Chapter 8

LOCALIZATION

CHAPTER 8 LOCALIZATION

Goal

To identify application resources requiring localization and to learn how to implement multi-lingual applications.

Prerequisites

Experience with NSString, NSBundle, and basic NIB-based applications.

Objectives

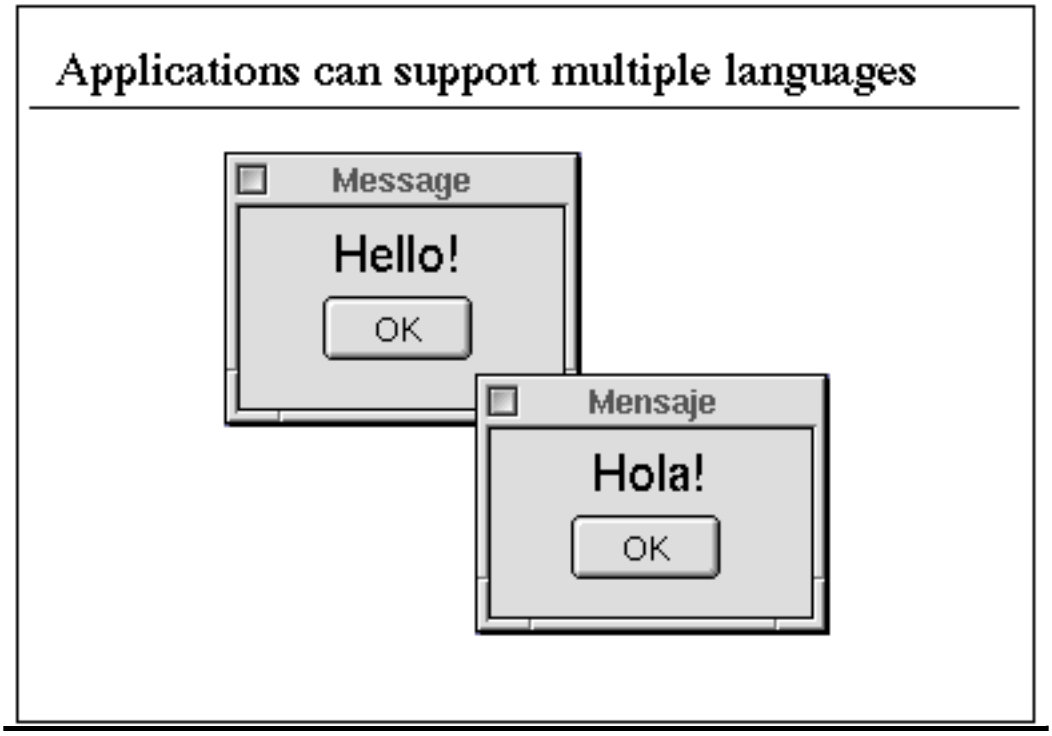
At the end of this lesson, you will be able to:

- Create multiple versions of NIB files, each localized to a particular language
- Dynamically load locale-specific strings and images for localization outside the NIB
- List other objects and interfaces that support localization

