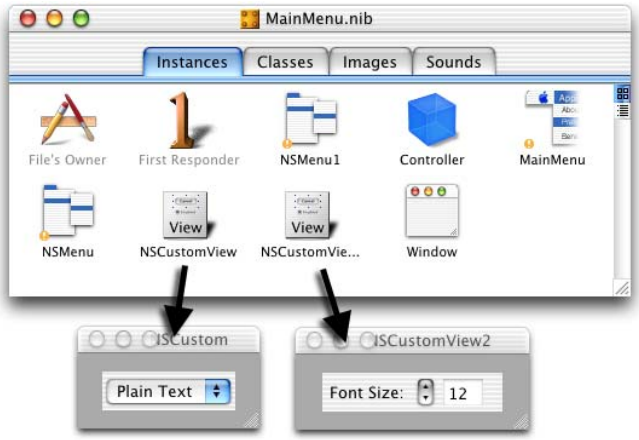
Print アイテムの画像はありませんが、これは、フレームワーク側に標準で用意されているツールバーアイテムだからです。このような標準で用意されているものを「 **標準(ツールバー)アイテム** 」と呼びます。アプリケーション側で用意するものを「 **カスタム(ツールバー)アイテム** 」と呼びます。標準アイテムには以下のものがあります。

【図】標準(ツールバー)アイテム

標準アイテム			
Print	:	印刷	(NSToolbarPrintItemIdentifier)
Colors	:	カラーパネルを表示	(NSToolbarShowColorsItemIdentifier)
Fonts	:	フォントパネルを表示	(NSToolbarShowFontsItemIdentifier)
Customize	:	設定パレットを表示	(NSToolbarCustomizeToolbarItemIdentifier)
Space	:	空白	(NSToolbarSpaceItemIdentifier)
Flexible Space	:	伸縮する空白	(NSToolbarFlexibleSpaceItemIdentifier)
Separator	:	区切りのマーク	(NSToolbarSeparatorItemIdentifier)

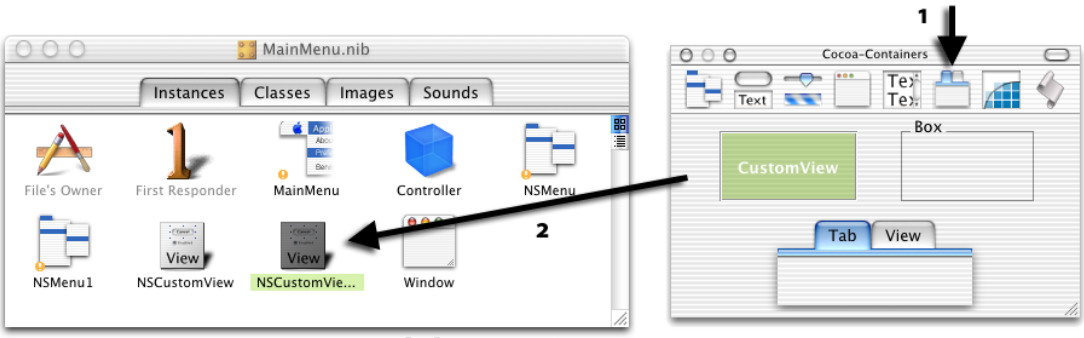
標準アイテムには、画像やアクションなどが予め準備されていますので、画像などの部品を準備しなくても使えるわけです。ただし、画像だけ差し換えたいというような場合に、差し換えることは可能です。括弧の中に書いている `NSToolbar...ItemIdentifier` というのは、コード中で標準アイテムを指定するための名前 (識別子) です。詳しくは後程説明します。

ビューアイテムの場合に、プロジェクトに準備しておくものはカスタムビューです。nib ファイルの中にカスタムビューを作って、その上に必要な部品 (ボタンやメニューなど) を並べたものを入れておきます。普通に Cocoa アプリケーションのプロジェクトを作成した場合は、MainMenu.nib の中にカスタムビューを作ればよいです。このサンプルの MainMenu.nib を開くと、以下のようになっています。カスタムビューが 2 つあるのが分かります。



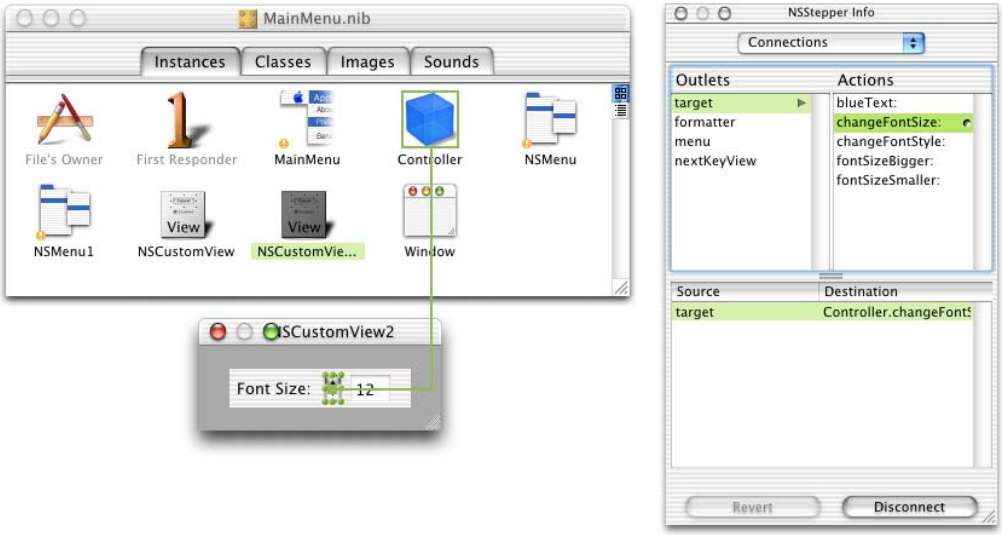
【図】nib ファイルの中のカスタムビュー

カスタムビューを作るには、パレットから CustomView と書かれたビューを MainMenu.nib の Instances タブへドロップします。そうしたら、この中に必要な部品を並べていけばよいです。



【図】カスタムビューの作成

ツールバーアイテムは、クリックしたりメニューを選んだりすると、当然ながら対応する機能が実行されます。ビューアイテムの場合は、一般のアプリケーションのウィンドウにビューを配置した時と同じようにアクションのコネクトを行うことができます。アウトレットについても、一般のアプリケーションの開発と全く同じように扱えます。

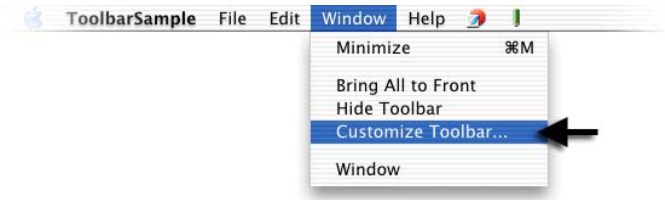


【図】ビューアイテムのアクションのコネクト

一方、イメージアイテムの場合は、画像にはアクションがありませんので、アクションの指定を Interface Builder 上では行えません。コード上でアクションを指定することになります。

◎ ツールバーのカスタマイズ

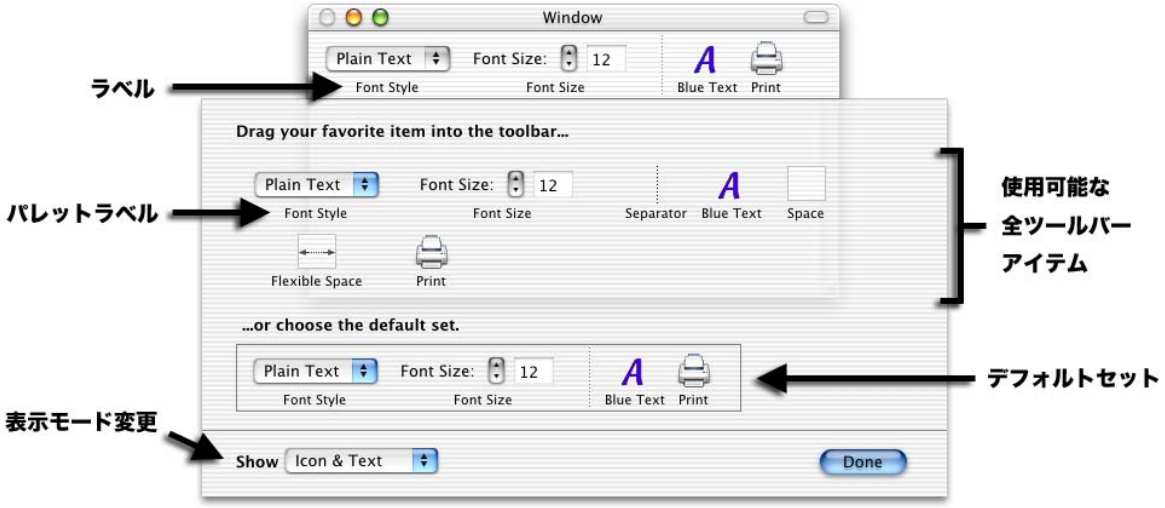
ツールバー上のアイテムは、ユーザーが自由に順番を入れ替えたり、用意されているアイテムの中から取捨選択ができるようになっています。「 Window → Customize Toolbar... 」メニューを選択してみてください。



