Working with Dates in WebObjects

The approach of the new millennium has given rise to a number of concerns about how dates are being handled in computer programs. The so-called “Y2K” problem relates to both how dates are stored internally, and how they’re displayed and accepted as input.

Writing software that is “Y2K-compliant,” or modifying an existing program so that it becomes Y2K-compliant, requires that you have a solid understanding of the software package that you are using to represent and manipulate dates. Unfortunately, that package is typically complex, mirroring the fact that dates themselves are complex objects and that any package that works with dates needs to be extremely flexible in order to be useful in the largest number of situations.

In a WebObjects application, you work with date objects—which store the date values internally—and *formatters*—which translate between the internal representation and a textual one. While these objects can be used in a Y2K-compliant fashion, they can also be used in a way that is not Y2K compliant. This paper introduces the various objects you use when working with dates, and shows you how to avoid writing WebObjects applications that will fail in the new millennium.

Date Objects

On the server, WebObjects applications use a standard date object provided by the Foundation layer to contain date values. The NSGregorianDate class (NSCalendarDate in Objective-C) represents concrete date objects and performs date computations based on the Gregorian calendar. These date objects store a date as the number of seconds relative to the absolute reference date (January 1 2001, GMT).

On the client-side of a Java Client application, dates are represented using a version of NSCalendarDate that is a simple subclass of `java.util.Date`. Because the underlying object differs (NSGregorianDate on the server, `java.util.Date` on the client), the behavior of a given date object could conceivably change depending on whether it is running on the client or on the server (see the reference documentation for the NSGregorianDate class and the `java.util.Date` documentation provided with the JDK (and with WebObjects) for specifics on how these objects behave).

In a server-based WebObjects application, users frequently interact with date objects through WOTextFields (date values can also be displayed using WOStrings). To map between the textual representation of a date and the internal representation that a date object uses, Foundation relies on *formatter* objects.

Server-Side Date Formatters

WOTextField relies on a date formatter when formatting dates for display and interpreting user-entered dates. This formatter (NSGregorianDateFormatter for Java applications, NSDateFormatter for Objective-C and WebScript applications) allows great flexibility in formatting dates: “Thu 22 Dec 1994” and “12/12/94” are two possibilities.

The Gregorian date formatter has two attributes: a format string and a “natural language processing” flag. The format string is constructed from a set of date conversion specifiers (see Table 1-1) and explicitly specifies the display format or the expected date entry format. You specify the format that a WOTextField or WOString should use either by binding the format string to the element’s `dateFormat` attribute or by binding a date formatter that was instantiated with the desired format string to the element’s `formatter` attribute.

Table 1-1 Date Formatter Conversion Specifiers

Specifier	Description
%%	a ‘%’ character
%a	abbreviated weekday name
%A	full weekday name

Table 1-1 Date Formatter Conversion Specifiers (continued)

Specifier	Description
%b	abbreviated month name
%B	full month name
%c	shorthand for " %X %x", the locale format for date and time
%d	day of the month as a decimal number (01-31)
%e	same as %d without the leading 0 for days 1 through 9
%F	milliseconds as a decimal number
%H	hour based on a 24-hour clock as a decimal number (00-23)
%I	hour based on a 12-hour clock as a decimal number (01-12)
%j	day of the year as a decimal number (001-366)
%m	month as a decimal number (01-12)
%M	minute as a decimal number (00-59)
%p	AM / PM designation for the locale
%S	second as a decimal number (00-59)
%w	weekday as a decimal number (0-6), where Sunday is 0
%x	date using the date representation for the locale
%X	time using the time representation for the locale
%y	year without century (00-99)
%Y	year with century (such as 1990)
%Z	time zone abbreviation (such as PDT)
%z	time zone offset in hours and minutes from GMT (HHMM)

Natural Language Processing of Dates

The natural language processing flag adds an extra level of intelligence to the interpretation of strings: if the entered string doesn't match the date format string and natural language processing has been requested, the date formatter attempts to interpret the date anyway. This allows users to enter dates in a more colloquial fashion: “today,” “the day after tomorrow,” and “a month from today” are all valid input.

Although natural language processing might appear to be a good thing, in actual practice it introduces a level of ambiguity that is difficult to ignore. Imprecision in the English language is partly to blame: the word “Tuesday” by itself isn’t absolute: *which* Tuesday does it refer to? (The natural language processor assumes “next Tuesday.”) Phrases like “last Tuesday” and “next Tuesday” are interpreted as you would expect, and “next week” is translated as “one week from today.” But the natural language processor has its limits: “a week ago last Tuesday” gives the same result as “last Tuesday”—which may or may not be what you want.

The natural language processing logic can also be a problem when working with numeric dates when the date doesn’t match the format string or the format string isn’t supplied (In Objective-C, NSCalendarDate’s `dateWithNaturalLanguageString:` method engages the natural language processing logic directly, bypassing any attempt at matching to a format string). See “Years as Two or Four Digits,” below, for more information.

