Data Formatting Guide for Core Foundation

Developer

Contents

Listings

Introduction to Data Formatting Guide for Core Foundation

Important: This is a preliminary document for an API or technology in development. Apple is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein for use on Apple-branded products. This information is subject to change, and software implemented according to this document should be tested with final operating system software and final documentation. Newer versions of this document may be provided with future betas of the API or technology.

Formatters define a common interface for creating, interpreting, and validating the textual representation of objects. The Core Foundation provides two opaque types, CFDateFormatter and CFNumberFormatter, that you use to format dates and numbers respectively. The Foundation framework provides two analogous classes, `NSDateFormatter` and `NSNumberFormatter`. Although the formatter objects in Core Foundation and Foundation are similar, they are not toll-free bridged.

This document assumes you are familiar with the concepts presented in *Data Formatting Guide*.

- Creating and Using CFDateFormatter Objects (page 5) describes how to create and use date formatters.
- Creating and Using CFNumberFormatter Objects (page 11) describes how to create and use number formatters.

Creating and Using CFDateFormatter Objects

Date formatters format the textual representation of date objects and convert textual representations of dates and times into date objects. You create date formatter objects by specifying a locale (typically the user's current locale) and a time style, you can also specify a custom format string.

Creating Date Formatters

You create a date formatter using the function `CFDateFormatterCreate`. You specify a locale for the format, and styles for the date and time parts of the format. You use `CFDateFormatterCreateStringWithDate` to convert a date to a textual representation.

CFDateFormatter defines several date and time format styles—short, medium, long, and full. It also defines a "none" style that you can use to suppress output of a component. The use of styles is illustrated in Use Formatter Styles to Present Dates and Times With the User's Preferences (page 5). The date and time styles do not specify an exact format—they depend on the locale, the user preference settings, and the operating system version. If you want an exact format, use the `CFDateFormatterSetFormat` function to change the format strings, as shown in Use Format Strings to Specify Custom Formats (page 8).

Use Formatter Styles to Present Dates and Times With the User's Preferences

The following code sample creates a date formatter that provides a full representation of a date using the `kCFDateFormatterLongStyle` style.

```
CFDateRef date = CFDateCreate(NULL, 123456);
CFLocaleRef currentLocale = CFLocaleCopyCurrent();

CFDateFormatterRef dateFormatter = CFDateFormatterCreate
       (NULL, currentLocale, kCFDateFormatterLongStyle, kCFDateFormatterLongStyle);

CFStringRef formattedString = CFDateFormatterCreateStringWithDate
        (NULL, dateFormatter, date);
```

```
CFShow(formattedString);

// Memory management
CFRelease(date);
CFRelease(currentLocale);
CFRelease(dateFormatter);
CFRelease(formattedString);

// Output (for en_US locale): January 2, 2001 2:17:36 AM PST
```

The following example shows the use of `kCFDateFormatterNoStyle` to suppress output of the time component.

```
CFDateRef date = CFDateCreate(NULL, 123456);
CFLocaleRef currentLocale = CFLocaleCopyCurrent();

CFDateFormatterRef dateFormatter = CFDateFormatterCreate
       (NULL, currentLocale, kCFDateFormatterShortStyle, kCFDateFormatterNoStyle);
CFStringRef formattedString = CFDateFormatterCreateStringWithDate
        (NULL, dateFormatter, date);
CFShow(formattedString);

// Memory management
CFRelease(date);
CFRelease(currentLocale);
CFRelease(dateFormatter);
CFRelease(formattedString);

// Output (for en_US locale): 1/2/01
```

The code sample shown in Listing 1 (page 7) formats a date value using different styles as a comparison. For the purposes of illustration, the sample specifies a particular locale.

