今から始める Cocoa プログラミング》
文書ファイルを扱うアプリケーションを作る(18)
ローカライズ

Cocoa の大きな特徴は、アプリケーションのローカライズが考慮されているというところにある。文字列の扱いから、ユーザインタフェースの設計まで、ユーザが選択している言語に合わせて切り替えることができる。あらかじめそうした言語ごとのリソースを用意しておくことや、プログラミングへの配慮も必要となるのではあるが、いずれにしても、複数の言語に対応したアプリケーション作成は作りやすい環境となっている。こうした機能を利用する方法と、開発ツール上での効率的な使い方を説明しよう。

Cocoa のローカライズ対応機能

Cocoa でのローカライズ対応機能はいくつかあるが、まずいちばん目立つのは、メニューやウインドウといったユーザインタフェースの設計内容を、言語ごとに別々に定義できるということだ。つまり、英語システム向けには英語のダイアログボックス、日本語システム向けには日本語のダイアログボックスを用意しておく。そして、それらの中のコントロールは同一に扱えるようにしておき、ユーザが選択している言語に応じてあらかじめ用意されているダイアログボックスがきちんと機能するように表示されるということである。ただ、その場合、用意されていない言語はどうなるということになるだが、その場合はユーザのシステム環境設定にある設定に応じて、優先順位の高いものが表示される。全世界の言語を用意しないといけないということはないけども、少なくとも、日本語と英語のものを用意しておくというのは最低限の目標かもしれない。

ただ、こうしたリソースの 2 重化ということについては、問題を感じる人も多いだろう。たとえば、完全に出来上がってから、コピーをもとに別の言語のものを作るということならなんとかなるとしても、その後にダイアログボックスの中身に変更があったら、いちいち各言語のダイアログボックスを設定しなすのかということになる。だが、後で説明するように、そういう心配は必要ない。別の手段があるので、設定の変更を別の言語のダイアログボックスなど nib ファイルの中身に反映させるということができる。

一方、もう 1 つのローカライズは、NSBundle という Cocoa のクラスを使ったものだ。このクラスを利用すると、プログラムソースの中に入れ込んだ文字列を、言語設定に応じた適切なものに切り替えるということができる。たとえば、プログラムで出すメッセージは、ソース中でダブルクォーテーションを使って直接書き込むことが多いかと思う

が、NSBundle の機能を使えば、たとえば、英語ユーザには「Processing」、日本語ユーザには「処理中」という表示をさせることができる。このためにはあらかじめテーブルなどを用意するのだが、簡便な方法として、Localizable.strings というファイルを言語ごとのリソースのフォルダに用意しておき、そこから表引きさせるという方法がある。

ただ、こうした Cocoa 独自の方法だけでなく、Java の場合には、地域化リソースとして、言語ごとの文字列データなどをクラスとして持たせる機能があるので、そちらを使うという選択肢もある。なお、今回の記事では、NSBundle については解説しないで、nib ファイルの言語対応の部分だけを説明しよう。