Reading

NSBundle class in the Foundation

Locales in Other Features of the Reference



Applications can support multiple languages

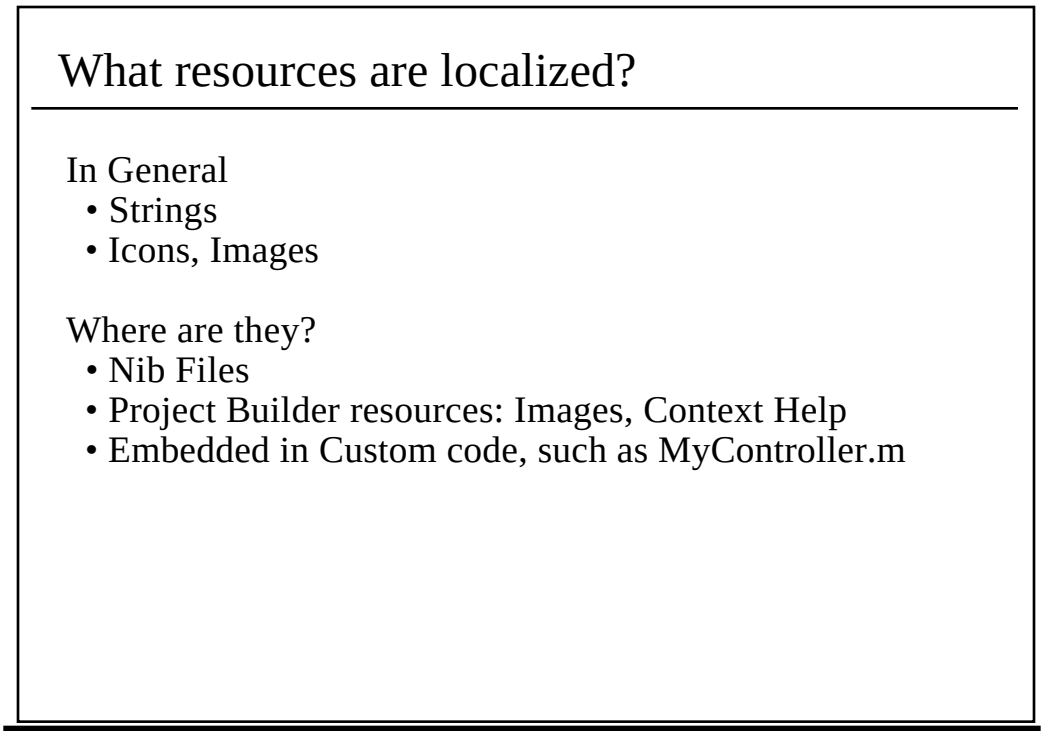
In the increasingly smaller world of international business and distributed computing, it is likely that your application needs to speak more than one language. It is easily possible that different users even at the same site will have individual language preferences. What are the concerns and how can multi-lingual support be implemented?

Users have individual language preferences

It is all driven by the user's individual preferences. A particular computer installation can have a default language for the system as a whole. A particular user can further specify a language preference, even a list of possible languages arranged in order of preference.

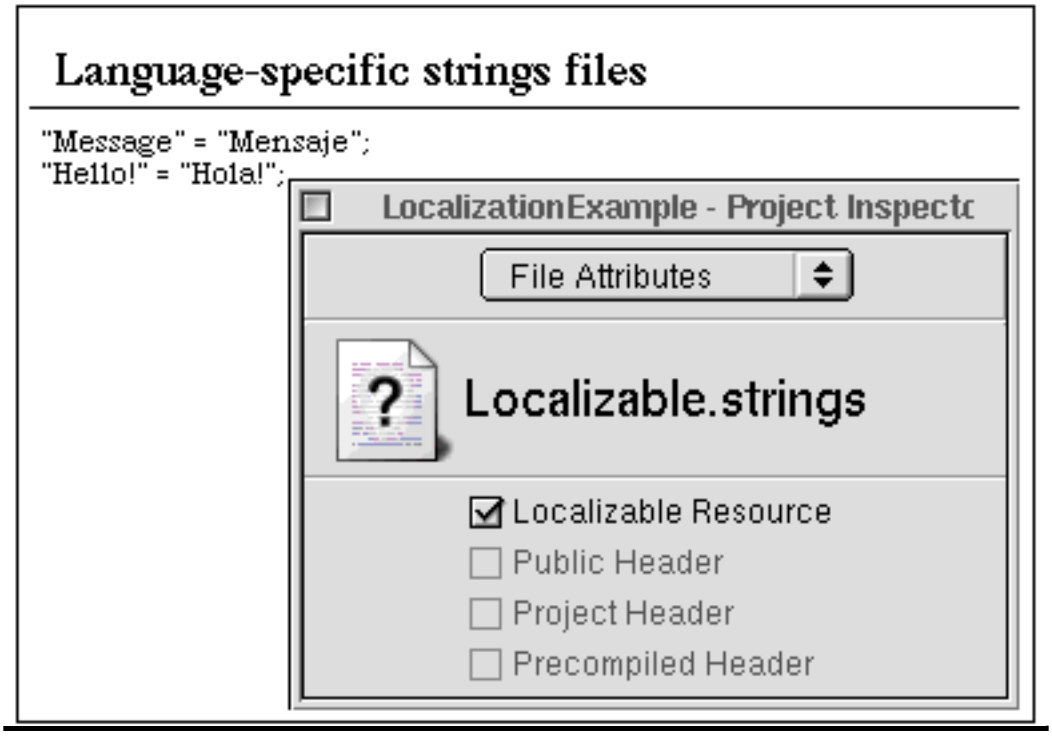
Available as a runtime resource, these preferences are accessible to your application, enabling it to dynamically handle locale-specific resources and speak in terms the user understands.