Listing 1 Comparing date format styles

```
CFDateRef date = CFDateCreate(NULL, 123456);
CFStringRef enUSLocaleIdentifier = CFSTR("en_US");
CFLocaleRef enUSLocale = CFLocaleCreate(NULL, enUSLocaleIdentifier);

// Create different date formatters
CFDateFormatterRef shortFormatter = CFDateFormatterCreate
       (NULL, enUSLocale, kCFDateFormatterShortStyle, kCFDateFormatterShortStyle);
CFDateFormatterRef mediumFormatter = CFDateFormatterCreate
       (NULL, enUSLocale, kCFDateFormatterMediumStyle, kCFDateFormatterMediumStyle);
CFDateFormatterRef longFormatter = CFDateFormatterCreate
        (NULL, enUSLocale, kCFDateFormatterLongStyle, kCFDateFormatterLongStyle);
CFDateFormatterRef fullFormatter = CFDateFormatterCreate
        (NULL, enUSLocale, kCFDateFormatterFullStyle, kCFDateFormatterFullStyle);

// Create formatted strings
CFStringRef shortString = CFDateFormatterCreateStringWithDate
        (NULL, shortFormatter, date);
CFStringRef mediumString = CFDateFormatterCreateStringWithDate
        (NULL, mediumFormatter, date);
CFStringRef longString = CFDateFormatterCreateStringWithDate
        (NULL, longFormatter, date);
CFStringRef fullString = CFDateFormatterCreateStringWithDate
        (NULL, fullFormatter, date);

fprintf(stdout, "Short formatted date = %s\n",
        CFStringGetCStringPtr(shortString, CFStringGetSystemEncoding()));
fprintf(stdout, "Medium date = %s\n",
        CFStringGetCStringPtr(mediumString, CFStringGetSystemEncoding()));
fprintf(stdout, "Long formatted date = %s\n",
        CFStringGetCStringPtr(longString, CFStringGetSystemEncoding()));
fprintf(stdout, "Full formatted date = %s\n\n",
        CFStringGetCStringPtr(fullString, CFStringGetSystemEncoding()));

// Memory management
```

```
CFRelease(date);
CFRelease(enUSLocale);
CFRelease(shortFormatter);
CFRelease(mediumFormatter);
CFRelease(longFormatter);
CFRelease(fullFormatter);
CFRelease(shortString);
CFRelease(mediumString);
CFRelease(longString);
CFRelease(fullString);

// Output
Short formatted date = 1/2/01 2:17 AM
Medium date = Jan 2, 2001 2:17:36 AM
Long formatted date = January 2, 2001 2:17:36 AM PST
Full formatted date = Tuesday, January 2, 2001 2:17:36 AM PST
```

Use Format Strings to Specify Custom Formats

Typically, you are encouraged to use the predefined styles that are localized by the system. There are, though, broadly speaking two situations in which you need to use custom formats:

1. For fixed format strings, like Internet dates.
2. For user-visible elements that don't match any of the existing styles

Fixed Formats

To specify a custom fixed format for a date formatter, you use `setDateFormat:`. The format string uses the format patterns from the Unicode Technical Standard #35. The version of the standard varies with release of the operating system:

- OS X v10.9 and iOS 7 use version tr35-31.
- OS X v10.8 and iOS 6 use version tr35-25.
- iOS 5 uses version tr35-19.
- OS X v10.7 and iOS 4.3 use version tr35-17.

- iOS 4.0, iOS 4.1, and iOS 4.2 use version tr35-15.
- iOS 3.2 uses version tr35-12.
- OS X v10.6, iOS 3.0, and iOS 3.1 use version tr35-10.
- OS X v10.5 uses version tr35-6.
- OS X v10.4 uses version tr35-4.

The following example illustrates using a custom format string. (Note use of `yyyy` to format the year. A common mistake is to use `YYYY` which represents the week year, not the calendar year.)

```
CFLocaleRef currentLocale = CFLocaleCopyCurrent();
CFDateRef date = CFDateCreate(NULL, 123456);

CFDateFormatterRef customDateFormatter = CFDateFormatterCreate
        (NULL, currentLocale, kCFDateFormatterNoStyle, kCFDateFormatterNoStyle);
CFStringRef customDateFormat = CFSTR("yyyy-MM-dd*HH:mm");
CFDateFormatterSetFormat(customDateFormatter, customDateFormat);

CFStringRef customFormattedDateString = CFDateFormatterCreateStringWithDate
        (NULL, customDateFormatter, date);
CFShow(customFormattedDateString);

// Memory management
CFRelease(currentLocale);
CFRelease(date);
CFRelease(customDateFormatter);
CFRelease(customFormattedDateString);

// Output: 2001-01-02*02:17
```

Custom Formats for User-Visible Dates

To display a date that contains a specific set of elements, you use `CFDateFormatterCreateDateFormatFromTemplate`]. The function generates a format string with the date components you want to use, but with the correct punctuation and order appropriate for the user (that is, customized for the user's locale and preferences). You then use the format string to create a formatter.