【図】Customize Toolbar メニュー

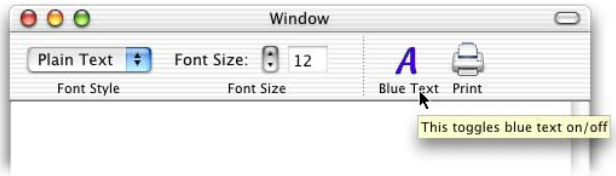
ちょっと補足：設定パレットを表示する方法は、ツールバーから表示されるコンテキストメニューから実行する方法や、ウィンドウの右上に表示されているボタンを、オプションキーとコマンドキーを同時に押したままクリックする方法もあります。

すると、以下のようなシートが表示されます。これを「 **設定パレット（ Customize Palette ）**」と呼びます。



【図】設定パレット

ツールバー上のテキストを「 **ラベル** 」といい、設定パレット上のテキストを「 **パレットラベル** 」といいます。ラベルは常に表示されるため短めが好まれますし、パレットラベルは多少長くても分かりやすい名前が好まれますので、別の名称が指定できます。ただし、このサンプルでは同じになっています。さらに詳細な説明を付けたい場合は、ツールチップを設定することもできます。マウスを重ねたまましばらく動かさないと下のようなツールチップが表示されます。

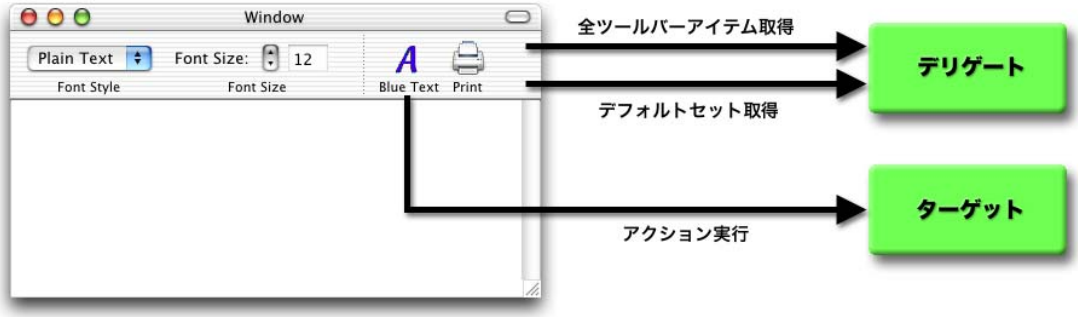


【図】ツールチップ

設定パレットには、アプリケーションで使用できるツールバーアイテムの全てが表示されていますし、かつ、よく使われると思われるツールバーのセットが**デフォルトセット**として表示されています。設定パレットを表示するのは、ツールバー（ `NSToolbar` クラス ）の役目ですので、この「 全ツールバーアイテムのリスト 」と「 デフォルトセットのリスト 」を教えてあげる必要があります。そのために、この 2 つのリストの情報を返すメソッドを作っておいて、これらのメソッドを持っているインスタンスをツールバーのデリゲート（ delegate ）として登録します。ツールバーは、これらのリスト情報が必要になったら、デリゲートのメソッドを呼び出します。

ちょっと補足：デリゲートというのは、**アプリケーション固有の処理のようにフレームワーク側ではできないことを、外部に用意してあるインスタンスに任せるための仕組み**です。また、その外部のインスタンスもデリゲートと呼びます。デリゲートにどういったメソッドを実装していないといけないかは、デリゲートを呼び出す側のクラスの仕様によりますので、リファレンスを参照して必要なメソッドを知っておく必要があります。ちなみに、デリゲートとは「 **委任** 」という意味です。Cocoa のフレームワークでは、このデリゲートの仕組みはよく使われます。

ここで、これまでの内容を図にしてみましょう。ツールバーを作成した時には、ツールバーの下請け的に働いて色んな情報を返したりするデリゲートと各ツールバーアイテムの処理を実行するためのターゲットのインスタンスが必要になります。ツールバーを使うためには、このデリゲートとターゲットを作るのが主な作業となります。今回のサンプルでは、`Controller` というクラスにデリゲートやターゲットなどの全ての処理が書かれています。



【図】ツールバーが動くために必要なインスタンスと送られるメッセージ

◎ アイテムリスト取得のための 2 つのデリゲートメソッド

ツールバーにはデリゲートがあって、デリゲートには、ツールバーアイテムの全リストとデフォルトセットの 2 つのアイテムリストを返すメソッドが必要だということが分かりました。ここでは、この 2 つのメソッドについて掘り下げていきます。

全てのツールバーアイテムには、重複しないような名前（ 文字列 ）が付けられていまして、この名前でツールバーを識別したり、ツールバーを指定したりします。この名前のことを「 **識別子（ Identifier ）** 」と呼びます。この 2 つのメソッドは、識別子の配列を返すメソッドになっています。具体的にどうなっているかを、`Controller.m` を開いて見てみましょう。まずは、全ツールバーアイテムを返す **`toolbarAllowedItemIdentifiers :`** からです。

Controller.m > toolbarAllowedItemIdentifiers

```
- (NSArray *) toolbarAllowedItemIdentifiers : (NSToolbar*) toolbar {

    return [ NSArray arrayWithObjects :
                 @"FontStyle",
                 @"FontSize",
                 NSToolbarSeparatorItemIdentifier,
                 @"BlueLetter",
                 NSToolbarSpaceItemIdentifier,
                 NSToolbarFlexibleSpaceItemIdentifier,
                 NSToolbarPrintItemIdentifier,
                 nil ];

}
```

このように、識別子の配列を 1 つ作ってそれを返しているだけです。NSToolbar...ItemIdentifier となっているのは、先程出てきた標準アイテムの識別子です。このように使うわけです。それ以外は、カスタムアイテムの識別子です。この**配列の順番と設定パレット上の順番が同じになっている**ところにも注目です。続いて、デフォルトセットを返す **toolbarDefaultItemIdentifiers :** も見てみます。

Controller.m > toolbarDefaultItemIdentifiers

```
- (NSArray *) toolbarDefaultItemIdentifiers : (NSToolbar *) toolbar {

    return [ NSArray arrayWithObjects :
                 @"FontStyle",
                 @"FontSize",
                 NSToolbarSeparatorItemIdentifier,
                 @"BlueLetter",
                 NSToolbarPrintItemIdentifier,
                 nil ];

}
```

これも同様に識別子の配列を返しているだけです。こちらも、配列の順番と設定パレット上の順番が同じになっています。メソッドの詳細は以下のとおりです。

NSToolbarDelegate : 全ツールバーアイテムの取得

書式
```
- (NSArray *) toolbarAllowedItemIdentifiers : (NSToolbar*) toolbar
```
入力
toolbar : 親のツールバーのインスタンス
出力
返り値 : 全ツールバーアイテムの識別子(文字列)の配列。配列の順番に設定パレットに表示される。
備考
このメソッドは、NSToolbarのデリゲートに必須のメソッド。
このメソッドは、主に以下のケースで呼ばれる(フレームワークのバージョンで変わる可能性がある)。
・設定パレットを表示する直前

NSToolbarDelegate : ツールバーアイテムのデフォルトセットの取得

書式
```
- (NSArray *) toolbarDefaultItemIdentifiers : (NSToolbar*) toolbar
```
入力
toolbar : 親のツールバーのインスタンス
出力
返り値 : デフォルトセットの識別子(文字列)の配列。配列の順番に設定パレットに表示される。
備考
このメソッドは、NSToolbarのデリゲートに必須のメソッド。
このメソッドは、主に以下のケースで呼ばれる(フレームワークのバージョンで変わる可能性がある)。
・設定パレットを表示する直前
・ツールバーがデフォルトセットに戻された時
・環境設定ファイルが無い状態で空のツールバーがウィンドウにセットされた時

ちょっと補足：どちらのメソッドもパラメータに親のツールバーのインスタンスが渡ってきます。今回のサンプルでは無視していますが、例えば、Mac OS X にバンドルされている Mail アプリケーションのように、受信メールのウィンドウとメール作成のウィンドウで別のツールバーを表示する場合もあります。そのため、見分ける必要がある場合にはこのパラメータを使います。もちろん、ツールバー毎にデリゲートを別にする方法もあります。

◎ アイテムリスト取得メソッドのちょっとした改造

説明ばかりでは面白くありませんので、ここで、ちょっとソースを改造してみましょう。2 つのメソッドに以下の改造を加えます。

Controller.m > toolbarAllowedItemIdentifiers

```
- (NSArray *) toolbarAllowedItemIdentifiers : (NSToolbar*) toolbar {

    NSLog( @"toolbarAllowedItemIdentifiers" ); // 追加

    return [ NSArray arrayWithObjects :
                    : 以下の3行を追加
                NSToolbarShowColorsItemIdentifier,
                NSToolbarShowFontsItemIdentifier,
                NSToolbarCustomizeToolbarItemIdentifier,
                nil ];

}
```