The configuration of site and user language preferences is platform specific and will not be covered here.



What resources are localized?

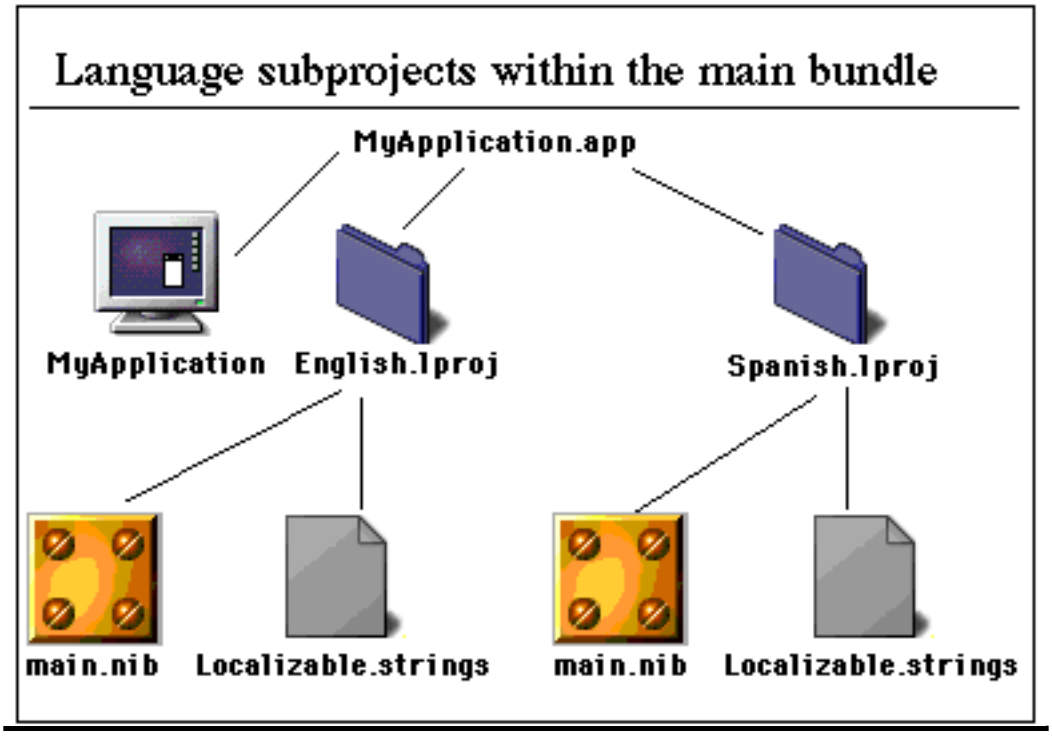
The most obvious locale-specific aspect of an application is the text in the user interface. Menu item titles, button labels, messages appearing on alert panels and on-line help all contain string object values written in a specific language. Besides alphabet, vocabulary and sentence structure, locales may have unique string representations for dates, numbers, currency and other symbols. It is easily possible that different languages require moderate differences in the user interface in general—the selection and layout of graphical components. The same label or concept expressed in different languages may differ substantially in length, requiring the size and placement of controls to be unique. With the different strings may go different mnemonics, accelerators and icons. With language differences often come cultural differences that may require different images, either more effective or more sensitive to inappropriate associations.



Language-specific strings files

String constants embedded in your code need to be extracted, placed in a string file and replaced with string tokens that map to the specific string value in the file. A string file is much like an NSDictionary in that it is a set of key-value pairs. The key is the string token used much like a variable name. The value is a locale-specific value that should be used where ever its symbolic string token appears in your code.