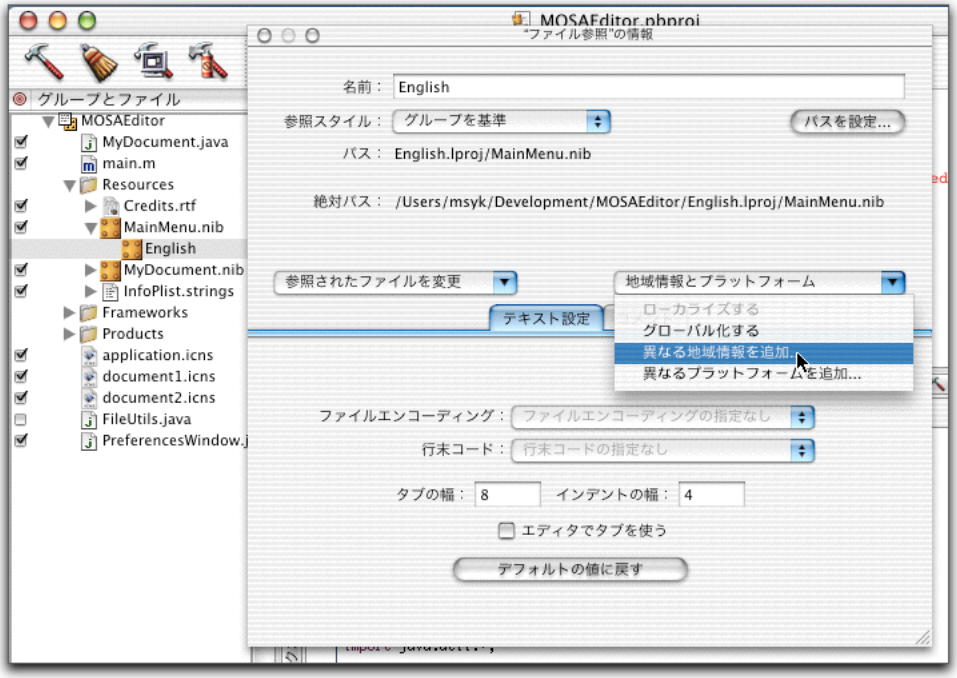
（ここで「リソース」という単語は、Mac OS のリソースとは違って、一般的な意味での供給されるデータと言うことである。）

言語ごとの nib ファイルを作成する

MOSAEditor では、最初から、MainMenu.nib が存在しているが、その nib ファイルは、英語対応のものである。Project Builder では、MainMenu.nib の項目の左に三角形があるが、それをクリックして下位の項目を表示すると、English という項目がでてくる。つまり、初期状態では英語のリソースだけがあるということだ。そこに日本語のリソースを追加してみよう。

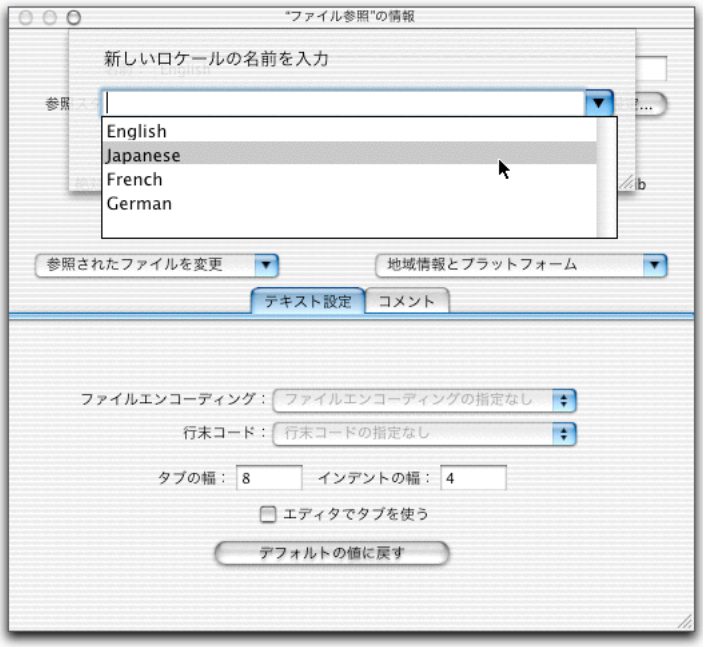
まず、MainMenu.nib の項目を選択して、「プロジェクト」メニューから「情報を表示」（Command+I）を選択する。すると、パレット形式の情報表示ウインドウがでてくる。このパレットは、選択した項目のものを表示するのであるが、邪魔な場合は Dock にしまうなり、消すなりしておけばよいだろう。MainMenu.nib ないしはその下の English が選択されていることを書くにして、情報パレットの「地域情報とプラットフォーム」のドロップダウンリストから「異なる地域情報を追加」を選択する。