Disabling Natural Language Processing

Although the use of natural language processing is discouraged, WebObjects assumes its use by default. Further, when you bind a date object to a WOTextField or WOString, although there is an attribute that allows you to specify the format string directly in the binding, there isn’t a corresponding attribute that allows you to enable or disable natural language processing. In order to disable it, you must explicitly create and use a formatter object. You can create this object in a number of places: in your application’s constructor (or `init` method), in your component’s `awake` method, or in a particular method as needed. In Java, you can create a date formatter with natural language processing disabled like this:

```
NSGregorianDateFormatter dateFormatter =
    new NSGregorianDateFormatter(“%m/%d/%Y”, false);
```

In Objective-C or WebScript, do this:

```
_dateFormatter = [[NSDateFormatter alloc]
    initWithDateFormat:@”%m/%d/%Y” allowNaturalLanguage:NO];
```

Note that when applying a date formatter to a cell using Interface Builder in a Java Client application, you’re presented with a checkbox that allows you to select or de-select natural language processing. On the client side, the underlying objects are part of the JDK and have no such natural language processing ability. Thus, this checkbox has no effect.

Client-Side Date Formatters

Java Client applications use NSGregorianDateFormatter objects to translate between the internal representation of a date and a string representation. On the client, the NSGregorianDateFormatter class inherits from `java.text.SimpleDateFormat` (on the server, it inherits from NSFormatter). Although the NSGregorianDateFormatter is essentially the same class on both the client and the server, the client-side version doesn't support natural language processing. (For more information on how the presence or absence of natural language processing affects user input, see "Years as Two or Four Digits", below.)

Since NSGregorianDateFormatter exists on the client, you use the same date formatter conversion specifiers in your Java Client applications as you do for your server-based applications, with one exception: the `%c` format specifier isn't supported on the client and thus shouldn't be used in Java Client applications (`%c` is shorthand for "%X %x", the locale format for date and time).

Years as Two or Four Digits

The server-side date formatter has two format specifiers specifically for handling years: `%y` deals with years represented using two digits, while `%Y` deals with four-digit years.

The use of `%y` is not recommended due to possible misinterpretation of years before and after the year 2000. Specifying a date of "1/1/99" is conventionally understood to mean January 1, 1999, but it's less clear what is meant by a date of "1/1/00"; is this January 1, 1900, or January 1, 2000? The way in which a two-digit year is interpreted by the date formatter changed in WebObjects 4.0. Prior to 4.0, two digit years were all considered to be offsets from 1900: the year "05," for example, meant 1905. Recognizing that a format specifier of `%y` prevented the entry of years after 1999, however, the meaning of a two-digit year was changed in the version of Foundation that shipped with WebObjects 4.0: years from "00" to "29" are now interpreted relative to the year 2000, while "30" to "99" are relative to 1900, as before. Thus, “1/1/29” is assumed to be January 1, 2029, while “1/1/30” works out to January 1, 1930.

To further complicate the issue, if your server-side formatter uses `%y` (for instance, “%m/%d/%y”) and the user enters a date with a four-digit year (“1/1/2000”, for example), the formatter takes you at your word and only looks at the first two digits of the year. It then applies the algorithm described in the above paragraph, making those two digits relative to 1900 or 2000, depending on the version of WebObjects. So, for example, if you are running WebObjects 4.0 and your formatter uses “%m/%d/%y”, an entry of “1/1/2000” is interpreted as
“1/1/2020”—and is redisplayed as “1/1/20”.

Given the problems with the `%y` format specifier, most server-based WebObjects applications should use `%Y`. This format specifier accepts both two-digit and four-digit years. However, the way in which it interprets two-digit years depends on whether or not natural language processing is active, and is not always intuitive. For instance, with natural language processing turned off, two-digit years are taken literally: the year “29” really is the year 29. If natural language processing is used (meaning that it is turned on and the entered date doesn’t match the format string), however, two-digit years are interpreted as with `%y`: in versions of WebObjects prior to 4.0, they are offsets from 1900; in WebObjects 4.0 and later, “00”-”29” are offsets from 2000, while “30”-”99” are offsets from 1900.

Be aware that small variations in the user’s input can cause natural language processing to be enabled or disabled (assuming that it isn’t turned off). If your format string is “%m/%d/%y”, an entry of “1/1/99” doesn’t activate natural language processing, while an entry of “1-1-99” does.

The following two tables summarize this behavior (the first table applies to WebObjects 4.0 and later versions, while the second applies to WebObjects 3.51 and earlier versions):

Table 1-2 Summary of Year Behavior in WebObjects 4.0 and later

Year Entered	Year Stored in Date Object		
	%y	%Y, NLP enabled	%Y, NLP disabled
00[1]	2000	2000	nil[2]
29	2029	2029	29
30	1930	1930	30
99	1999	1999	99
1900	2019[3]	1900	1900
1929	2019[3]	1929	1929
1999	2019[3]	1999	1999
2000	2020[3]	2000	2000

[1]In WebObjects 4.0 and in earlier versions, there is a bug in NSDateFormatter that causes the current year to be used if natural language parsing is enabled, %y is specified, and a year of "00" is entered. This bug was fixed in WebObjects 4.0.1.