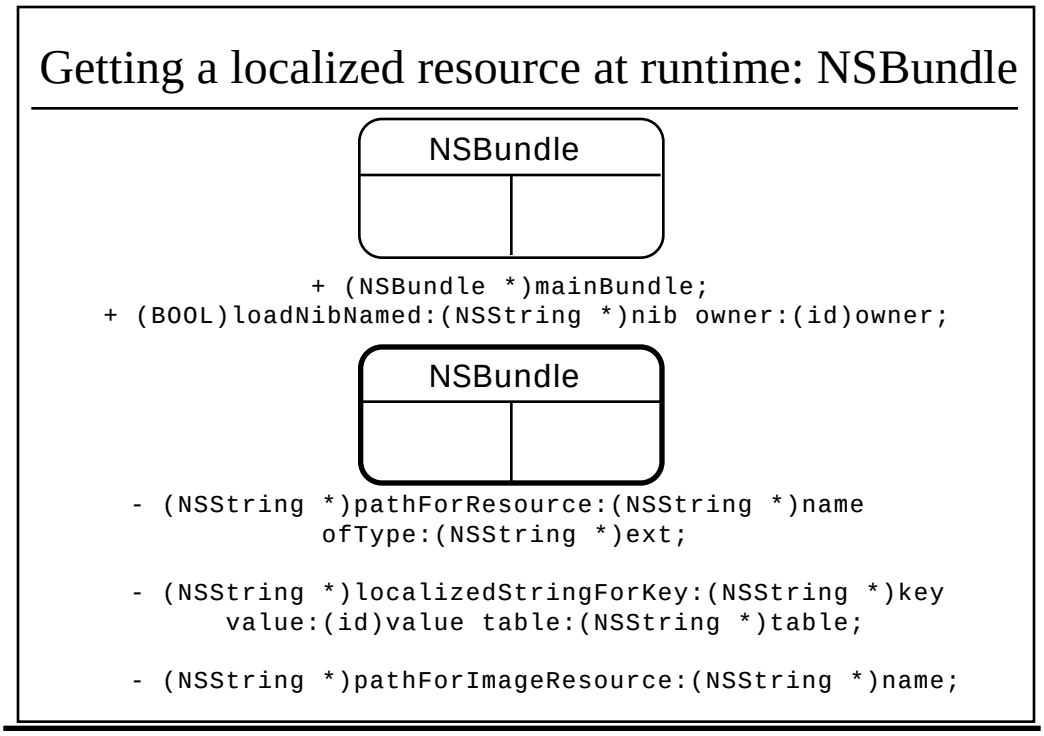
Once defined, the string file can then be replicated, the string values within each copy localized to the different language. Stored in an application's main bundle, the string files are available for dynamic loading by your application, based on the user's language preferences.



Language subprojects within the main bundle

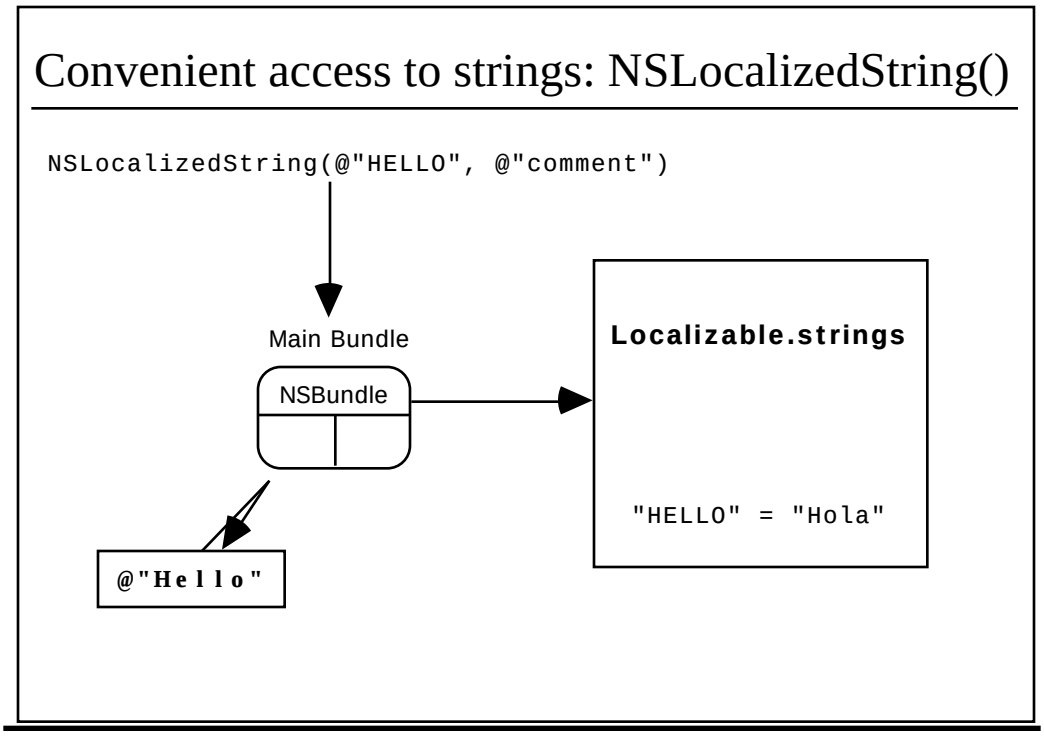
With Project Builder, application files can be flagged as “Localizable”. Within the project, these files are collected into language subprojects, directories under which a locale-specific version of each file is located. When built, the main bundle contains language project directories, one for each of the locales your application provides. Note that the same application resource file is replicated with the same name under each language subproject.