Controller.m > toolbarDefaultItemIdentifiers

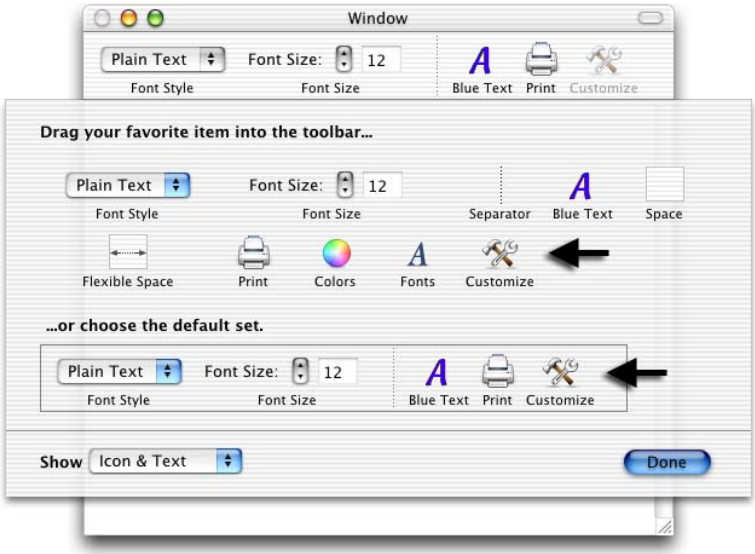
```
- (NSArray *) toolbarDefaultItemIdentifiers : (NSToolbar *) toolbar {

    NSLog( @"toolbarDefaultItemIdentifiers" ); // 追加

    return [ NSArray arrayWithObjects :
                    : 以下の1行を追加
                NSToolbarCustomizeToolbarItemIdentifier,
                nil ];

}
```

改造の結果は以下のとおりです。矢印のところにアイテムが増えているのが確認できます。NSLog を埋めたのは、どういうタイミングでデリゲートメソッドが呼ばれるかを知るためのものです。NSLog の出力を見ながら色んな操作をしてみると、ツールバーの動作の理解が進むと思いますので、試してみてください。



【図】改造の結果

◎ ツールバーが出来上がるまで

続いては、画面にツールバーが表示されるまでの流れを説明します。主に、ツールバー内部で自動的に行われる処理です。こちらでコーディングが必要なのは、デリゲートの部分だけです。

ツールバーをカスタマイズしたら、一旦アプリケーションを終了して、再度起動してみてください。すると、ツールバーは、カスタマイズの状態が記憶されていることが分かります。**ツールバーの状態は環境設定ファイルに自動的に保存されて**、同じツールバーを使おうとしたときには、**カスタマイズの状態を自動的に再現してくれます**（自動処理をオフにすることも可能です）。

初めてアプリケーションを起動した時には、環境設定ファイルはありませんので、デフォルトセットをツールバーに表示します。つまり、先程のデリゲートのメソッド（ toolbarDefaultItemIdentifiers : ）を呼んで、デフォルトセットのリストを取得します。環境設定ファイルがあった場合は、読み込んで、その中からツールバーアイテムのリストを取得します。

リストが取得できたら、次はツールバーアイテムをツールバー上に表示することになりますが、表示するためには、イメージビューならば画像とラベル、ビューアイテムならば、カスタムビューとラベルが最低限必要です。この情報を得るために、デリゲートにはもう 1 つメソッドが必要になります。**識別子を与えるとツールバーアイテムそのもの（ インスタンス ）を返すメソッド**です。ツールバーアイテムのインスタンスそのものが取得できれば、画像取得メソッド（ image ）やラベル取得メソッド（ label ）などを使って、情報を得ることが出来ます。

このインスタンス取得メソッドの詳細は以下になっています。

NSToolbarDelegate : ツールバーアイテムの識別子からインスタンスを取得

書式

```
- (NSToolbarItem *)   toolbar : (NSToolbar *) toolbar
          itemForItemIdentifier : (NSString  *) itemIdentifier
      willBeInsertedIntoToolbar : (BOOL       ) flag
```

入力

```
toolbar        : 親のツールバーのインスタンス
itemIdentifier : ツールバーアイテムの識別子
flag           : このツールバーアイテムをこれからツールバーに挿入するかどうか
```

出力

```
返り値         : itemIdentifierに対応するツールバーアイテムのインスタンス
```

備考

このメソッドは、NSToolbarのデリゲートに必須のメソッド。
このメソッドは、主に以下のケースで呼ばれる(フレームワークのバージョンで変わる可能性がある)。
・ツールバーがウィンドウに付けられた時 (flag = YES)
・設定パネルを開いた時(全アイテムとデフォルトセットの両方を表示する時) (flag = NO)
・ユーザーが設定パネルからツールバーにドロップした時 (flag = YES)

このメソッドでは、ツールバーアイテム識別子がパラメータとして与えられ、対応するインスタンスを返すわけですが、この処理のために、このサンプルでは、アプリケーションの起動時に識別子からインスタンスを取り出せる辞書（ NSMutableDictionary ）を作成しています。詳細は後程説明しますが、Controller.m の awakeFromNib で辞書を作成しています。

Controller.m > awakeFromNib

```
- (void) awakeFromNib {
      : 省略
    // アイテム辞書作成
    toolbarItems = [ [ NSMutableDictionary dictionary ] retain ];

    // ツールバーアイテムを辞書(toolbarItems)に登録
    addToolbarItem( … 省略 … ); // Font Style
    addToolbarItem( … 省略 … ); // Font Size
    addToolbarItem( … 省略 … ); // Blue Text
      : 省略
```

辞書に登録しているのはカスタムアイテムの 3 つだけで、標準アイテムは辞書に登録していませんが、フレームワーク自身がインスタンスを持っているので、問い合わせは発生しないためです。そして、Controller.m のデリゲートメソッドは、以下のようになっています。ここでは概略だけです。

Controller.m > toolbar : itemForItemIdentifier : willBeInsertedIntoToolbar :

```
- (NSToolbarItem *)    toolbar : (NSToolbar *) toolbar
         itemForItemIdentifier : (NSString  *) itemIdentifier
     willBeInsertedIntoToolbar : (BOOL       ) flag {

    NSToolbarItem *newItem = [ [ [ NSToolbarItem alloc ]
                                   initWithItemIdentifier : itemIdentifier ]
                                     autorelease ]; // アイテムを作成
    // アイテム辞書から取り出し
    NSToolbarItem *item    = [ toolbarItems objectForKey : itemIdentifier ];
      : 省略 ( アイテムのコピー処理 )
    return newItem;
}
```

新規にツールバーアイテムを作って、辞書の中のツールバーアイテムのコピーを作ってそれを返すようになっています。alloc と initWithItemIdentifier : で識別子指定で空のツールバーアイテムが作れます。そして、辞書から objectForKey : でインスタンスを取り出しています。

NSToolbarItem : 識別子を指定して初期化

書式

```
  - (id) initWithItemIdentifier : (NSString *) itemIdentifier
```

入力

itemIdentifier : ツールバーアイテムの識別子

出力

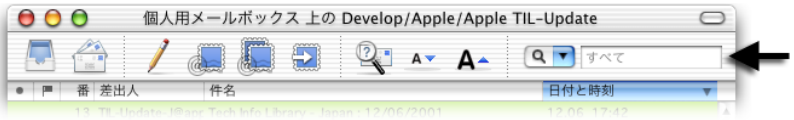
返り値　　: 初期化したNSToolbarItem

このメソッドにも以下の改造を加えてみましょう。動かしてみると、どういうタイミングでこのメソッドが呼ばれるかがよく分かると思います。

```
Controller.m > toolbar : itemForItemIdentifier : willBeInsertedIntoToolbar :

    // メソッド中のどこかに追加
    NSLog( @"                  toolbar : %@", [ toolbar identifier ] );
    NSLog( @"    itemForItemIdentifier : %@", itemIdentifier );
    NSLog( @"willBeInsertedIntoToolbar : %@", flag ? @"YES" : @"NO" );
```

3 つ目のパラメータの **flag** は、問い合わせがあったツールバーアイテムがこの後にツールバー上に登録されるかどうかを表しています。ツールバーアイテムによっては、**ツールバーに登録する時に初期化が必要なものがある**と思います。例えば、Mac OS X の Mail アプリケーションには検索用のツールバーアイテムがありますが、このようなアイテムでは検索文字列の初期化が必要なことがありますが、このメソッド内で初期化を行うことができます。初期化すべきかどうかを、この flag で判断することができるというわけです。