[2]In WebObjects 4.0 and later, the %Y format specifier doesn't accept "00" as a valid year.

[3]Although these years are stored in the date object as indicated in the table, only the first two digits are accepted as input by the %y date formatter. Then, because the %y formatter displays only the year within the century, they are displayed as "19" or "20" in these cases.

Table 1-3 Summary of Year Behavior in WebObjects 3.51 and earlier

Year Entered	Year Stored in Date Object		
	%y	%Y, NLP enabled	%Y, NLP disabled
00[1]	1900	1900	0
29	1929	1929	29
30	1930	1930	30
99	1999	1999	99
1900	1919[2]	1900	1900

Table 1-3 Summary of Year Behavior in WebObjects 3.51 and earlier (continued)

Year Entered	Year Stored in Date Object		
	%y	%Y, NLP enabled	%Y, NLP disabled
1929	1919[2]	1929	1929
1999	1919[2]	1999	1999
2000	1920[2]	2000	2000

[1]In WebObjects 4.0 and in earlier versions, there is a bug in NSDateFormatter that causes the current year to be used if natural language parsing is enabled, %y is specified, and a year of "00" is entered. This bug was fixed in WebObjects 4.0.1.

[2]Although these years are stored in the date object as indicated in the table, only the first two digits are accepted as input by the %y date formatter. Then, because the %y formatter displays only the year within the century, they are displayed as "19" or "20" in these cases.

For most WebObjects applications, the right-most column in the two tables above represent the recommended approach, combined with code to validate the year.

Validating Dates

Because of the possibly unexpected behavior when your formatter uses `%Y` and your application's user enters a two-digit year, you may want to disable natural language processing and then implement a validation method on your enterprise objects that "sanity-checks" the year portion of any entered date.

Objects that inherit from EOCustomObject or EOGenericRecord inherit a default implementation of EOValidation. The default implementation of EOValidation's `validateValueForKey` method searches for a method of the form `validate`*`Key`* (where *`Key`* is the name of one of a class's properties) and invokes it if it exists. Thus, as long as your objects inherit from EOCustomObject or EOGenericRecord, you need only implement the appropriate `validate`*`Key`* method to validate a given enterprise object property.

For more information on validating enterprise object attributes, see the online reference documentation for EOControl's EOValidation interface.

Two-Digit Years on the Client

The SimpleDateFormat object that's part of the `java.text` package uses different logic to interpret years entered with two digits. When the formatter specifies a two-digit ("abbreviated") year, SimpleDateFormat interprets the entry by adjusting the date to be within 80 years before and 20 years after the time the SimpleDateFormat instance was created. Thus, for example, using a pattern of "MM/dd/yy" and a SimpleDateFormat instance created on Jan 1, 1997, the string "01/11/12" would be interpreted as Jan 11, 2012 while the string "05/04/64" would be interpreted as May 4, 1964.

API Differences between Objective-C and Java

Unlike most of the classes in the Foundation framework, the Java and Objective-C versions of the date classes differ in some fairly important respects. As mentioned earlier, in Java you work with NSGregorianDate and NSGregorianDateFormatter objects, while in Objective-C you work with NSCalendarDate and NSDateFormatter objects. Although the NSGregorianDateFormatter and NSDateFormatter objects are closely related, the date objects themselves aren't.

The NSGregorianDate and NSCalendarDate classes are both designed specifically for working with dates that are based upon the Gregorian calendar. NSCalendarDate, however, is the more flexible of the two. When constructing or initializing an NSCalendarDate from an NSString, for example, you don't need an NSDateFormatter; NSCalendarDate has methods that allow you to specify the format string directly (`+ dateWithString:...` and `- initWithString:...`, respectively). NSGregorianDate, on the other hand, has no such methods. As well, the Objective-C version of NSDate has two class methods that return an NSDate object initialized solely from an NSString using natural language processing (`+ dateWithNaturalLanguageString:...`). These methods are not available to the Java programmer.

Finally, the Objective-C date classes allow you to take advantage of a locale dictionary that contains a set of implicit calendar formats. This dictionary is not used by the Java date classes.

For More Information

When working with date objects, be sure to read through the documentation carefully. Relevant documentation for server-based applications can be found under NSDate, NSGregorianDate, NSFormatter, and NSGregorianDateFormatter if you're programming in Java, and under NSDate, NSCalendarDate, NSFormatter, and NSDateFormatter if you're working in Objective-C or WebScript. (Be aware that this documentation has been written with the AppKit programmer in mind; you'll find numerous references to AppKit's NSCell class.) For programmers developing Java Client applications, the documentation for Sun's JDK 1.1.6 is installed along with WebObjects Developer and can be accessed through the WebObjects Info Center (under Reference->JavaSoft), or can be accessed over the Internet at `http://www.javasoft.com`.