別の弁護の追加を行う



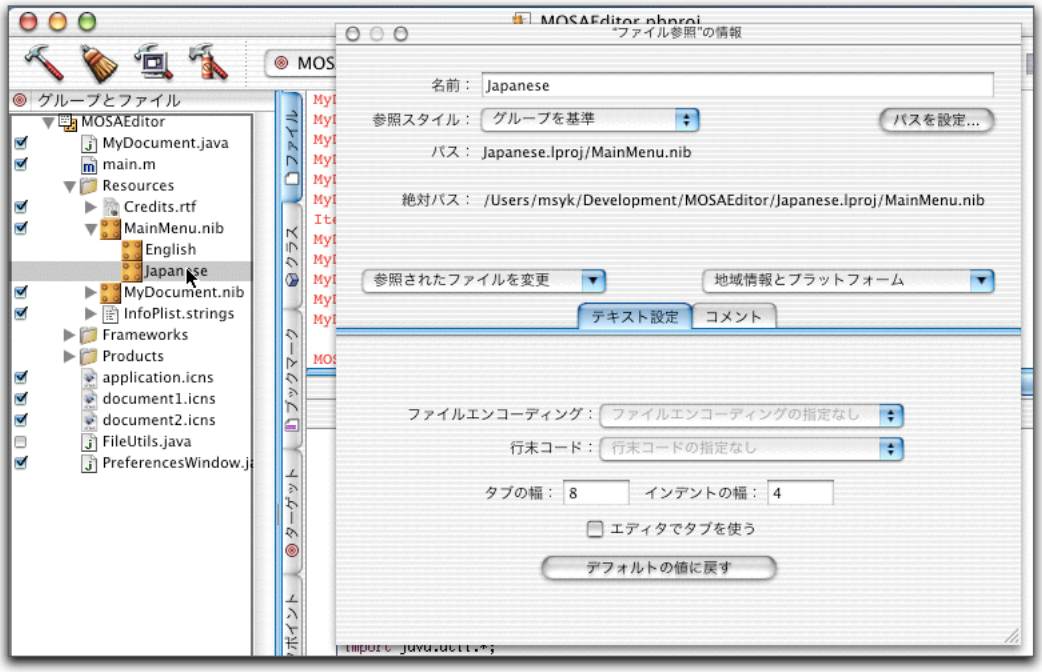
すると、シートがでてくるので、追加する言語を選択するドロップダウンリストがでてくる。ここで、日本語を追加する場合には、Japanese を選択する。

追加する言語を選択する



そうすると、Project Builder の MainMenu.nib の下には、Japanese という項目が追加する。つまり、MainMenu.nib に対して、English と Japanese の 2 つの言語のバージョンのリソースが加わったことになる。

日本語の MainMenu.nib が加わった



ここで、実際のプロジェクトのフォルダの中を必ず確認してもらいたい。プロジェクトのフォルダには、最初にいきなり MainMenu.nib があるわけではななかった。まず、English.lproj というフォルダがあり、その中に、MainMenu.nib というファイルがある（実際にはフォルダなのだが、ファイルのように見えるものはファイルと称することにしよう）。この、「言語名.lproj」というフォルダは、Mac OS X ではいろいろなところで見られる言語に対応したリソースを分類しておくフォルダだ。

新たに、日本語のリソースを、Project Builder にある MainMenu.nib に付け加えたのであるが、実際に作成されるのは、プロジェクトと同じフォルダにある Japanese.lproj というフォルダにある MainMenu.nib というファイルである。つまり、Finder で見えるレベルでは、言語ごと別々のフォルダに同じ名前のファイルが作成されたということになる。Project Builder では、ファイル名の下に言語が並ぶが、実際のファイル階層では、言語ごとのフォルダの下にファイルが存在するという逆の階層になっている点は注意すべきところである。

では、ここで、Project Builder で、MainMenu.nib の下にある Japanese をダブルクリックして開いて見よう。すでに日本語になっている…ということを期待しつつもさすがにそこまではなっていない。将来は分からないが、Dec 2001 版では、Japanese を新たに作った場合は、この場合だと English の MainMenu.nib がそのままコピーされているだけである。