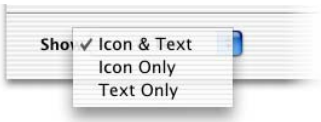
【図】Mail のウィンドウの検索用のツールバーアイテム

さて、ここまででデリゲートに実装する 3 つのメソッドについて見てきましたが、実は、**この 3 つがツールバーのデリゲートの必須メソッド**です。ツールバーを使うためには、この 3 のメソッドを必ずデリゲートに実装しておかなくてはいけません。逆にいえば、この 3 つのデリゲート以外の処理は、フレームワーク側で行ってくれるということなのです。

■ 表示モードによる挙動の変化

◎ 3 つの表示モード

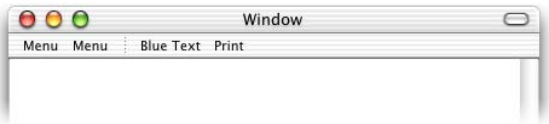
ツールバーには 3 つの表示モードがあります。「 Icon & Text 」と「 Icon Only 」と「 Text Only 」です。変更は、設定パレットの下部のポップアップボタンで行います。



【図】表示モードの変更

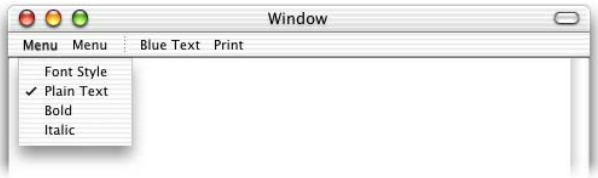
> **ちょっと補足**：表示モードを変えるには別の方法もあります。ツールバーのコンテキストメニューから選ぶ方法と、ウィンドウの右上のボタンをコマンドキーを押したままクリックする方法です。

注目すべきなのは、Text Only モードです。このモードでは、**ツールバーアイテムがテキストで表示される**ため、各ツールバーアイテムは、複雑な形状は持てません。



【図】Text Only モードでの表示

ただし、単なるボタンになってしまってはビューアイテムの機能を実現することは困難ですので、各**ツールバーアイテムにはメニューを付けることができる**ようになっています。Font Style アイテムからは以下のようなメニューが表示されます。

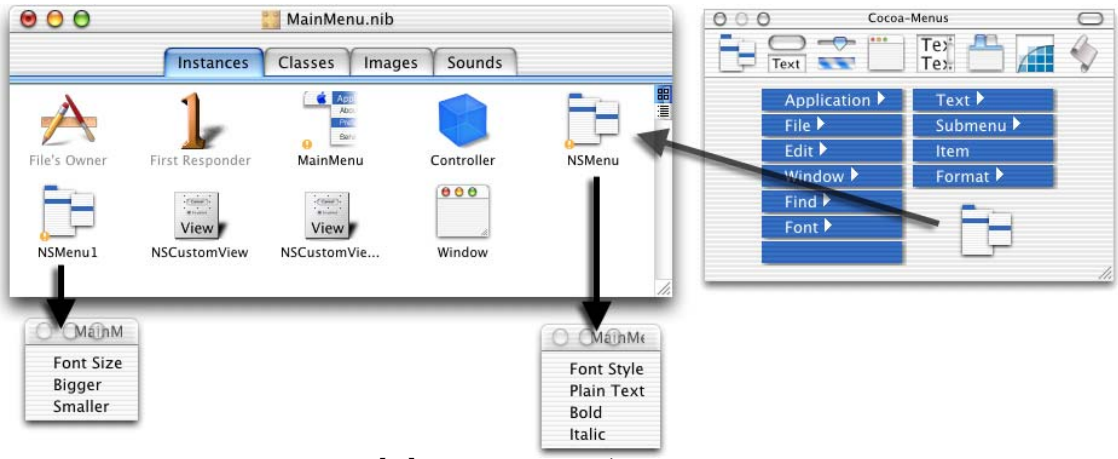


【図】Text Only モードでの Font Style アイテム

Mail アプリケーションの検索機能のようなものは、メニューでも代用できませんので、使えなくなるという仕様になっています。**ビューアイテムの場合は、メニューを付けなければ自動的に使えなくなります**。ただし、メニューから検索文字列を入れるダイアログが表示させるという方法も考えられます。

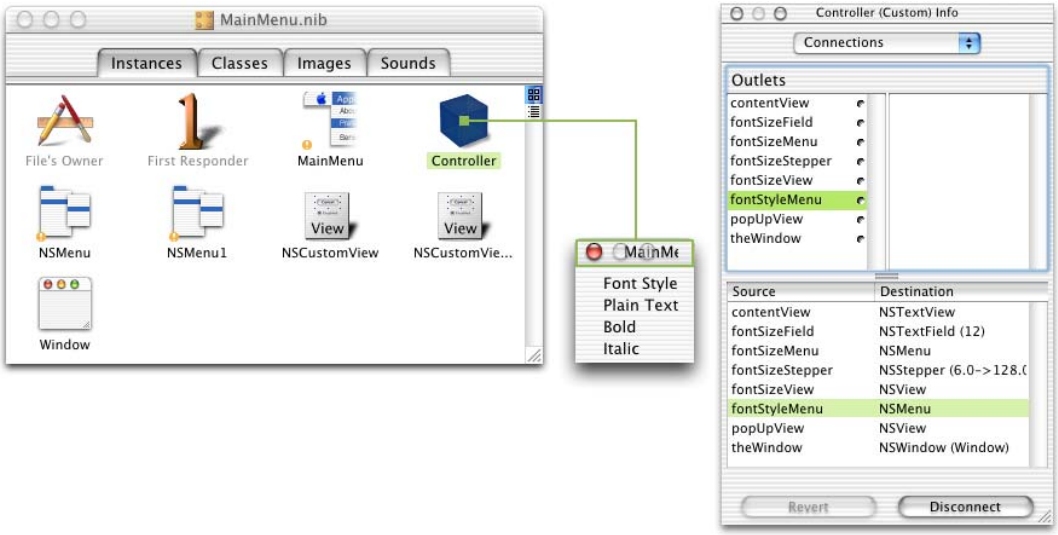
【図】Text Only モードでの Mail のツールバー

メニューの作成は、一般の Cocoa アプリケーションと同じで Interface Builder で行います。このサンプルでは、以下のように 2 つのメニューがあることがわかります。



【図】MainMenu.nib の中のメニュー

このサンプルでは、`Controller` というインスタンスからメニューを利用していますので、参照のためのアウトレットを用意してコネクトしています。Font Style メニューは以下のように `fontStyleMenu` というアウトレットにコネクトしています。

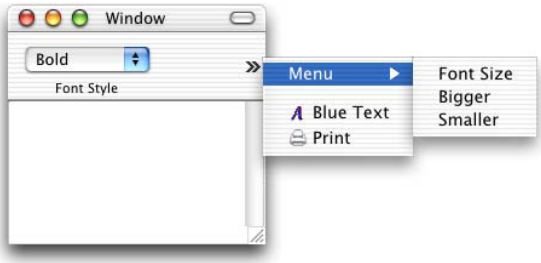


【図】ツールバーアイテムのメニューとのコネクト

メニューのアクションのコネクトもいつものとおり行えますので、メニューから実行されるアクションもコネクトしておきます。

◎ オーバーフローメニュー

Icon & Text モードや、Icon Only モードでも、ウィンドウの横幅が狭くなると、ツールバーに入りきらないアイテムは強制的にメニューにされてしまいます。このメニューを「 **オーバーフローメニュー (Overflow Menu)** 」と呼びます。Text Only モードのときと同じメニューが使用されます。



【図】オーバーフローメニュー

◎ ツールバーアイテムのおさらい

これで、ツールバーとツールバーアイテムの大体の機能を見ることが出来ましたので、ツールバーアイテムのおさらいをしてみます。ツールバーアイテムは、以下のようになっています。イメージアイテムとビューアイテムに分けて書いておきます。

イメージアイテムの特徴
・画像とラベルで表示される。 ・設定パネル用にパレットラベルを持つ。ツールチップを持つ。 ・プロジェクトには画像ファイルを用意する。 ・アクションを1つ持つ。アイテムとのコネクトはコード上から行う。 ・テキスト表示の時のメニューを持つことができる(通常は使わない)。

ビューアイテムの特徴

・カスタムビューとラベルで表示される。
・設定パネル用にパレットラベルを持つ。ツールチップを持つ。
・プロジェクトには、カスタムビューの入ったnibファイルを用意する。
・アクションは、カスタムビューの中の部品をコネクトしておく(コード上での指定も可能)。
・テキスト表示の時のメニューを持つことが出来る(nibファイル内に用意するのが簡単)。

■ ツールバーの初期化

◎ 起動時の処理の概要

ツールバーを初期化するコードは、awakeFromNib に書かれています。アプリケーションが起動した時に MainMenu.nib が読み込みまれますが、その時に、その中に含まれるインスタンスの awakeFromNib が実行されます。