To understand the need for this, consider a situation where you want to display the day name, day, and month. You cannot create this representation of a date using formatter styles (there is no style that omits the year). Neither, though, can you *easily and consistently* create the representation correctly using format strings. Although at first glance it may seem straightforward, there's a complication: a user from the United States would typically expect dates in the form, "Mon, Jan 3", whereas a user from Great Britain would typically expect dates in the form "Mon 31 Jan".

The following example illustrates the point:

```
CFStringRef dateComponents = CFSTR("yMMMMd");


CFLocaleRef usLocale = CFLocaleCreate(NULL, CFSTR("en_US"));
CFStringRef usDateFormatString =
    CFDateFormatterCreateDateFormatFromTemplate(NULL, dateComponents, 0, usLocale);
// Date format for English (United States): MMMM d, y


CFLocaleRef gbLocale = CFLocaleCreate(NULL, CFSTR("en_GB"));
CFStringRef gbDateFormatString =
    CFDateFormatterCreateDateFormatFromTemplate(NULL, dateComponents, 0, gbLocale);
// Date format for English (United Kingdom): d MMMM y
```

Creating and Using CFNumberFormatter Objects

Number formatters format the textual representation of number objects and convert textual representations of numeric values into number objects. The representation encompasses integers, floats, and doubles; floats and doubles can be formatted to a specified decimal position. You create number formatter objects by specifying a number style, you can also specify a custom format string.

Creating Number Formatters

To create a CFNumberFormatter, you must specify a locale and a formatter style as illustrated in Listing 1 (page 11), or a format string, as shown in Listing 2 (page 12). Format styles do not specify an exact format—they depend on the locale, user preference settings, and operating system version. If you want to specify an exact format, use the `CFNumberFormatterSetFormat` function to set the format string, and the `CFNumberFormatterSetProperty` function to change specific properties such as separators, the "Not a number" symbol, and the padding character.

Listing 1 Code sample showing how to create a number formatter using a formatter style

```
float aFloat = 1234.567;
int fractionDigits = 2;
CFLocaleRef currentLocale = CFLocaleCopyCurrent();

CFNumberFormatterRef numberFormatter = CFNumberFormatterCreate
        (NULL, currentLocale, kCFNumberFormatterDecimalStyle);
CFNumberRef maxFractionDigits = CFNumberCreate
        (NULL, kCFNumberIntType, &fractionDigits);
CFNumberFormatterSetProperty
       (numberFormatter, kCFNumberFormatterMaxFractionDigits, maxFractionDigits);
CFStringRef formattedNumberString = CFNumberFormatterCreateStringWithValue
        (NULL, numberFormatter, kCFNumberFloatType, &aFloat);

CFShow(formattedNumberString);
```

```
// Memory management
CFRelease(currentLocale);
CFRelease(numberFormatter);
CFRelease(maxFractionDigits);
CFRelease(formattedNumberString);

// Output (for en_US_POSIX locale): 1234.57
```

Listing 2 Code sample showing how to create a number formatter using a formatter string

```
float aFloat = 1234.567;
CFStringRef frLocaleIdentifier = CFSTR("fr_FR");
CFLocaleRef frLocale = CFLocaleCreate(NULL, frLocaleIdentifier);

CFNumberFormatterRef numberFormatter = CFNumberFormatterCreate
        (NULL, frLocale, kCFNumberFormatterNoStyle);
CFStringRef formatString = CFSTR("#.##");
CFNumberFormatterSetFormat(numberFormatter, formatString);
CFStringRef formattedNumberString = CFNumberFormatterCreateStringWithValue
        (NULL, numberFormatter, kCFNumberFloatType, &aFloat);

CFShow(formattedNumberString);

// Memory management
CFRelease(frLocale);
CFRelease(numberFormatter);
CFRelease(formattedNumberString);

// Output (for fr_FR locale -- note "," decimal separator): 1234,57
```

