Locales Programming Guide

Data Management: Dates, Times, & Numbers

Contents

4

Tables

Introduction to Locales

Locales encapsulate information about linguistic, cultural, and technological conventions and standards. Examples of information encapsulated by a locale include the symbol used for the decimal separator in numbers and the way dates are formatted. Locales are typically used to provide, format, and interpret information about and according to the user's customs and preferences. They are frequently used in conjunction with formatters (see *Data Formatting Guide for Core Foundation*). Although you can use many locales, you usually use the one associated with the current user.

The operating system supplies data for dozens of different locales, regardless of which languages are installed.

Organization of This Document

The following articles explain what locales are, how they work, and common tasks you might perform with them:

- "Locale Concepts" (page 9) describes what locales are, why they are useful, and how they are identified. It also introduces the relationship between locales and user preferences.

- "Working With Core Foundation Locales" (page 13) explains how to create locale objects in Core Foundation, how to get the current user's locale, and how to use locale objects in conjunction with other objects. It also introduces aspects of the lifetime of a locale object.

Locale Concepts

Some computational tasks require information about the current user context to be able to process data—particularly when formatting output for presentation to the user or when interpreting input. A locale object provides a repository for that information. An operation that requires a locale object to perform its task is called **locale-sensitive**.

What is a Locale?

A locale is not a language; it's a set of conventions for handling written language text and various units (for example, date and time formats, currency used, and the decimal separator).

Conceptually, a locale identifies a specific user community—a group of users who have similar cultural and linguistic expectations for human-computer interaction (and the kinds of data they process). A locale's **identifier** is a label for a given set of settings. For example, "en" (representing "English") is an identifier for a linguistic (and to some extent cultural) locale that includes (among others) Australia, Great Britain, and the United States. There are also specific regional locales for Australian English, British English, U.S. English, and so on.

Practically, a locale is a set of default settings for a given identifier: A given locale object is simply a collection of settings. In Core Foundation, locales are represented by instances of CFLocaleRef. In Cocoa, with Mac OS X version 10.4 and later, locales are represented by instances of NSLocale.

In Mac OS X the locale preference need not be the same as the language preference—they are set independently. Users choose their locale in System Preferences > International, using the Region pop-up menu in the Formats pane. You can programmatically retrieve the array of language preferences from System Preferences using the key `AppleLanguages`. You retrieve the locale preference using the key `AppleLocale`.

Note that it is also possible for users to specify their own preferences, which override the system-defined defaults for the chosen locale (see "Interaction Between Locales and Preferences" (page 11)).

Why Are Locales Necessary?

When you display data to a user it should be formatted according to the conventions of the user's native country, region, or culture. Conversely, when users enter data, they may do so according to their own customs or preferences. Locale objects are used to provide information required to localize the presentation or interpretation of data. This information can include decimal separators, date formats, and units of measurement, as well as language and region information.

For example, by convention in the United States "7/4/76" represents the Bicentennial of the Declaration of Independence. However, in Great Britain, it represents the "7th of April, 1976"; in Thailand using the Thai Traditional Calendar it might represent "April 7th, 2519"; and in France it represents "7 avril 1976". To take a more subtle example, in the United States "12.125" represents the decimal number twelve and one eighth, whereas in Germany it represents twelve thousand one hundred and twenty-five.

Locale Naming Conventions

A locale's **identifier** is based on the naming convention defined by the International Components for Unicode (ICU). See http://icu.sourceforge.net/userguide/locale.html for information on their convention. The identifier consists of up to three pieces of ordered information: a **language code**, a **region code**, and a **variant code**.

The language code is based on the ISO 639-x/IETF BCP 47 standard. ISO 639-1 defines two-character codes, such as "en" and "fr", for the world's most commonly used languages. If a two-letter ISO 639-1 code is not available, then ISO 639-2 three-letter identifiers are accepted as well, for example "haw" for Hawaiian. For more details, see http://www.loc.gov/standards/iso639-2/php/English_list.php.

The region code is defined by ISO 3166-1 (see, for example, http://www.iso.org/iso/country_codes/iso_3166_code_lists/english_country_names_and_code_elements.htm). The region code is given in capital letters and appended, after an underscore, after the language code, for example "en_US", "en_GB", and "fr_FR".

The variant code is arbitrary and may have any number of keywords (which may be application-specific), each separated by an underscore. Developers are discouraged from using variant codes, as the format may change in the future.

Table 1 provides further examples.