Controller.m > awakeFromNib

```
- (void) awakeFromNib {
      : 省略
    NSToolbar *toolbar = [ [ [ NSToolbar alloc ]
                                initWithIdentifier : @"myToolbar" ]
                                  autorelease ];  // ツールバーを作成
    // アイテム辞書作成
    toolbarItems = [ [ NSMutableDictionary dictionary ] retain ];

    // ツールバーアイテムを辞書(toolbarItems)に登録
    addToolbarItem( … 省略 … ); // Font Style
    addToolbarItem( … 省略 … ); // Font Size
    addToolbarItem( … 省略 … ); // Blue Text

    // ツールバーの属性をセット
    [ toolbar setDelegate                : self ]; // デリゲートをセット
    [ toolbar setAllowsUserCustomization : YES  ]; // 設定パレットを使う
    [ toolbar setAutosavesConfiguration  : YES  ]; // 設定の自動保存をする
    [ toolbar setDisplayMode : NSToolbarDisplayModeIconOnly ]; // 表示モード設定

    [ theWindow setToolbar   : toolbar ]; // ツールバーをウィンドウにセット
      : 省略
}
```

順に細かく見ていきましょう。まず、最初にツールバーのインスタンスを作っています。これを後程ウィンドウにセットします。alloc で生成して **initWithIdentifier :** で初期化しています。

```
    NSToolbar *toolbar = [ [ [ NSToolbar alloc ]
                               initWithIdentifier : @"myToolbar" ]
                                 autorelease ];  // ツールバーを作成
```

initWithIdentifier : は、名前のとおり**識別子で初期化**をします。ツールバーアイテムにも識別子がありましたが、**ツールバーにも識別子があります**。Mail のように、受信メールのウィンドウとメール作成ウィンドウで異なるツールバーを使用する場合もありますが、識別子を使うことで、どちらのツールバーかが識別できます。

NSToolbar : 識別子を指定して初期化

書式
- (id) initWithIdentifier : (NSString *) identifier
入力
identifier : ツールバーの識別子
出力
返り値 : 初期化したNSToolbar

次は、全てのツールバーアイテムを入れておくための辞書 toolbarItems を作成します。その後ろで addToolbarItem をいう関数を 3 回呼んでいますが、この addToolbarItem の中で NSToolbarItem を作成して toolbarItems に登録を行っています。ここは、addToolbarItem と合わせて後程説明します。

```
    // アイテム辞書作成
    toolbarItems = [ [ NSMutableDictionary dictionary ] retain ];

    // ツールバーアイテムを辞書(toolbarItems)に登録
    addToolbarItem( … 省略 … ); // Font Style
    addToolbarItem( … 省略 … ); // Font Size
    addToolbarItem( … 省略 … ); // Blue Text
```

ツールバーアイテムの辞書が出来たところで、**setDelegate :** でデリゲートを自分自身にセットします。**setAllowsUserCustomization :** でユーザーに設定パレットを使わせるかどうかを指定します。常に同じツールバーを表示してカスタマイズさせないのであれば NO でも構いませんが、通常は YES でいいでしょう。続いて、**setAutosavesConfiguration :** でツールバーのカスタマイズ状態の自動保存をするかどうかを指定します。こちらも特殊な処理をしない限りは YES でよいでしょう。

```
    // ツールバーの属性をセット
    [ toolbar setDelegate                : self ]; // デリゲートをセット
    [ toolbar setAllowsUserCustomization : YES  ]; // 設定パレットを使う
    [ toolbar setAutosavesConfiguration  : YES  ]; // 設定の自動保存をする
    [ toolbar setDisplayMode : NSToolbarDisplayModeIconOnly ]; // 表示モード設定

    [ theWindow setToolbar   : toolbar ]; // ツールバーをウィンドウにセット
```

そして、**setDisplayMode :** でデフォルトの表示モードを指定します。表示モードの指定は以下のようにな

っています。

NSToolbar : 表示モード

```
typedef enum {
    NSToolbarDisplayModeDefault,
    NSToolbarDisplayModeIconAndLabel, // Icon & Text
    NSToolbarDisplayModeIconOnly,     // Icon Only
    NSToolbarDisplayModeLabelOnly     // Text Only
} NSToolbarDisplayMode;
```

最後に、**setToolbar :** でウィンドウにツールバーをセットします。setToolbar : をすると、セットしようとしたツールバーの識別子を見て、アプリケーションの環境設定ファイルの中にそのツールバーのカスタマイズ状態が記録されているかをチェックします。記録されていればその状態を再現しますし、無ければデフォルトセットをツールバーに表示します。環境設定ファイルが無い場合もデフォルトセットを表示します。ただし、この自動処理は、setAutosavesConfiguration : で YES を指定している時に限られます。

ちなみに、環境設定ファイルには以下のような内容が書き込まれます。

/Users/<USERNAME>/Library/ToolbarSample.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist SYSTEM "file://localhost/System/Library/DTDs/PropertyList.dtd">
<plist version="0.9">
<dict>
    <key>NSToolbar Configuration myToolbar</key>
    <dict>
        <key>TB Display Mode</key>
        <integer>1</integer>
        <key>TB Is Shown</key>
        <integer>1</integer>
        <key>TB Item Identifiers</key>
        <array>
            <string>FontStyle</string>
            <string>FontSize</string>
            <string>NSToolbarSeparatorItem</string>
            <string>BlueLetter</string>
            <string>NSToolbarPrintItem</string>
        </array>
        <key>TB Size Mode</key>
        <integer>1</integer>
    </dict>
    <key>NSWindow Frame NSFontPanel</key>
    <string>603 269 300 221 0 0 1152 746 </string>
</dict>
</plist>
```

◎ ツールバーアイテム辞書の登録処理

さて、先程説明をとばした addToolbarItem のところに戻ります。パラメータが沢山ありますので、まずはパラメータの説明から行います。addToolbarItem 関数の先頭部分を見ていただきましょう。

Controller.m > addToolbarItem

```
static void addToolbarItem (
    NSMutableDictionary *theDict, // 登録先のツールバーアイテム辞書
    NSString *identifier,         // 識別子
    NSString *label,              // ラベル
    NSString *paletteLabel,       // パレットラベル
    NSString *toolTip,            // ツールチップ
    id        target,             // イメージアイテム用のターゲット
    SEL       settingSelector,    // ツールバーにセットする時のメソッド名
    id        itemContent,        // ツールバーにセットする画像かビュー
    SEL       action,             // イメージアイテム用のアクション
    NSMenu   *menu                // メニュー
) {
```

最初の theDict が、ツールバーアイテムを登録する辞書のインスタンスで、それ以降は、登録するツールバーアイテムに関する情報になっています。

identifier から toolTip までは、今まで説明したツールバーアイテムの属性そのものですので、対応する文字列を渡します。target は、イメージアイテムがクリックされた時のメッセージの送り先です。送るメッセージは action で指定します。ビューアイテムの場合は、ビューアイテムの中の部品に予め設定されているため、このターゲットとアクションは無視されます。itemContent は、イメージアイテムなら画像のインスタンス、ビューアイテムならカスタムビューのインスタンスを渡します。これをツールバーアイテムへセットするわけですが、セットする際に使用するメソッドを settingSelector で指定します。イメージアイテムの場合は、**setImage :** を、ビューアイテムの場合は、**setView :** を指定します。menu は、Text Only モードの時やオーバーフローメニューで使用するメニューです。

呼び出し元のところを見てみます。addToolbarItem は、3 回呼ばれていますが、それぞれはツールバーのアイテムの「 Font Style 」「 Font Size 」「 Blue Text 」の 3 つの登録の処理です。Font Style と Font Size は、ビューアイテムでほぼ内容が同じなので、Font Style のみ説明します。Blue Text は、イメージアイテムのなのでビューアイテムとはパラメータがちょっと異なります。

Controller.m > awakeFromNib

```
    // Font StyleとFont Sizeを辞書に登録 ( ビューアイテム )
    addToolbarItem( toolbarItems,
                    @"FontStyle",  @"Font Style", @"Font Style",
                    @"Change your font style",
                    self, @selector(setView:), popUpView, NULL,
                    fontStyleMenu );

    addToolbarItem( … 省略 … );

    // Blue Textを辞書に追加 ( イメージアイテム )
    addToolbarItem( toolbarItems,
                    @"BlueLetter", @"Blue Text", @"Blue Text",
                    @"This toggles blue text on/off",
                    self, @selector(setImage:),
                    [ NSImage imageNamed : @"blueLetter.tif" ],
                    @selector(blueText:), fontSizeMenu );
```

では、addToolbarItem の中を詳しく見ていきます。

Controller.m > addToolbarItem

```
static void addToolbarItem (
   : 省略
) {

    NSMenuItem *mItem;
    NSToolbarItem *item = [ [ [ NSToolbarItem alloc ]
                                initWithItemIdentifier : identifier ]
                                  autorelease ]; // ツールバーアイテム作成

    [ item        setLabel : label        ]; // ラベルを設定
    [ item setPaletteLabel : paletteLabel ]; // パレットラベルを設定
    [ item      setToolTip : toolTip      ]; // ツールチップを設定
    [ item       setTarget : target       ]; // ターゲットを設定