NIB and help files are automatically flagged as Localizable. The default language project is English and appears on the Project Builder inspector as the project attribute “Language”.



Getting a localized resource at runtime: NSBundle

Nib files, string files, even arbitrary file-based resources using a unique file type can all be dynamically located and loaded using NSBundle. NSBundle is aware of the user's language preferences as well as the set of locales, language subprojects, for which your application provides resources. Where multiple versions of a nib or string table are found, a particular one will be selected to suit the desired locale.



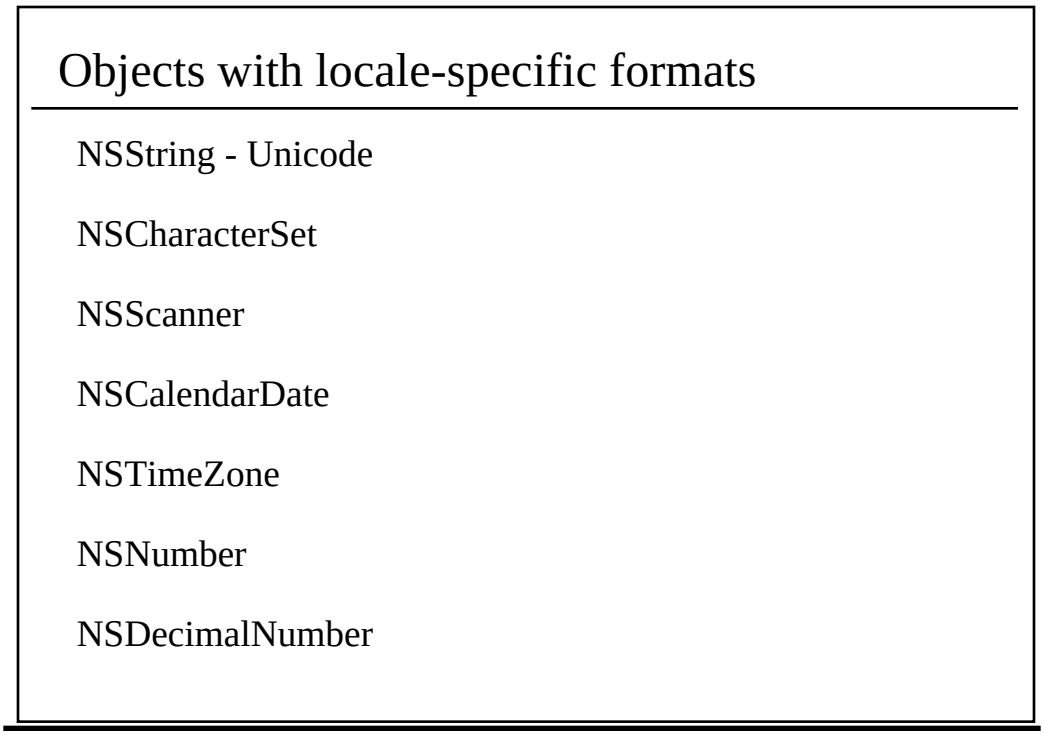
Convenient access to strings: NSLocalizedString()

NSBundle provides a macro for encapsulating the most common case—loading a string value from a default file name in the main bundle. If your application requires only one string file, you can name it **Localizable.strings** and access string values using **NSLocalizedString**. The first parameter is the string key, the second is a string comment used only for documenting the code by specifically indicating the intent of the string in the context of the application.

