LIONBRIDGE | Topic Brief

INTERNATIONALIZATION BEST PRACTICES FOR DATE-TIME FORMATTING

Date and time formatting issues occur in practically any software application, and there is a high probability that errors go silently undetected.

One best practice for handling date and time related information is also a general purpose best practice across all software development: keep data in native format as long as possible and only convert to other types when necessary at the outer boundaries of the application. For example, use date and time data types such as struct tm, time_t, Date, DateTime, etc., to hold date and time data and use them throughout your application processing. Only convert these values to character strings, if necessary, when performing operations such as displaying values in a user interface, accepting input from a user, or passing data to a system interface that requires string values. Never pass locale-formatted strings across a subsystem interface unless it is really required.

Native date and time data types store their data in a locale-neutral format so every consumer of that data knows its value exactly. Date-time formatting varies internationally, so the value “8/25/2008 9:12PM” in the United States could be represented as a number of different strings depending on the locale of the end user. Some examples of formatted strings in other locales are:

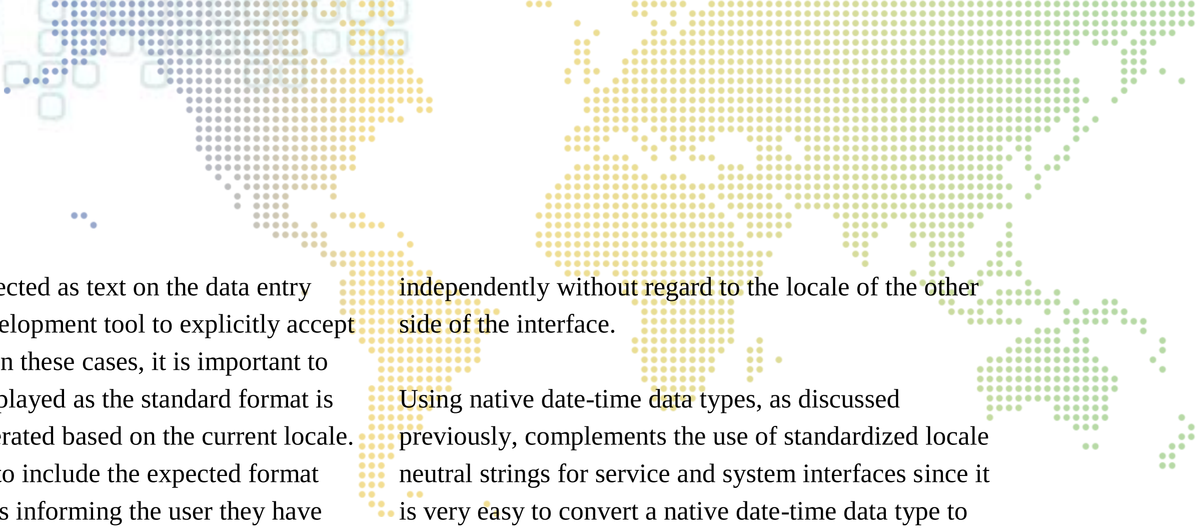
- 25/08/2008 21:12
- 25.08.2008 21:12
- 2008/8/25 下午 09:12
- 25/8/2008 PM 9:12
- 2008-08-25 21:12

Obviously, if the date format used includes a month or day name, the long format strings might be different for each language, even if the general short format were the same between locales

Many external system libraries or internal libraries within a development environment provide support that helps format and parse strings in these localized formats. However, due to the complexity and variation of formats, some are overly lenient in their attempt to parse and recognize the actual value represented. For example, some library functions or methods may attempt to parse the date string with a variety of separator characters. If an application is executing with a locale that uses a format of “MM/dd/yyyy” and a user from a different locale accidentally enters a value with a format “dd.MM.yyyy,” the application code silently accepts an incorrect date format. For example, if the user enters “05.02.2008,” it could be accepted and interpreted as May 2, 2008 rather than February 5, 2008 as the user intended.

This is particularly important in applications that support multiple locales, such as localized client applications or web pages communicating with the same server code. If an application carries date-time data values around in string variables, then it not only greatly increases the chance that a value is misinterpreted, but it also must retain the locale the date-time value was meant for throughout the application so the value can be interpreted correctly. Whereas converting the data to a date-time data type as soon as the value is entered allows the rest of the application to understand the value without question and without caring what locale the end user was using.

To help mitigate potential user data entry errors, in cases where dates and times are commonly entered and users can be from multiple locales, an application can



display the format expected as text on the data entry form and force the development tool to explicitly accept a certain format only. In these cases, it is important to ensure that the text displayed as the standard format is programmatically generated based on the current locale. It may also be helpful to include the expected format string in error messages informing the user they have entered an invalid value.

Another mechanism to help mitigate data entry errors is to separate date-times into their component parts for data entry—for example, different fields for each of the month, day, year, hour, minute, and am/pm indicator. In these cases, it is important to ensure that the development tools have the ability to arrange the fields in different orders for certain locales, that some fields may not appear in some locales (such as am/pm indicator) and that data values used as defaults or for validation can change based on locale (acceptable values for hours may be 1 through 12 or 0 through 23 depending on locale).

In some cases, especially service or system interfaces, date-time data is explicitly represented as a string. This allows two communicating processes to interact with each other without having to know the architecture and internal representation of native date-time data types of the other process. In these cases, the best practice is to always use a standardized locale-neutral date-time format for the strings being passed. For example, the W3C definition of the XML dateTime data type uses the same order of component fields, the same separator characters, and the same acceptable data values without regard to locale. So, the data value can be understood in any locale. This format is also the same as specified by the ISO 8601 standard and is generally accepted by default by development tools or products, such as databases.

This does require some extra work to convert between locale-neutral and locale-specific format each time the service boundary is exercised; however, it eliminates ambiguity, reduces the potential for misinterpretation, and allows each side of the interface to execute independently without regard to the locale of the other side of the interface.

Using native date-time data types, as discussed previously, complements the use of standardized locale neutral strings for service and system interfaces since it is very easy to convert a native date-time data type to and from a standardized locale-neutral string. These types of conversion functions are included in many development tools. In applications that contain little date-time information, or where it is not a common or a main component of the application, it can be acceptable to display data using a standard international format, such as ISO 8601, which will also make conversions from native data types to display formats easily accomplished in all locales.

In conclusion, a best practice end-to-end application uses locale-specific strings at the outer (end user facing) edge of the application, converts to native date-time data types as soon as possible, uses the native date-time data types for business logic and as many service and system interfaces as possible (such as a database interface) and converts native date-time data types to a standardized locale-neutral string format for other service and system interfaces that require this format.

About Lionbridge

Lionbridge (Nasdaq: LIOX) is the leading provider of translation, localization, and testing services. Organizations in all industries rely on Lionbridge language and testing services to increase international market share, speed adoption of products and content, and ensure the integrity of their global brands. Based in Waltham, Mass., Lionbridge operates across 26 countries, and provides services under the Lionbridge and VeriTest® brands.

Corporate Headquarters

Lionbridge
1050 Winter Street
Waltham, MA 02451
USA
www.lionbridge.com