ターゲットの設定では、こうして作成した各言語の nib ファイルは、パッケージアプリケーションにある Resources フォルダにコピーされる。コピーされるのは、English.lproj フォルダであり Japanese.lproj フォルダである。これらが作成されたパッケージでどのように配置されるかもチェックしておこう。なお、言語のフォルダだけがあって、そこの MainMenu.nib ファイルで正しい設定がされていないのであれば、その言語が選択されているときには、アプリケーションが全然機能しないことにもつながるので、きちんと設定していない言語リソースは削除しておくのが基本である。

言語ごとのリソースを管理する

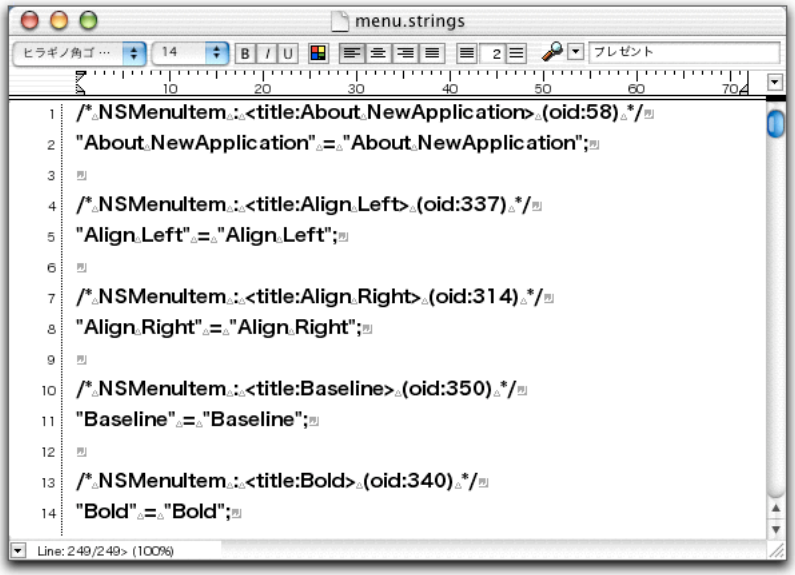
それでは、Japanese の MainMenu.nib の英語になっている項目を 1 つ 1 つ手作業で日本語化するということもあるわけだが、それではあまりに非効率的だ。そこで、nibtool というコマンドを使って作業を行なう。考え方は次の通りだ。まず、English をマスターとなるものとする。そして、マスターの中にある定義を、別途用意したテーブルをもとに変換をして、新たな定義を作成する…という具合の動作を行う。

nibtool は、Developer Tools のインストールによって、/usr/bin ディレクトリにコピーされるので、通常はコマンド名はそのまま利用できるはずだ。まず、この nibtool を使って、変換テーブルのひな形を作る。ここでは、MOSAEditor のプロジェクトのフォルダをカレントフォルダにしておく。つまり、English.lproj や Japanese.lproj があるフォルダをカレントしてにおくということだ。そして以下のようなコマンドを入力した。

```
% nibtool -L English.lproj/MainMenu.nib > menu.strings
```