    [ item performSelector : settingSelector
                withObject : itemContent ];  // アイテムに画像かビューをセット
    [ item       setAction : action ];       // アクションを設定

    if ( menu != NULL ) { // メニューを設定
        mItem = [ [ [ NSMenuItem alloc ] init ] autorelease ];
        [ mItem setSubmenu : menu           ];
        [ mItem   setTitle : [ menu title ] ];
        [ item setMenuFormRepresentation : mItem ];
    }

    [ theDict setObject : item
                 forKey : identifier ]; // アイテム辞書に登録
}
```

最初に空のツールバーアイテムを alloc と **initWithItemIdentifier :** で作りまして、後は、**setXXX :** メソッドで次々に属性をセットしていっています。performSelector : withObject : は、メッセージを指定のインスタンスに送るためのメソッドです。

```
    [ item performSelector : settingSelector
                withObject : itemContent ];  // アイテムに画像かビューをセット
```

item インスタンスが持っている settingSelector メソッドを itemContent をパラメータとして実行するというものです。つまり、イメージアイテムの場合は、

```
  [ item setImage : itemContent ];
```

が実行されますし、ビューアイテムの場合は、

```
[ item setView : itemContent ];
```

が実行されることになります。**ツールバーアイテムは、setImage : を呼ぶことでイメージアイテムになり、setView : を呼ぶことでビューアイテムになります。**

そして、**setMenuFormRepresentation :** でメニューをセットしています。このメソッドのパラメータは、NSMenuItem ですが、Interface Builder では NSMenu を作ることになりますので、ちょっと細工をして NSMenuItem を作っています。最後に、辞書に登録しています。

では、ここで出てきたメソッドを一気に紹介します。

NSToolbarItem : ラベル名を取得する

書式
```
- (NSString *) label
```
出力
返り値 : ラベル名
備考
ラベル名とは、ツールバー上の各アイテムの下部に表示されるの名称のこと。

NSToolbarItem : ラベル名を変更する

書式
```
- (void) setLabel : (NSString *) label
```
入力
label : 変更するラベル名

NSToolbarItem : パレットラベル名を取得する

書式
```
- (NSString *) paletteLabel
```
出力
返り値 : パレットラベル名
備考
パレットラベル名とは、設定パネル上の各アイテムの下部に表示されるの名称のこと。

NSToolbarItem : パレットラベル名を変更する

書式
```
- (void) setPaletteLabel : (NSString *) paletteLabel
```
入力
paletteLabel : 変更するパレットラベル名

NSToolbarItem : ツールチップを取得する

書式
```
- (NSString *) toolTip
```
出力
返り値 : ツールチップ
備考
ツールチップとは、ツールバー上の各アイテムにマウスを重ねた時に表示される簡易ヘルプのこと。

NSToolbarItem : ツールチップを変更する

書式
```
- (void) setToolTip : (NSString *) toolTip
```
入力
toolTip : 変更するツールチップ

NSToolbarItem : ターゲットを取得する

書式
```
- (id) target
```
出力
返り値 : ターゲット

NSToolbarItem : ターゲットを変更する

書式
```
- (void) setTarget : (id) target
```
入力
target : 変更するターゲット
備考
イメージアイテムにのみ有効。

NSToolbarItem : アクションを取得する

書式
```
- (SEL) action
```
出力
返り値 : アクション

NSToolbarItem : アクションを変更する

書式
```
- (void) setAction : (SEL) action
```
入力
action : 変更するアクション
備考
イメージアイテムにのみ有効。

NSToolbarItem : メニューを取得する

書式
```
- (NSMenuItem *) menuFormRepresentation
```
出力
返り値 : メニュー

NSToolbarItem : メニューを変更する

書式
```
- (void) seMenuFormRepresentation : (NSMenuItem *) menuItem
```
入力
menuItem : 変更するメニュー

NSToolbarItem : 画像を取得する

書式
```
- (NSImage *) image
```
出力
返り値 : 画像
備考
イメージアイテムにのみ有効。

NSToolbarItem : 画像を変更する

書式
```
- (void) setImage : (NSImage *) image
```
入力
image : 変更する画像
備考
このメソッドを実行することによってイメージアイテムになる。

NSToolbarItem :ビューを取得する

書式
```
- (NSView *) view
```
出力
返り値 : ビュー
備考
ビューアイテムにのみ有効。

NSToolbarItem : ビューを変更する

書式
```
- (void) setView : (NSView *) view
```
入力
image : 変更するビュー
備考
このメソッドを実行することによってビューアイテムになる。

■ インスタンス取得メソッドの詳細

これで、ツールバーアイテムの辞書が出来ましたので、この辞書を主に使うインスタンス取得メソッドの toolbar : itemForItemIdentifier : willBeInsertedIntoToolbar : を詳しく見ていきましょう。

Controller.m > toolbar : itemForItemIdentifier : willBeInsertedIntoToolbar :

```
- (NSToolbarItem *)    toolbar : (NSToolbar *) toolbar
         itemForItemIdentifier : (NSString  *) itemIdentifier
     willBeInsertedIntoToolbar : (BOOL       ) flag {

    NSToolbarItem *newItem = [ [ [ NSToolbarItem alloc ]
                                   initWithItemIdentifier : itemIdentifier ]
                                     autorelease ]; // アイテムを作成
    // アイテム辞書から取り出し
    NSToolbarItem *item    = [ toolbarItems objectForKey : itemIdentifier ];

    // 以降でコピーする
    [ newItem        setLabel : [ item label        ] ]; // ラベルをコピー
    [ newItem setPaletteLabel : [ item paletteLabel ] ]; // パレットラベルをコピー

    // ビューと画像をコピー
    if ( [ item view ] != NULL ) { [ newItem setView  : [ item view  ] ]; }
    else                         { [ newItem setImage : [ item image ] ]; }

    [ newItem setToolTip : [ item toolTip ] ]; // ツールチップをコピー
    [ newItem  setTarget : [ item target  ] ]; // ターゲットをコピー
    [ newItem  setAction : [ item action  ] ]; // アクションをコピー
    // メニューをコピー
    [ newItem setMenuFormRepresentation : [ item menuFormRepresentation ] ];

    if ( [ newItem view ] != NULL ) { // ビューアイテムならサイズを設定
        [ newItem setMinSize : [ [ item view ] bounds ].size ];
        [ newItem setMaxSize : [ [ item view ] bounds ].size ];
    }

    return newItem;

}
```

先程説明しましたが、アイテム辞書に登録されているインスタンスをコピーしてメソッドの返り値にしています。わざわざコピーをしなければ以下のようになりますが、コピーするには理由があります。アプリケーションにツールバーを表示するウィンドウが 1 つしかない場合は、以下のように辞書の中にあるインスタンスを返す書き方でも構いません。

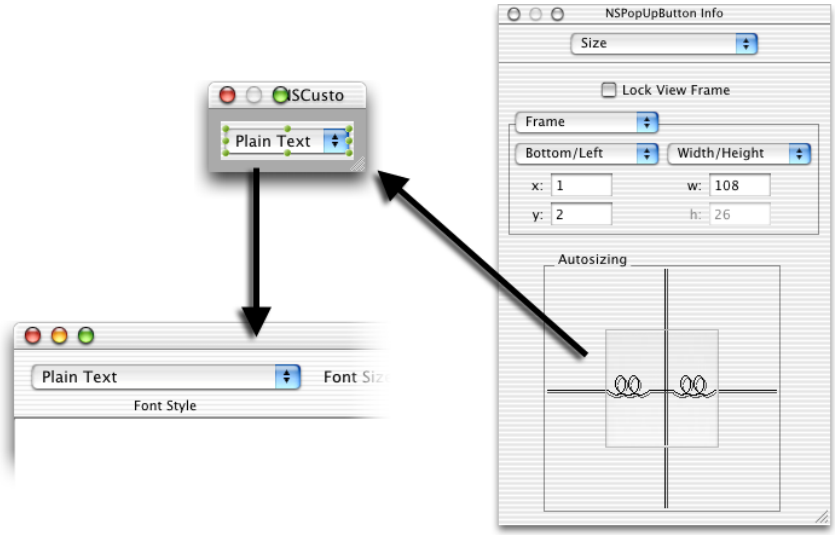
```
- (NSToolbarItem *)    toolbar : (NSToolbar *) toolbar
         itemForItemIdentifier : (NSString  *) itemIdentifier
     willBeInsertedIntoToolbar : (BOOL       ) flag {