Table 1   Components of a locale: language, country and variant

Locale ID	Language	Country	Variant	Definition
es_PE	es	PE		Spanish, Peru
es_ES	es	ES		Spanish, Spain
es_ES_PREEURO	es	ES	PREEURO	Spanish, Spain prior to Euro support
kl_GL	kl	GL		Kalaallisut, Greenland

Locale names such as "English", "French", and "Japanese" are deprecated in Mac OS X and are supported solely for backward compatibility. The Script Manager and all its concepts are deprecated. CFLocale never uses old-style Script Manager codes (except for one compatibility function, `CFLocaleCreateCanonicalLocaleIdentifierFromScriptManagerCodes`).

Note that you should typically have no reason to use locale identifiers directly in your code.

Locale Hierarchies

Locales are arranged in a hierarchy. At the root is the system locale, which provides default values for all settings. Below the root hierarchy are language locales. These encapsulate settings for language groups, such as English, German and Chinese (using identifiers "en", "de", and "zh"). Normal locales specify a language in a particular region (for example "en_GB", "de_AT", and "zh_SG").

When you look up a value in a locale, the receiver itself will be searched first for a value specific to that locale. If the value is not found in that locale, its parent is searched, and so on, up to the root locale.

Interaction Between Locales and Preferences

It is common to think of locales as providing information which is shared by a community of users. Every individual, however, may have their own preferences. To interpret input data from the current user, or to format data to display to the current user, you should use the user's locale. You access the logical "user" locale for the current user using `CFLocaleCopyCurrent` (see "Locale for the Current User" (page 14)). This returns a locale object which represents the settings for the current user's chosen system locale overlaid with any custom settings the user has specified in System Preferences.

There are four separate settings in International Preferences that involve language. The first three are set in the Languages tab, and the last is set in the Formats tab.

1. Primary language: affects user interface, used for other places where a *language* (not locale) preference is operative (e.g., Safari)

2. Collation (sort) order. This is set by default to match the primary language, but it can be overridden by the user. It only affects collation (localized string comparison).

3. Text break. This is set by default to match the primary language, but it can be overridden by the user. It only affects text boundary analysis (words, lines, and so on).

4. Region (Locale). This is set in Setup Assistant to combine the primary language at install time and the country specified in Setup Assistant. It can be changed by the user. If it doesn't match the primary language, the user gets a warning in the Languages tab.

Note that *only the setting in the Formats tab affects CFLocale properties*. The primary language has no influence on CFLocale properties—it only affects the user interface, not regional settings.

If a user should be able to specify their own settings for an application, the application can store those in the Preferences system (using the appropriate key). If the user updates their preferences (internally or externally to the application), these changes must be propagated through the application by synchronizing preferences and re-fetching Locale objects (see "Lifetime of Locale Objects" (page 14)).

Working With Core Foundation Locales

In Core Foundation, locales are represented by instances of CFLocale. You generally use the locale object for the current user, rather than for a specific locale, and typically in conjunction with other objects, usually formatters. Locale objects are currently immutable. It is important to understand the interaction between locales and the Preferences system.

Creating a Locale Object

Typically you want to format or interpret information for the current user. You generally therefore use the user's locale, which you access with `CFLocaleCopyCurrent` (see "Locale for the Current User" (page 14)). Sometimes, however, you may need to display information formatted for a particular locale, independently of the user's preference.

You can create a locale object for a specific locale with `CFLocaleCreate` by supplying the suitable identifier (see "Locale Naming Conventions" (page 10)).

```
CFStringRef localeIdent = CFSTR("fr_FR");
CFLocaleRef localeRef = CFLocaleCreate(kCFAllocatorDefault, localeIdent);
```

To ensure that you have the appropriate representation for a locale's identifier, you should use `CFLocaleCreateCanonicalLocaleIdentifierFromString` to create the canonical form—this function is especially useful for dealing with legacy information.

```
CFStringRef localeIdent = CFSTR("French");
CFStringRef stringRef =
    CFLocaleCreateCanonicalLocaleIdentifierFromString(kCFAllocatorDefault,
    localeIdent);
CFLocaleRef localeRef = CFLocaleCreate(kCFAllocatorDefault, stringRef);
```

There is no guarantee that information exists for the locale you specify. The following example creates a valid locale object (for the Thai language, country US). Mac OS X does not, however, provide specific data for this locale, so it will fall back first to "th" (Thai language), then to the root locale (see "Locale Hierarchies" (page 11)).

```
CFStringRef localeIdent = CFSTR("th_US");
CFLocaleRef localeRef = CFLocaleCreate(kCFAllocatorDefault, localeIdent);
```

The system default settings are available from the root locale using `CFLocaleGetSystem`. This function provides default values for all settings for all locales (see "Locale Hierarchies" (page 11)).