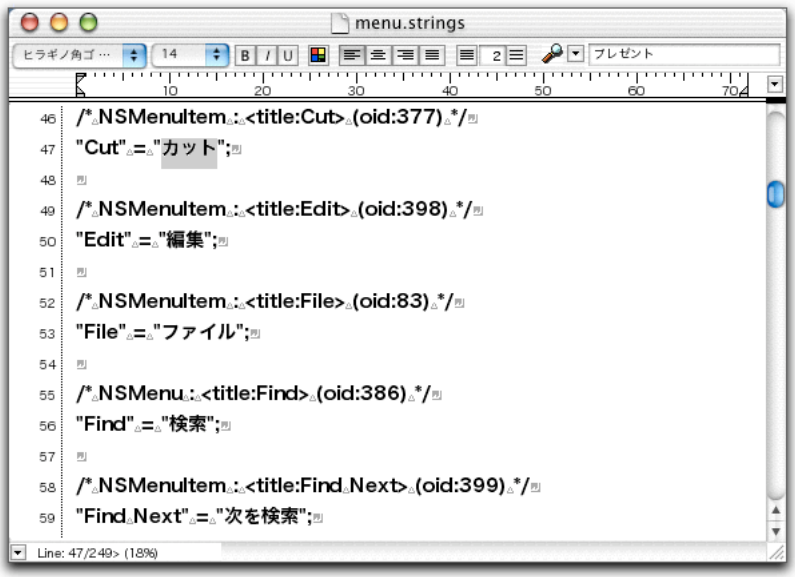
ここでは、nibtool のオプションとして、-L があり、出力結果をリダイレクトしている。この、-L というオプションは、その後の nib ファイルから、ローカライズすべき文字列を取り出して、テーブル形式で一覧するという機能を持っている。結果は標準出力に出すのだが、この状態では、プロジェクトのフォルダに、menu.strings という名前のファイルを出力することとなる。そのファイルをエディタで見てみると次の通りだ。

-L オプションで作成された menu.strings ファイル



コメントを除くと、メニューやダイアログボックスにある項目名が、イコールの左右に並んでいる。ここで、英語のリソースをもとに日本語のリソースを作るとすれば、次のように、イコールの左側はそのままにして、イコールの右側に日本語の文字列に入れ替える。ここでは作業を Jedit でやっているが、ファイルは Shift-JIS で保存してよさそうだ。

対応表を日本語で書き換える



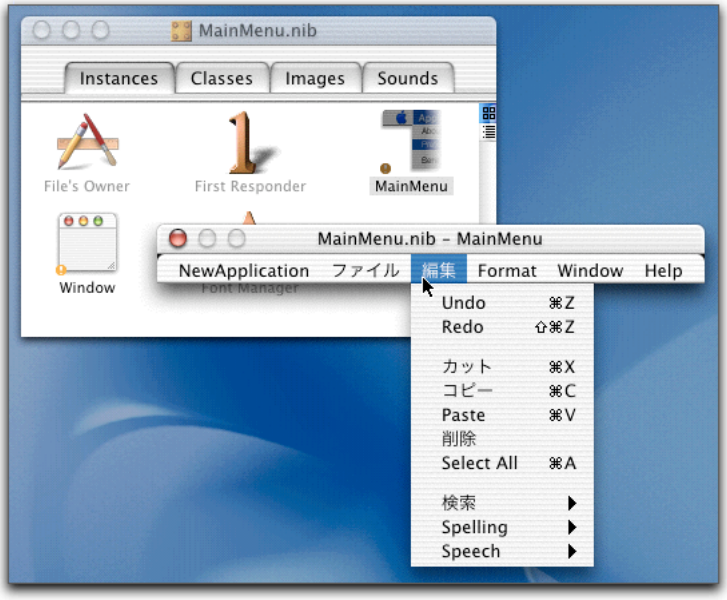
もちろん、全部の項目について、“日本語化”する必要がある（のだが、以下の説明は一部さぼっている。お許し頂きたい）。こうして、英語リソースを日本語リソースに変更する対応表ができた。続いて、次のようにコマンドを入力してみよう。もちろん、プロジェクトのフォルダがカレントになっている状態で入力する。

```
% nibtool -d menu.strings -W Japanese.lproj/MainMenu.nib English.lproj/MainMenu.nib
```

ここでは、nibtool コマンドについて、-d と-W のオプションがあるということになる。そして、一番最後のパラーメタである English.lproj/MainMenu.nib が元のリソースである。-d は変換テーブルの定義されたファイルを指定する。そして、-W オプションは変換結果を書き出す先だ。つまり、English 言語の MainMenu.nib を読み取り、menu.strings の結果に照らし合わせて文字列の置換をしたものを、Japanese 言語の MainMenu.nib ファイルとして書き込むというものである。