    return [ toolbarItems objectForKey : itemIdentifier ];
}
```

しかしながら、複数のウィンドウにツールバーを表示する場合は、ツールバー毎にツールバーアイテムのインスタンスを作る（つまりコピーする）のが一般的です。というのは、ウィンドウ毎にツールバーアイテムの状態が異なる可能性があるためです。例えば、ドキュメントの保存のボタンをツールバーアイテムとして表示している場合、ドキュメントに変更のあった場合のみツールバーアイテムは使えるべきです。ドキュメントが変更されているかどうかというのは、ウィンドウ単位で異なるため、同じインスタンスを共有してはいけないということになります。

ものによっては、アプリケーションの状態によってツールバーアイテムの状態が変化するというのもあるでしょう。こういう場合は、コピーしなくてもよいということになります。

最後の方で **setMinSize :** と **setMaxSize :** というメソッドを呼んでいますが、これによって**アイテムサイズの最大値と最小値を指定できます**。最小値よりも最大値の方が横幅が広くなっていると、**アイテムがウィンドウの幅に応じて伸縮します**。これは、**ビューアイテムにのみ有効**です。また、伸縮するのは、カスタムビューの背景なので、中に配置している部品も連動させたい場合は、以下のように Autosizing の設定をしておく必要があります。



【図】伸縮するアイテムのオートサイズの設定と実行結果

NSToolbarItem : 最大サイズを取得
書式 - (NSSize) maxSize **出力** 返り値 : 最大サイズ

NSToolbarItem : 最大サイズを変更

書式
```
- (void) setMaxSize : (NSSize) size
```
入力
size : 変更する最大サイズ
備考
ビューアイテムにのみ有効。

NSToolbarItem : 最小サイズを取得

書式
```
- (NSSize) minSize
```
出力
返り値 : 最小サイズ

NSToolbarItem : 最小サイズサイズを変更

書式
```
- (void) setMinSize : (NSSize) size
```
入力
size : 変更する最小サイズ
備考
ビューアイテムにのみ有効。

■ 必須ではないデリゲートメソッド

ここでは、必須ではないデリゲートメソッドを紹介していきます。必須ではないので、実装しなくてもツールバーは動きますが、より細かい制御をしたい場合にはこれらのメソッドを実装するとよいでしょう。

◎ ツールバーアイテムの登録通知

Controller.m には toolbarWillAddItem : というメソッドもありますが、これは、ツールバーにアイテムが登録される直前に呼ばれます (toolbar : itemForItemIdentifier : willBeInsertedToolbar : よりは後)。主に、標準アイテムの属性を変更するのに使用します。ここでは、Print のツールチップとターゲットの変更を行っています。パラメータの NSNotification の userInfo の item に NSToolbarItem が入っていますので、取り出して、setToolTip : と setTarget : を実行しています。ちなみに、標準では、ツールチップは「 Print 」で、ターゲットはファーストレスポンダーになっています。

Controller.m > toolbarWillAddItem

```
- (void) toolbarWillAddItem : (NSNotification *) notif {

    NSToolbarItem *addedItem = [ [ notif userInfo ] objectForKey : @"item" ];

    if ( [ [ addedItem itemIdentifier ]
                isEqual : NSToolbarPrintItemIdentifier ] ) { // Printの場合
        [ addedItem setToolTip : @"Print your document" ];  // ツールチップ変更
        [ addedItem setTarget  : self                   ];  // ターゲット変更
    }

}
```

NSToolbarNotifications : ツールバーアイテム登録通知

書式
- (void) toolbarWillAddItem : (NSNotification *) notification
入力
notification : 通知情報

◎ ツールバーアイテムの削除通知

登録通知の逆で、ツールバー上から取り除かれた後にも通知がきます。例えば、ツールバーアイテムの初期化時にメモリを確保しているようなものの場合は、ここで破棄を行います。

Controller.m > toolbarDidRemoveItem

```
- (void) toolbarDidRemoveItem : (NSNotification *) notif {

    NSToolbarItem *removedItem = [ [ notif userInfo ] objectForKey : @"item" ];
      : removedItemに対する終了処理
}
```

NSToolbarNotifications : ツールバーアイテム削除通知

書式
- (void) toolbarDidRemoveItem : (NSNotification *) notification
入力
notification : 通知情報

◎ ツールバーのバリデーション

状況によって一時的にツールバーアイテムを使用不可にしたいことがありますが、これを制御するにはバリデーションの機構（有効かどうかを返す）を使います。ツールバーアイテムのターゲットのインスタンスに **validateToolbarItem :** メソッドを書いておきますと、バリデーションが必要な時にツールバーアイテム単位でこのメソッドが呼ばれます。

Controller.m > validateToolbarItem :

```
- (BOOL) validateToolbarItem : (NSToolbarItem *) theItem {

    // NSLog( @"validateToolbarItem : %@", [ theItem label ] ); // 追加

    return YES; // or NO

}
```

パラメータにツールバーアイテムのインスタンスが渡ってきますので、判断をして有効ならば YES を無効ならば NO を返すようにします。上のように NSLog を入れておきますと分かりますが、マウスが動いただけでこのメソッドは呼ばれることがあります。

NSToolbarItemValidation : ツールバーアイテムが有効かどうかを返す

書式
- (BOOL) validateToolbarItem : (NSToolbarItem *) theItem
入力
theItem: 対象のツールバーアイテム
出力
返り値　: 有効 = YES、無効 = NO

これは、イメージアイテムにのみ有効です。

◎ メニューのバリデーション

ツールバーアイテムにメニューがついていて、Text Only モードやオーバーフローメニューになった場合は、**validateMenuItem :** が代わりに呼ばれるようになります。メニューがクリックされてプルダウンする直前に呼ばれますので、この中でメニューにチェックマークをつけるなどの処理も行うのが楽でいいでしょう。このサンプルでもこの中でチェックを付けています。

Controller.m > validateMenuItem :

```
- (BOOL) validateMenuItem : (NSMenuItem *) menuItem {

    if ( [ menuItem action ] == @selector( changeFontStyle : ) ) {
        if ( [ menuItem tag ] == fontStylePicked )
            [ menuItem setState : NSOnState ]; // チェックをつける
    }