Locale for the Current User

Although a locale specifies defaults for a given language or region, individual users may specify their own settings, which override those for their chosen locale. Mac OS X provides locale objects that represent the settings for the current user's chosen system locale overlaid with any custom settings the user has specified. You access the logical user locale for the current user using `CFLocaleCopyCurrent`.

Even though settings for a user's locale may differ from those of the system defaults, the identifier for the two may be the same. For example, a user may choose British English as their language preference but specify custom date and time formatters. The following code sample illustrates a case where even though the formatters for the user and system locales are different, the locales' identifiers are the same.

```
systemLocaleRef = CFLocaleCreate(kCFAllocatorDefault, CFSTR("en_GB"));
// created "default" localeRef for "en_GB"
dateFormatter = CFDateFormatterCreate(kCFAllocatorDefault, systemLocaleRef,
kCFDateFormatterLongStyle, kCFDateFormatterMediumStyle);
CFShow(CFDateFormatterGetFormat(dateFormatter));
// output "d MMMM yyyy HH:mm:ss"
userLocaleRef = CFLocaleCopyCurrent();
CFShow(CFLocaleGetIdentifier(userLocaleRef));
// output "en_GB"
dateFormatter = CFDateFormatterCreate
    (kCFAllocatorDefault, userLocaleRef, kCFDateFormatterLongStyle,
kCFDateFormatterMediumStyle);
CFShow(CFDateFormatterGetFormat(dateFormatter));
// output "MMM' 'd', 'yyy h':'mm':'ss' 'a"
```

If you create a locale object for a given region, you get the defaults for that region even if the users has chosen that region as his or her default and provided his or her own preferences.

Lifetime of Locale Objects

Locale objects represent snapshots of settings at a particular time. For system default locale settings, values may change with different releases of the operating system. For a user's locale, users may at any point change their language preference, modify their preferred date format, or alter their measurement units.

Locale objects are currently immutable; in future versions of the operating system this may not be true. You should ensure that when you require an immutable locale object, you make an immutable copy of an existing locale object to use for as long as necessary.

The object you get back from `CFLocaleCopyCurrent` does not change when the user changes their Preferences settings. Moreover, the object itself may be cached by the runtime system, so successive calls of `CFLocaleCopyCurrent` may return the same object, even if a user has changed preference settings. If you want to ensure that your locale settings are consistent with user preferences, you must synchronize preferences and get a new locale object with `CFLocaleCopyCurrent`.

How often an application should synchronize preferences and refetch a locale object depends on the granularity of the operation in which it's used and on how responsive you want your application to be to changes in its environment. For example, a graphical application that displays rulers on screen should update

the ruler units as often as is appropriate. In a long-running report, however, it may be appropriate to cache the locale at the beginning of the procedure and use that throughout to ensure that the date format is consistent.

Using a Locale Object

You typically use locales in conjunction with other objects such as formatters, as shown in the following example.

```
userLocaleRef = CFLocaleCopyCurrent();
CFNumberFormatterRef numberFormatter =
    CFNumberFormatterCreate(kCFAllocatorDefault, userLocaleRef,
kCFNumberFormatterDecimalStyle);
```

You can also retrieve information such as decimal separators, date formats and units of measurement from a locale object:

```
userLocaleRef = CFLocaleCopyCurrent();
stringRef = CFLocaleGetValue(userlocaleRef, kCFLocaleDecimalSeparator);
```

To retrieve information such as the array of names of the days of the week, you can use use a formatter:

```
CFLocaleRef locale = CFLocaleCopyCurrent();
CFDateFormatterRef formatter =
    CFDateFormatterCreate (NULL, locale, kCFDateFormatterMediumStyle,
kCFDateFormatterMediumStyle);
CFArrayRef weekdaySymbols =
    CFDateFormatterCopyProperty (formatter, kCFDateFormatterWeekdaySymbols);
```

Document Revision History

This table describes the changes to *Locales Programming Guide*.

Date	Notes
2008-10-15	Corrected link to example of ISO 3166-1 country codes.
2008-07-11	Updated a code example showing how to retrieve weekday symbols.
2007-03-06	Corrected typographical errors.
2005-04-29	Updated for Mac OS X v10.4.
2004-08-31	Minor clarifications and corrections, including updated reference to ISO 639 standard.
2003-09-25	First version of *Locales*.