こうして、Japanese.lproj/MainMenu.nib は書き換えられた。そこで、再度 Interface Builder で開いてみる。Project Builder で、MainMenu.nib の下にある Japanese を開くわけだ。そうして、メニューなどを見てみると、確かに、menu.strings で書き換えを行った部分は、日本語になっている。

テーブルに従ってメッセージが日本語化された



nib ファイルは単にユーザインタフェースを定義するだけでなく、オブジェクトのインスタンス化やあるいは線でつないだ参照関係などさまざまな情報を含む。こうした情報をのなかの、メッセージだけを変換して、新たな nib ファイルを作るということで、ローカライズができるわけだ。

もし、たとえば、新しくメニューが増えたとしたら、そのときは、そのメニュー項目に対応した変換項目を、この場合だと、menu.strings に付け加えればよい。そして、前期の nibtool コマンドを実行すればいいわけだ。こうして、Interface Builder でのさまざま

な設定を引き継いだローカライズリソースが作成できる。とにかく作業は、英語版のリソースで設定をして、変換テーブルを追加変更してコマンドを走らせるということになる。もし、英語のメッセージに比べて日本語だと何倍にも長くなるようなら、レイアウトの変更も必要になる。そのときは、Japanese 言語にある nib ファイルを開いて編集すれば良いだろう。

nibtool で新たな nib ファイルを生成するのなら、わざわざ最初に Project Builder で Japanese のリソースを作る必要はないと思うかもしれないが、nibtool はバグとして、既存のファイルへの上書きしかできないことが明記されている。また、Project Builder でリソースを作っておけば、プロジェクトの登録や、ビルド関連の登録もされて、便利だということもあるわけだ。

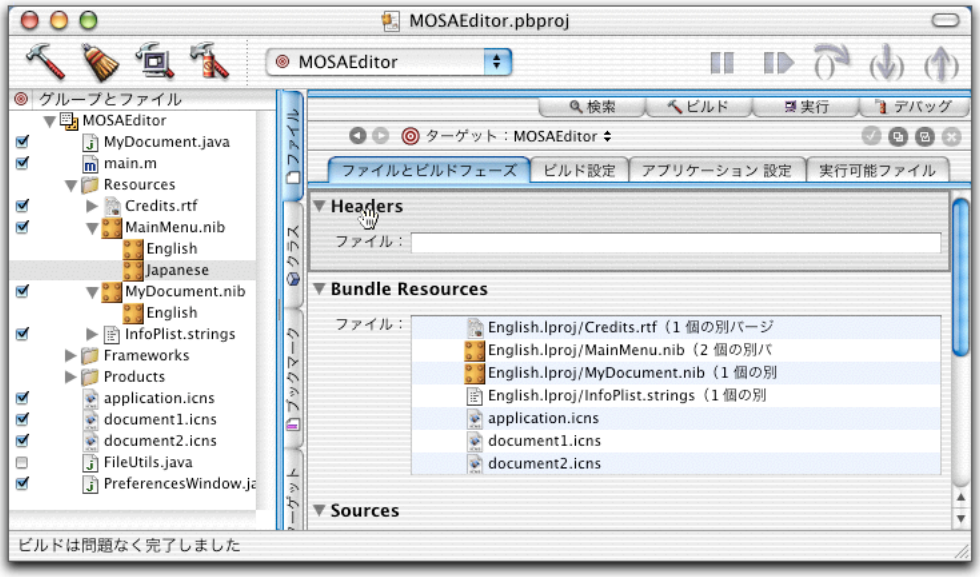
なお、nibtool を nib ファイルのすべての中身について適用するような利用方法として、前記のものを紹介したが、一部分の設定だけを取り出すなどさまざまな機能があるので、詳細なニーズがある場合には、man コマンドでどんな機能があるのかをチェックしてもらいたい。たとえば、変換テーブルの適用だけでなく、さらに第三の nib ファイルでレイアウト情報だけを取り出して適用するといった手法もある模様だ（-p オプション）。言語ごとに大きくレイアウトが違う場合で、後からの nib ファイル修正があるような場合にはそうした方法も取ることができるだろう。

nibtool の自動化

いちいち nibtool コマンドを入力するのも面倒だし、パラメータの指定も大変である。そういうときには、自動化してしまえばよい。Project Builder で、コンパイルしてパッケージを作る前に、前記の nibtool コマンドを実行するというわけである。Project Builder のターゲットの変種にある「ファイルとビルドフェーズ」というところで設定を行うが、1 つのターゲットで、複数のビルドフェーズを定義することができ、そのフェーズを順番に行うというのが基本となっている。