    return YES; // or NO
}
```

NSMenuValidation : メニューが有効かどうかを返す

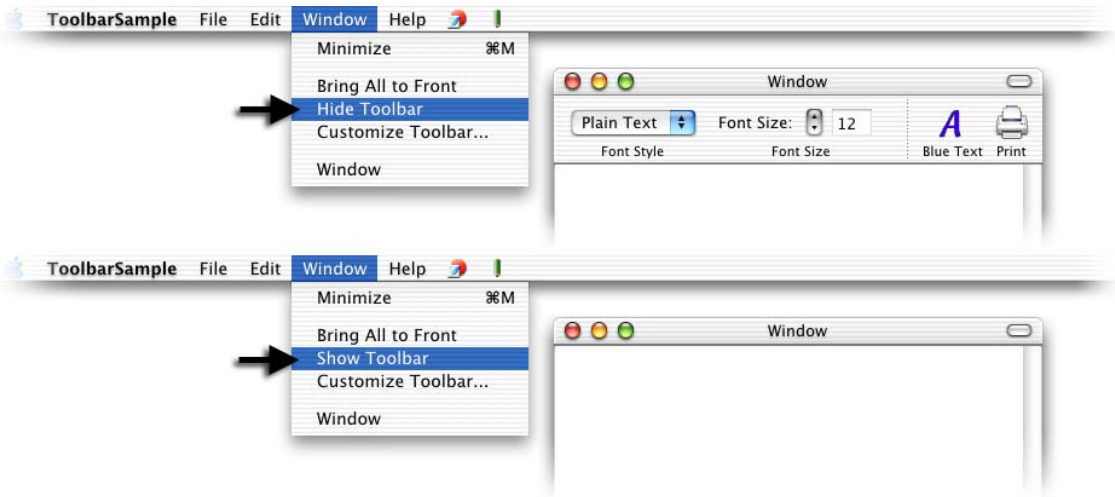
書式
```
- (BOOL) validateMenuItem : (id <NSMenuItem>) menuItem
```
入力
```
menuItem : 対象のメニューアイテム
```
出力
```
返り値     : 有効 = YES、無効 = NO
```

■ メニューバーからの操作

◎ ツールバーを隠す

ツールバーは不要な場合は、ユーザーが隠すことが出来るようになっています。ウィンドウの右上にあるボタンで隠したり表示したり出来るので、特に何もしなくてもとりあえずは大丈夫ですが、一般的にはウィンドウメニューや表示メニューなどに「 Hide Toolbar (ツールバーを隠す)」というメニューをつけることが多いです。このメニューは、ツールバーが隠されている場合は「 Show Toolbar (ツールバーを表示)」というタイトルに切り替わります。

このメニューを実現したい場合は、メニューを作って、ウィンドウの **toggleToolbarShown :** というアクションにコネクトするだけで作業は完了です。メニューのタイトルもツールバーの状態によって自動的に変わりますので、タイトルを付けておく必要もありませんし、コーディングの必要はありません。さらに、設定パレットを表示している間は、自動的に使えなくなります。



【図】Hide Toolbar メニューの挙動

このサンプルでは、toggleToolbarShown : のコネクトをウィンドウに対して行っていますが、複数のウィンドウを持つアプリケーションの場合は、ファーストレスポンダ (First Responder) にコネクトすることになります (この方が一般的だと思います)。こうしておくと、ツールバーを持たないウィンドウがアクティブな時にはメニューが自動的に使えなくなってくれます。このサンプルでは、アバウトパネルをアクティブにしておくとメニューが使えなくなります。

◎ 設定パレットの表示

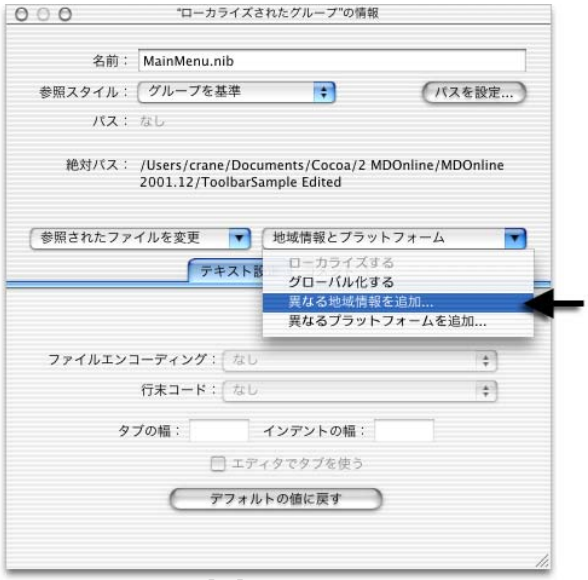
設定パレットの表示も同様です。メニューを作って、ウィンドウの **runToolbarCustomizationPalette :** アクションにコネクトするだけです。このメニューも同様に、ファーストレスポンダにコネクトするのが一般的といえます。コネクトをするだけで、設定パネルが表示されて、閉じられるまでの操作も全てフレームワークが行ってくれます (ツールバーからアイテムを取り除いた時の爆発のアニメーションも自動です)。

■ ツールバーの日本語化

◎ nib ファイルの日本語化

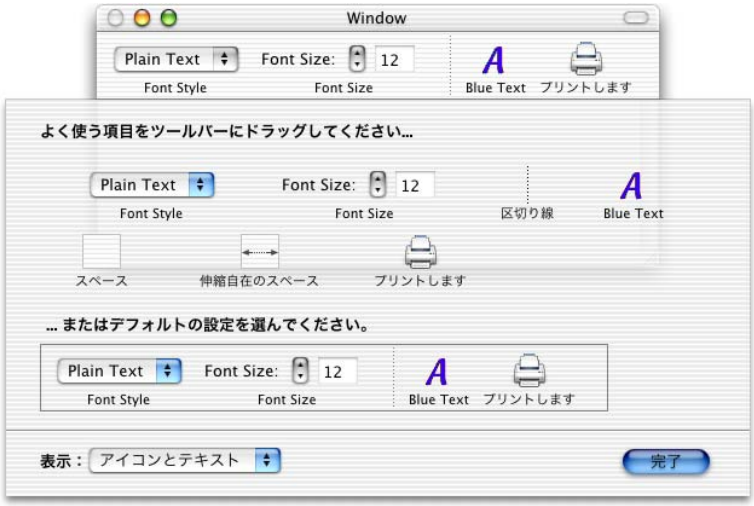
ここでは、ツールバーの日本語化を扱っていきます。一般のアプリケーションの日本語化と基本的に同じですが、フレームワークによって自動的に日本語化される部分もありますので、その辺りに注目してください。

まず、Project Builder で MainMenu.nib を選択して、「 プロジェクト → 情報を見る 」メニューを選択します。



【図】情報パネル

「 地域情報とプラットフォーム 」というポップアップボタンから「 異なる地域情報を追加... 」を選択して、「 Japanese 」と入力して OK ボタンをクリックすると、日本語用の MainMenu.nib が追加されます。このまま何もせずに、ビルドして実行すると以下のようになります。



【図】日本語の MainMenu.nib による設定パレット

◎ アイテム辞書の日本語化

シート上の説明も日本語になりますし、標準アイテムのラベルなども自動的に日本語になりますので、ほとんど何もする必要はありません。ただし、カスタムアイテムは自動的に日本語になりませんので、コードの一部をマルチリンガル対応する必要があります。このサンプルでは、`awakeFromNib` を修正するのが一番簡単ではないかと思います。Blue Text アイテムのラベルだけ日本語化してみたソースが以下のものです。

Controller.m > awakeFromNib

```
    // Blue Textを辞書に追加 ( イメージアイテム )
    addToolbarItem( toolbarItems,
                    @"BlueLetter", NSLocalizedString( @"BlueLetter", @"" ),
                    @"Blue Text",
                    @"This toggles blue text on/off",
                    self, @selector(setImage:),
                    [ NSImage imageNamed : @"blueLetter.tif" ],
                    @selector(blueText:), fontSizeMenu );
```

`NSLocalizedString` 関数は、プロジェクト内の **`Localizable.strings`** というファイルを参照しますので、このファイルを作って MainMenu.nib と同様の手順でマルチリンガル化して、日本語用のファイルに以下のように記述しておきます。文字コードは Unicode にしておきます（「 形式 → ファイルエンコーディング → Unicode 」メニューを選択します)。

Localizable.strings > Japanese

```
"BlueLetter" = "文字を青くする";
```

すると、以下のように日本語化されます。

【図】ツールバーアイテムを日本語化したところ

メニューは以下のようになります。「 Hide Toolbar 」は自動的に日本語化されますが、「 Customize Toolbar... 」は日本語化されませんので、こちらは編集する必要があります。

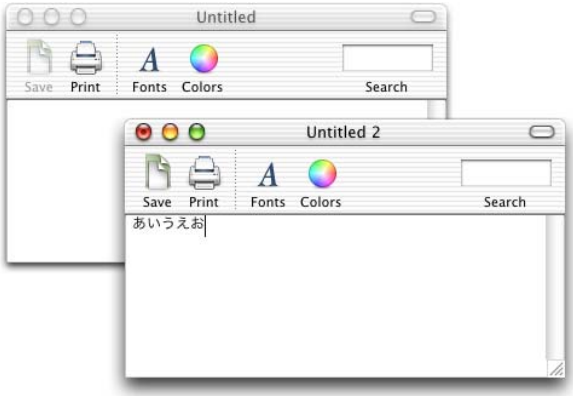
【図】日本語の MainMenu.nib によるメニュー

■ もう 1 つのサンプル : SimpleToolbar

Apple から提供されているツールバーのサンプルはもう 1 つあります。これは、Developer Tools によってインストールされるもので、SimpleToolbar というものです。起動ドライブの以下の場所にあります。

/Developer/Examples/AppKit/SimpleToolbar/

こちらのサンプルはドキュメントベースのアプリケーションにツールバーを実装したものになっています。そのため、複数のウィンドウにツールバーを表示するようなアプリケーションの動きを追うにはこちらを使うのがいいかもしれません。



【図】SimpleToolbar のウィンドウ

この図を見ると、手前のウィンドウはテキストが編集されているために Save アイテムが有効になっていて、奥のウィンドウは無効になっていることが分かります。

また、ツールバーのインスタンス取得メソッドである `toolbar : itemForItemIdentifier : willBeInsertedIntoToolbar :` メソッドは、ToolbarSample とちょっと違った書き方になっていま

す。アイテム辞書を用意するのではなくて、アイテムの識別子を if 文でアイテムの数だけ比較するというやり方になっています。アイテム辞書を使う場合は便利ですが、全てのアイテムを統一的な処理で書けるようにする必要があります。複雑なツールバーアイテムを使っているような場合は、こういうやり方が楽なこともあるかもしれません。

MyDocument.m > toolbar : itemForItemIdentifier : willBeInsertedIntoToolbar :

```
- (NSToolbarItem *)    toolbar : (NSToolbar *) toolbar
         itemForItemIdentifier : (NSString  *) itemIdent
     willBeInsertedIntoToolbar : (BOOL       ) flag {
        : 省略
    if ( [ itemIdent isEqual : SaveDocToolbarItemIdentifier ] ) {
    } else if ( ... ) {
    } else if ( ... ) {
        : 省略
```

■ ツールバー作成手順

この記事は、サンプルコードの解説ということで書いていますので、実際に作る時の手順の説明にはなっていません。そこで、最後に手順を書いておきます。もちろん、この順番どおりでないと作れないわけではありません。

・Project Builder で、プロジェクトの作成。
・Interface Builder で、ビューアイテム用のカスタムビューを作成。
・デリゲート、ターゲットのインスタンスの作成 (アクションとアウトレットの作成)。
・ビューアイテムとターゲットのコネクト。
・ビューアイテム用のメニューの作成とターゲットとのコネクト。
・イメージアイテム用の画像の作成とプロジェクトへの登録。
・Project Builder でアプリケーション起動時のアイテム辞書作成や初期化モジュールのコーディング。
・デリゲートの 3 つの必須メソッドをコーディング (これでとりあえず動きます)。
・デリゲートの必須以外のメソッドのコーディング。
・日本語対応。

■ まとめ

個人的には Cocoa アプリケーションなので、ツールバーはグラフィカルな操作で作れるのかと思っていましたが、そうではなくてコーディングがそれなりに必要です。Interface Builder でのサポートがもっとあるとよいのですが、今後の改善に期待したいところです。とはいえ、フレームワーク側の補助もかなりあるため、慣れれば、さほどの時間をかけずにツールバーをつけることが可能です。是非チャレンジを。