INTERNATIONALIZATION OF DATA VALIDATION AND CHARACTER PROCESSING

Many applications contain some form of logic related to parsing or processing text entered by a user or text data received from a system interface. This logic can deal with the text value as a single element or it could attempt to process the individual characters. In either case, the logic could contain code and data specific to a certain locale that may not work internationally.

One of the most common expectations is that this occurs only with legacy code such as C and C++ that, by default, uses single byte character string data types. However, new applications based on code that, by default, use Unicode character string data types, such as Java and .NET, can also contain internationalization issues. Also, some issues may be concealed by implicit character processing in logic that is more specifically used for other data types such as numeric or date-time types.

Legacy code that is not Unicode enabled performs string processing on byte boundaries rather than on character boundaries. This can cause the code to interpret the value of the characters incorrectly. For example, if a double-byte Unicode string is read in by an application, the code may see the second byte has a value of zero, assume this is the end-of-string null byte indicator, and incorrectly think the string is only one character long. Similarly if a general multi-byte-character string is read in by an application, the code can interpret each byte as a character and may not be able to interpret the string correctly.

Some legacy applications can be made to run correctly for some additional locales; however, to gain true internationalization the code generally must be modified to use “wide character” strings and functions to interpret multi-byte data correctly. Often times this process is very involved, especially for code that performs pointer arithmetic to navigate through strings, searches for characters or substrings in a string, or otherwise parses and processes strings and substrings.

An alternative fast-track to multi-byte enabling (to avoid re-writing code to use special data types and functions) is to convert character encoding to UTF-8 (a special Unicode transformation encoding) at subsystem boundaries. UTF-8 has the interesting property that it introduces multi-byte character strings while uniquely and unambiguously preserving ASCII characters without any byte-level misinterpretations like those abovementioned. So if one is only parsing subsequences delimited by various ASCII substrings, without any need to do anything more than pass-through non-ASCII characters, using UTF-8 is a good approach because it can eliminate worries about whether any of the byte-level processing is multi-byte safe.

Even when Unicode string data types are used in an application, there can still be character processing issues. These issues may be associated with data validation processing; however, they can appear in other functionality also.

One of the most popular business logic rules built into applications is that user names and passwords must have certain criteria, such as at least one upper case letter, at least one lower case letter, at least one number, and so forth. Sometimes logic like this is implemented in a manner where the code checks if a character is between “a” and “z” or “A” and “Z” or “0” and “9”.

In many cases, this code can be converted to use methods like isDigit, isUpperCase, isLowerCase, etc., to avoid issues in locales with additional characters in their alphabet, locales that don’t use the ASCII alphabet at all, and locales that use alternate representations to Western-style digits. In

other cases, such as regular expressions, other solutions could be more appropriate.

Some code attempts to parse numeric or currency values and assumes that characters like "- , . and $" are used for the purposes that they are in the United States. This type of code also generally expects these characters to be in the same position within a valid numeric or currency value as they appear in the United States. These situations can be addressed in some system or other i18n-enabled libraries that have functionality to return appropriate values for separator, currency, and negative value strings and their positioning information. However, this is typically more complicated than simply passing the complete string entered by a user to a method that validates and converts the value to a native data type such as Decimal.Parse, NumberFormat.getCurrencyInstance().Parse, Double(string), and so forth.

In some unusual circumstances Unicode string data is converted to another data type such as Byte and then processing occurs on the converted data. If the processing is ambivalent to the contents of the data, the code may work fine internationally. However, if the processing parses the contents or makes assumptions about the content size and structure, errors can occur. One potential problem could be reading or writing Byte data to a file. Some development tools perform implicit conversions between Unicode and native character encoding when reading and writing files. This conversion may not occur for Byte data and application code, or the consumers of the files may not recognize the encoding being used.

In conclusion, application code should use Unicode string data types and be careful of conversions to and from other data types. Application code should also use built-in parsing and validation functionality as much as possible to avoid issues when assumptions based on the locale of the developers are not held globally.

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