では、ビルドフェーズを追加して、nibtool コマンドを実行されるようにしておこう。「プロジェクト」メニューから「アクティブターゲットを編集」（Command+option+E）を選択するなどして、ターゲットの編集の状態にしておき「ファイルとビルドフェーズ」のタブを選択しておく。そして、とりあえずは、いちばん最初の Headers の領域を選択しておく。

「ファイルとビルドフェーズ」で 1 つのフェーズを選択しておく

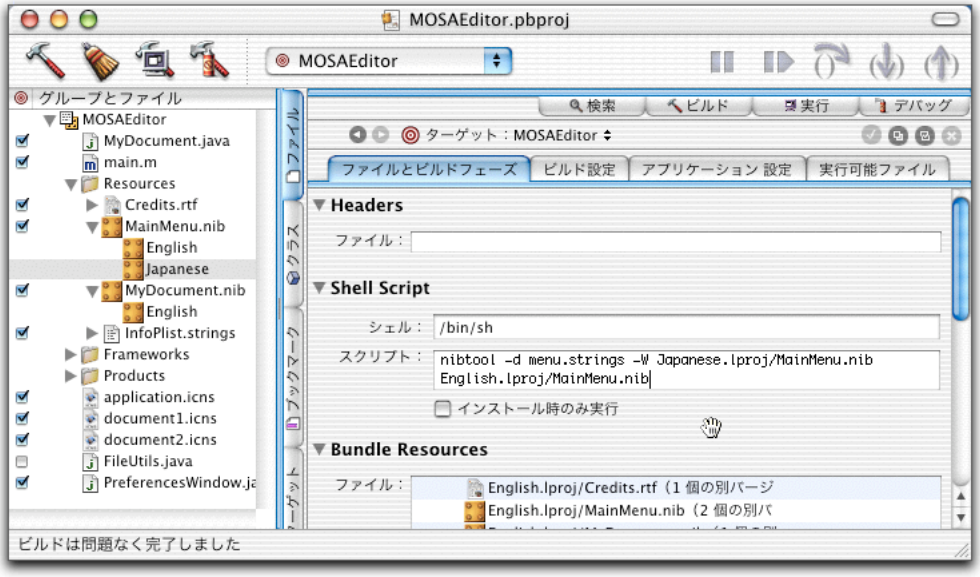


次に「プロジェクト」メニューの「新規ビルドフェーズ」から「新規フェルスクリプト・ビルドフェーズ」を選択する。

シェルスクリプトのビルドフェーズを追加する

そうすると、次のように、Shell Script というビルドフェーズが作成される。ここで、「スクリプト」として前記のコマンドを入力しておく。ここでは、プロジェクトのフォルダがカレントディレクトリとして機能するようなので、前のコマンドをそのまま入力すればよい。

スクリプトを定義した



こうすれば、アプリケーションんをビルドするときはいつも nibtool コマンドが実行され、そのつど、English 言語の MainMenu.nib の内容が変換されて、Japanese 言語の MainMenu.nib となることになる。ただし、nibtool の処理によって更新される内容によるのか、単にビルドしただけでは、nibtool による処理の結果はアプリケーションには反映されない。そこで、いったんクリアすることで、設定は反映されるようだ。毎回クリアするのも面倒だが、いずれにしても nib ツールの結果を反映させたいときには、手作業でクリアをするというのがいちばん手軽だろう。いずれにしてもビルドフェーズに定義しておくことで、nibtool の実行を忘れないでおくこともできる。

（この項、続く）