The following code fragment creates a number formatter that formats numbers as percentages using the `kCFNumberFormatterPercentStyle` number style. In this example, the `CFNumberFormatterCreateStringWithNumber` function converts the numeric value of `0.2` to a textual representation of `"20%"`.

```
// Creating a number formatter
float percent = 0.20;
CFNumberFormatterRef numberFormatter = CFNumberFormatterCreate
        (NULL, currentLocale, kCFNumberFormatterPercentStyle);
CFNumberRef number = CFNumberCreate(NULL, kCFNumberFloatType, &percent);
CFStringRef numberString = CFNumberFormatterCreateStringWithNumber
        (NULL, numberFormatter, number);
```

Using Number Format Styles

CFNumberFormatter defines several format styles. You set a formatter's style when you create the formatter. The code sample shown in Listing 3 (page 13) formats a numeric value using decimal, percentage, currency, and scientific notation styles. (The output format depends on user preference, so may vary in your application.)

Listing 3 Comparing number format styles

```
float n = 1.20;
CFNumberRef value = CFNumberCreate(NULL, kCFNumberFloatType, &n);
CFLocaleRef currentLocale = CFLocaleCopyCurrent();

// Create different number formatters
CFNumberFormatterRef decimalFormatter = CFNumberFormatterCreate
        (NULL, currentLocale, kCFNumberFormatterDecimalStyle);
CFNumberFormatterRef currencyFormatter = CFNumberFormatterCreate
        (NULL, currentLocale, kCFNumberFormatterCurrencyStyle);
CFNumberFormatterRef percentFormatter = CFNumberFormatterCreate
        (NULL, currentLocale, kCFNumberFormatterPercentStyle);
CFNumberFormatterRef scientificFormatter = CFNumberFormatterCreate
        (NULL, currentLocale, kCFNumberFormatterScientificStyle);

// Create formatted strings
CFStringRef decimalString = CFNumberFormatterCreateStringWithNumber
        (NULL, decimalFormatter, value);
CFStringRef currencyString = CFNumberFormatterCreateStringWithNumber
        (NULL, currencyFormatter, value);
```

```
CFStringRef percentString = CFNumberFormatterCreateStringWithNumber
        (NULL, percentFormatter, value);
CFStringRef scientificString = CFNumberFormatterCreateStringWithNumber
        (NULL, scientificFormatter, value);

// Print formatted strings to stdout
fprintf(stdout, "Decimal formatted number = %s\n",
        CFStringGetCStringPtr(decimalString, CFStringGetSystemEncoding()));
fprintf(stdout, "Currency number = %s\n",
        CFStringGetCStringPtr(currencyString, CFStringGetSystemEncoding()));
fprintf(stdout, "Percent formatted number = %s\n",
        CFStringGetCStringPtr(percentString, CFStringGetSystemEncoding()));
fprintf(stdout, "Scientific formatted number = %s\n",
        CFStringGetCStringPtr(scientificString, CFStringGetSystemEncoding()));

// Memory management
CFRelease(currentLocale);
CFRelease(decimalFormatter);
CFRelease(currencyFormatter);
CFRelease(percentFormatter);
CFRelease(scientificFormatter);
CFRelease(decimalString);
CFRelease(currencyString);
CFRelease(percentString);
CFRelease(scientificString);

// Output (for en_US_POSIX locale)
Decimal formatted number = 1.2
Currency number = $1.20
Percent formatted number = 120%
Scientific formatted number = 1.20000004768372E0
```

Custom Formatter Properties

Typically, you are encouraged to use the predefined styles that are localized by the system. If you want, however, you can change properties of number formatters using the `CFNumberFormatterSetProperty` function—see `CFNumberFormatterRef` for a complete list of the properties that can be changed using this function. For example, you can set the decimal separator to a comma, as shown in the following code fragment.

```
CFNumberFormatterRef decimalFormatter = CFNumberFormatterCreate
        (NULL, currentLocale, kCFNumberFormatterDecimalStyle);
CFNumberFormatterSetProperty(decimalFormatter,
        kCFNumberFormatterDecimalSeparator, CFSTR(","));
```

Using the formatter `decimalFormatter` above, you can convert a numeric value of `1.2` to a textual representation of `1,2`.