NSLocalizedString calls more general macros that you can use for cases where multiple string files are necessary, one per modular user interface component, or where you use multiple bundles internal or external to the application's main bundle:

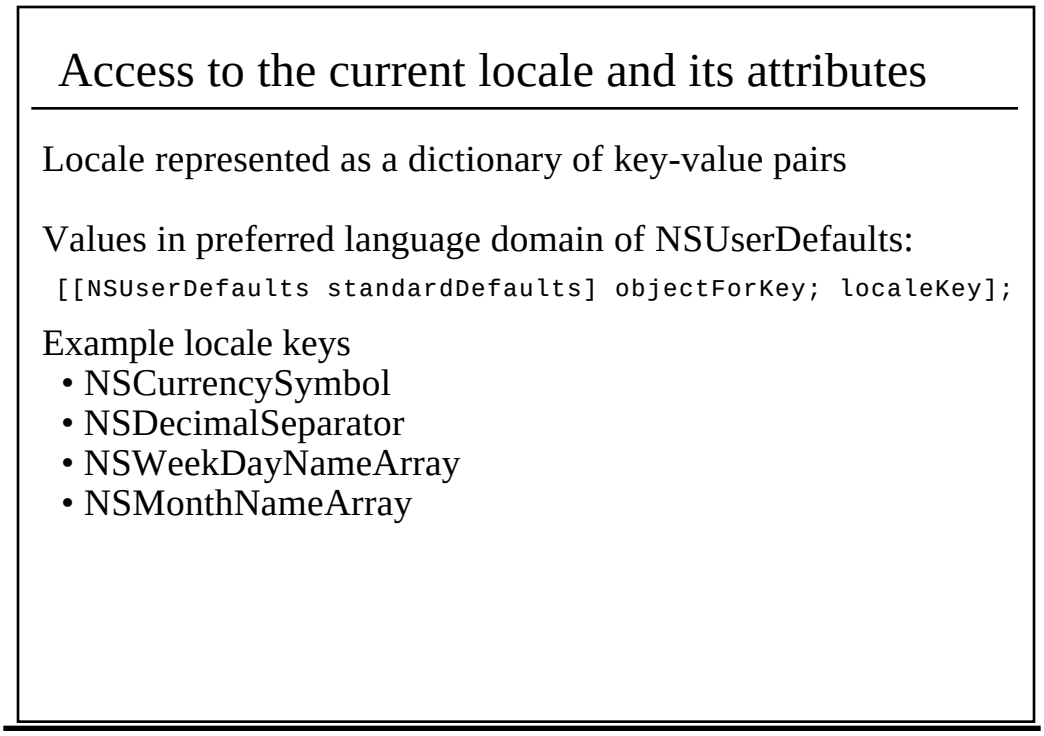
```
NSLocalizedStringFromTable(key, table, comment)

NSLocalizedStringFromTableInBundle(key, table,
bundle, comment)
```

All three of these are convenience functions that, after determining the correct bundle, use NSBundle's **localizedStringForKey:value:bundle** instance method.



Objects with locale-specific formats

A number of other Foundation objects have built-in locale support, dynamically sensing the user's current locale and adjusting the format and content of their string values to reflect it. NSString and NSCharacterSet are built around the Unicode international character encoding standard for full handling of alphabets. Calendar dates and time zones use locale-specific formatting as well as content strings like month and day. NSNumber and its subclasses use locale-specific decimal number representations. There are a number of ANSI functions for dealing with locales and retrieving a wide-range of parameters for proper localization support.



Access to the current locale and its attributes

A locale is a set of rules and values used to format locale-specific strings such as decimal numbers, currency figures, dates and times. These are packaged together in a dictionary of key-value pairs. The values are available from NSUserDefaults, stored in the preferred language domain. **NSUserDefaults.h** defines the set of available keys including such values as:

- NSCurrencySymbol—the string used to denote currency such as "$"
- NSDecimalSeparator—the string used in decimal numbers to separate the ones from the tenths place
- NSWeekDayNameArray—an array of strings giving the names for the days of the week
- NSMonthNameArray—an array of strings giving the full names for the months

Important ideas from this section

- Applications should provide multiple localizations to suit the preferences of a diverse multilingual user community. System-wide and user-specific language preferences are available to applications at runtime and direct the dynamic loading of localized resources.
- Application main bundles include language subprojects where localizable resources are replicated. Each instance is bound to a particular locale.
- NSBundle provides API for retrieving and loading file-based resources in language subprojects that best suit the user's language preferences.
- Several other objects encapsulate their own analogous dynamic behavior which is responsive to locale-specific formats and content when building and returning a variety of string values.

Class featured in this section

NSBundle