NSBundle

Inherits From:	NSObject
Conforms To:	NSBundle
Declared In:	foundation/NSBundle.h

Class Description

An NSBundle is an object that corresponds to a directory where program resources are stored. The directory, in essence, ªbundlesº a set of resources used by an application, and the NSBundle object makes those resources available to the application. It's able to find requested resources in the directory and, in some cases, dynamically load executable code. The term ªbundleº is used both for the object and for the directory it represents.

Bundled resources might include such things as:

- Images
- Sounds
- Character strings
- Nib filesÐfiles with a ª.nibº extensionÐarchived by Interface Builder
- Executable code

Each resource resides in a separate file.

Localized Resources

If an application is to be used in more than one part of the world, its resources may need to be customizedÐ*localized*Ðfor language, country, or cultural region. It may need, for example, to have separate Japanese, English, French, Hindi, and Swedish versions of the character strings that label menu commands. Its nib files might similarly need to be localized, as well as any images or sounds it uses.

The resource files specific to a particular language are grouped together in a subdirectory of the bundle directory. The subdirectory has the name of the language (in English) followed by a ª.lprojº extension (for ªlanguage projectº). The application mentioned above, for example, would have **Japanese.lproj**, **English.lproj**, **French.lproj**, **Hindi.lproj**, and **Swedish.lproj** subdirectories.

Each ª.lprojº subdirectory in a bundle has the same set of files; all versions of a resource file must have the same name. Thus, **myIcon.tiff** in **French.lproj** should be the French counterpart to the Swedish **myIcon.tiff** in **Swedish.lproj**, and so on.

If two or more languages share the same localized version of a file, the file can be stored in just one of the ª.lprojº subdirectories, while the other subdirectories keep (hard or soft) links to it. If a resource doesn't need to be localized at all, it's stored in the bundle directory itself, not in the ª.lprojº subdirectories.

The user determines which set of localized resources will actually be used by the application. NSBundle objects rely on the language preferences set by the user in the Preferences application. Preferences lets users order a list of available languages so that the most preferred language is first, the second most preferred language is second, and so on.

When an NSBundle is asked for a resource file, it provides the path to the resource that best matches the user's language preferences. In the following code, for example, the application sends a **pathForResource:ofType:** message to ask for the path to the **myIcon.tiff** file. With the path in hand, it can use other facilities (here NXImage's **initFromFile:** method) to access the resource.

```
NSString   *buf;
NSBundle   *bundle;
NXImage    *image;

bundle = [NSBundle bundleForClass:[self class]];
```

```
buf = [bundle pathForResource:"myIcon" ofType:"tiff"];
if (buf) {
    image = [[NXImage alloc] initFromFile:buf];
    . . .
}
```

The Main Bundle

Every application is considered to have at least one bundleÐits *main bundle*, the directory where its executable file is located. If the application is organized into a file package marked by a ª.appº extension, the file package is the main bundle.

Note: A file package is a directory that the Workspace Manager presents to users as if it were a simple file; the contents of the directory are hidden. A file package for an application includes the application executable plus other files required by the application as it runs. It bears the same name as the executable file but adds a ª.appº extension that identifies it to the Workspace Manager. For example, if you develop a Rutabaga application and place it in a **Rutabaga.app** directory with various ª.nibº and TIFF files that the application will use, the **Rutabaga.app** directory is its file package and its main bundle.

Other Bundles

An application can be organized into any number of other bundles in addition to the main bundle. These other bundles usually reside inside the file package, but they can be located anywhere in the file system. Each bundle directory is represented in the application by a separate NSBundle object.

By convention, bundle directories other than the main bundle end in a ª.bundleº extension, which instructs the Workspace Manager to hide the contents of the directory just as it hides the contents of a file package. The extension isn't required, but it's a good idea, especially if the bundle isn't already hidden by virtue of being inside a file package.

Dynamically Loadable Classes

Any bundle directory can contain a file with executable code. For the main bundle, that file is the application executable that's loaded into memory when the application is launched. The executable in the main bundle includes the **main()** function and other code necessary to start up the application.

Executable files in other bundle directories hold class (and category) definitions that the NSBundle object can dynamically load while the application runs. When asked, the NSBundle returns class objects for the classes (and categories) stored in the file. It waits to load the file until those classes are needed.

In the example below, the first line of code creates an instance of a class provided by an NSBundle object. If the class had not already been loaded into memory, asking for the class would cause it to be loaded.

```
id foo = [[[myBundle classNamed:"Reporter"] alloc] init];
if ( foo ) {
    [foo doSomething];
    . . .
}
```

By using a number of separate bundles in this way, you can split an application into smaller, more manageable pieces. Each piece is loaded into memory only when the code being executed requires it, so the application can start up faster than it otherwise would. And, assuming that only the rare user will exercise every part of the application, it will also consume less memory as it runs.

The file that contains dynamically loadable code must have the same name as the bundle directory, but without the ª.bundleº extension.

Since each bundle can have only one executable file, that file should be kept free of localizable content. Anything that needs to be localized should be segregated into separate resource files and stored in ª.lprojº subdirectories.

Instance Variables

None declared in this class.