If you want to specify an exact format, use the `CFNumberFormatterSetFormat` function to set the format string. The format string uses the format patterns from the Unicode Technical Standard #35. The version of the standard varies with release of the operating system:

- OS X v10.9 and iOS 7 use version tr35-31.
- OS X v10.8 and iOS 6 use version tr35-25.
- iOS 5 uses version tr35-19.
- OS X v10.7 and iOS 4.3 use version tr35-17.
- iOS 4.0, iOS 4.1, and iOS 4.2 use version tr35-15.
- iOS 3.2 uses version tr35-12.
- OS X v10.6, iOS 3.0, and iOS 3.1 use version tr35-10.
- OS X v10.5 uses version tr35-6.
- OS X v10.4 uses version tr35-4.

For example, specifying the format string as `"$#,##0.00"` yields text representations such as `"$156.30"`.

The code sample shown in Listing 4 (page 15) formats different numeric values using `"$#,##0.00"` as the format string for currency values.

Listing 4 Using number format strings

```
CFLocaleRef currentLocale = CFLocaleCopyCurrent();
CFNumberFormatterRef customCurrencyFormatter = CFNumberFormatterCreate
```

```
    (NULL, currentLocale, kCFNumberFormatterCurrencyStyle);
CFNumberFormatterSetFormat(customCurrencyFormatter, CFSTR("$#,##0.00"));

float n1 = 6.3;
CFNumberRef number1 = CFNumberCreate(NULL, kCFNumberFloatType, &n1);
float n2 = 156.3;
CFNumberRef number2 = CFNumberCreate(NULL, kCFNumberFloatType, &n2);
float n3 = 1156.372;
CFNumberRef number3 = CFNumberCreate(NULL, kCFNumberFloatType, &n3);

CFStringRef string1 = CFNumberFormatterCreateStringWithNumber
        (NULL, customCurrencyFormatter, number1);
CFStringRef string2 = CFNumberFormatterCreateStringWithNumber
        (NULL, customCurrencyFormatter, number2);
CFStringRef string3 = CFNumberFormatterCreateStringWithNumber
        (NULL, customCurrencyFormatter, number3);

fprintf(stdout, "Format of %f = %s\n",
        n1, CFStringGetCStringPtr(string1, CFStringGetSystemEncoding()));
fprintf(stdout, "Format of %f = %s\n",
        n2, CFStringGetCStringPtr(string2, CFStringGetSystemEncoding()));
fprintf(stdout, "Format of %f = %s\n\n",
        n3, CFStringGetCStringPtr(string3, CFStringGetSystemEncoding()));

// Memory management
CFRelease(currentLocale);
CFRelease(customCurrencyFormatter);
CFRelease(number1);
CFRelease(number2);
CFRelease(number3);
CFRelease(string1);
CFRelease(string2);
CFRelease(string3);
```

```
// Output (for en_US_POSIX locale)
Format of 6.300000 = $6.30
Format of 156.300003 = $156.30
Format of 1156.371948 = $1,156.37
```

Document Revision History

This table describes the changes to *Data Formatting Guide for Core Foundation* .

Date	Notes
2014-02-11	Updated links to ICU documentation.
2011-03-08	Updated links to Unicode format specifications.
2006-12-21	Updated links to Unicode format specifications.
2006-10-03	Corrected a typographical error.
2005-10-04	Corrected minor typographic errors.
2005-08-11	Changed the title from "Data Formatting." Reorganized the content of previous articles.
2003-08-08	First version of this document.