Method Types

Initializing an NSBundle	- initWithPath:

Getting an NSBundle	+ bundleForClass: + bundleWithPath: + mainBundle
Getting a Bundled Class	- classNamed: - principalClass
Setting Which Resources to Use	+ setSystemLanguages:
Finding a Resource	- pathForResource:ofType: + pathForResource:ofType:inDirectory:withVersion:
Getting the Bundle Directory	- bundlePath
Stripping Symbols	+ stripAfterLoading:
Managing Localized Resources	- localizedStringForKey:value:comment: - localizedStringForKey:value:comment:table
Managing the Version	- bundleVersion - setBundleVersion

Class Methods

bundleForClass:

+ **bundleForClass:***classObject*

Returns the NSBundle object that dynamically loaded *classObject*, or the main bundle object if *classObject* was not dynamically loaded.

See also: + **mainBundle**

bundleWithPath:

+ (NSBundle *)**bundleWithPath:**(NSString *)*path*

Returns the NSBundle object that corresponds to the specified *path*. This method allocates and initializes the NSBundle object, if it doesn't already exist.

See also: + **mainBundle**

pathForResource:ofType:inDirectory:withVersion:

+ (NSString *)**pathForResource:**(NSString *)*name*
 ofType:(NSString *)*extension*
 inDirectory:(NSString *)*directory*
 withVersion:(int)*version*

Returns the path for the resource identified by *name*, having the specified filename *extension*, residing in *directory*, and having version number *version*.

See also: - **pathForResource:ofType:**, - **setBundleVersion:**, + **setSystemLanguages**:

mainBundle

+ **(NSBundle *)mainBundle**

Returns the NSBundle object that corresponds to the directory where the application executable (the file that's loaded into memory to start up the application) is located. This method allocates and initializes the NSBundle object, if it doesn't already exist.

In general, the main bundle corresponds to an application file package, a directory that bears the name of the application and is marked by a ª.appº extension.

See also: + **bundleForClass:**

setSystemLanguages:

+ **(void)setSystemLanguages:**(NSArray *)*languageArray*

Informs the NSBundle class of the user's language preferences. The argument, *languageArray*, is a pointer to an ordered list of null-terminated character strings. Each string is the name of a language.

Language names used for ª.lprojº subdirectories should match those set by this method. By convention, the names are in English. These are among the names currently in use:

English
French
German
Japanese
Spanish
Swedish

This method responds to a message sent by the Application Kit when the application first starts up; it's not necessary for your application to set the system languages.

stripAfterLoading:

+ (void)**stripAfterLoading:**(BOOL)*flag*

Sets whether symbols are stripped when modules are loaded. The default is YES. Note that NO makes bundles easier to debug.

Instance Methods

bundlePath

- (NSString *)**bundlePath**

Returns a pointer to the full pathname of the receiver's bundle directory.

bundleVersion

- (unsigned)**bundleVersion**

Returns the version last set by ***setBundleVersion:***, or 0 if no version has been set.

See also: - **setBundleVersion:**

classNamed:

- **classNamed:**(NSString *)*className*

Returns the class object for the *className* class, or **nil** if *className* isn't one of the classes associated with the receiver.

See also: - **principalClass**

initWithPath:

- **initWithPath:**(NSString *)*fullPath*

Initializes a newly allocated NSBundle object to make it the NSBundle for the *fullPath* directory. *fullPath* must be a full pathname for a directory.

If the directory doesn't exist or the user doesn't have access to it, the NSBundle is freed and this method returns **nil**. If the application already has an NSBundle object for the *fullPath* directory, this method frees the receiver and returns the existing object.

It's not necessary to allocate and initialize an object for the main bundle; the **mainBundle** method provides it.

See also: + **mainBundle**

localizedStringForKey:value:comment:

- (NSString *)**localizedStringForkey:**(NSString *)*key* **value:**(NSString *)*value* **comment:**(NSString *)*comment*

Returns a localized version of the string designated by *key*. *value* and *comment* are associated with *key* in the default table

See also: localizedStringForKey:value:comment:table:

localizedStringForKey:value:comment:table:

- (NSString *)**localizedStringForkey:**(NSString *)*key* **value:**(NSString *)*value* **comment:**(NSString *)*comment* **table:**(NSString *)*tableName*

Returns a localized version of the string designated by *key*. *value* and *comment* are associated with *key* in the table specified by *tableName*. If *tableName* is **nil**, the default table is used.

See also: - localizedStringForKey:value:comment:

pathForResource:ofType:

- (NSString *)**pathForResource:**(NSString *)*name* **ofType:**(NSString *)*extension*;

Returns the path for the resource identified by *name* having the specified filename *extension*.

See also: + **pathForResource:ofType:inDirectory:withVersion:**, + **setSystemLanguages**:, **- setBundleVersion:**

principalClass

- **principalClass**

Returns the class object for a class that's dynamically loaded by the NSBundle, or **nil** if the NSBundle can't dynamically load any classes. Classes can be loaded from just one file within the bundle directory, a file that has the same name as the directory (but without the ª.bundleº extension). If that file contains a single class, this method returns it. If the file contains more than one loadable class, this method returns the first one it encountersÐthat is, the first one listed on the **ld** command line that created the file. In the following example, Reporter would be the principal class:

```
ld -o myBundle -r Reporter.o NotePad.o QueryList.o
```

In general, the principal class should be the one that controls all the other classes that are dynamically loaded with it.

Before returning, this method ensures that any loadable code in the bundle directory has in fact been loaded into memory. If the NSBundle can load any classes at all, the principal class will be part of the executable image.

If the receiver is the main bundle object, this method returns **nil**. The main bundle doesn't have a principal class.

See also: - classNamed:

setBundleVersion:

- (void)**setBundleVersion:**(unsigned)*version*

Sets the version that the NSBundle will use when searching ª.lprojº subdirectories for resource files.

See also: - bundleVersion