Hints and Tips for Java on z/OS – Applications and Environments

Topics:

1. ASCII to EBCIDIC
2. JAR File Considerations
3. Language Environment (LE)

Topic 1 - ASCII to EBCDIC

One important aspect of the System z environment that can sometimes raise portability issue with Java code is that z/OS uses the EBCDIC character encoding instead of the more common ASCII. Within the scope of the JVM, all character and string data is stored and manipulated in Unicode, and I/O data outside of the virtual machine (disk, network, and so forth) is converted to the native platform encoding. However, Java applications that implicitly assume ASCII in specific situations might require some alterations to run as expected under z/OS.

There are a number of environmental tricks that are often useful to test whether implicit ASCII assumptions are in effect. For instance, for config files read in during start-up (or other 'adminstrative' file manipulation), you can use a simple command-line utility to convert those files to the appropriate encoding that you require.

In some cases, changing JVM properties can also allow successful functionality tests, but because this can produce other side effects, the long term solution is sometimes best addressed in code. The Java language contains the abstractions necessary to handle the switch between character encodings. Most important are the various Reader and Writer classes in the java.io package, which provide alternate constructors with a specified codepage. This mechanism is used for internationalization support, and it can also be used to force ASCII (or other) I/O where required. It is important to consider that all I/O does not need to be overridden; for example, character output to the display should remain in the native encoding.

In addition to the Reader and Writer classes, there are a few specific situations that might require additional care. For example, there is an overloaded getBytes() method in the String class that takes an encoding as an additional parameter. This is useful for direct string manipulation when implementing custom data streams or network protocols directly in Java.

Some Java applications explicitly assume ASCII encoding and therefore require some alterations to run as expected under OS/390. For example, a platform neutral application might have hard coded dependencies, such as literals in ASCII.

In general, straightforward workarounds are available for character encoding problems. Some encoding problems are not visible to the application because they are handled within products running on OS/390. An example of this is Java Database Connectivity (JDBC).

It is often best to deal with codepage issues by running the JVM with default encoding of ASCII (-Dfile.encoding=ISO8859-1) and then to specifically call out the codepage that you need when you are writing to MVS files and datasets that are in EBCDIC. This makes porting of code easier, which is a reason Websphere now runs this way.

For your information, the JZOS function now incorporated into z/OS Java SDK5 and SDK6 products may also help in ASCII-EBCDIC questions. The JZOS ZUtil class has a method "getDefaultPlatformEncoding()" that returns the EBCDIC codepage that is being used by the process running the JVM. Also, the FileFactory class can be used to read and write text files - if the file is a //DATASET name then JZOS uses this encoding under the covers automatically.

Topic 2 - JAR file considerations

A jar is an archive of files used by a java applet. Many or all of the files that an applet needs can be combined into a single jar file, which an appletviewer can download in a single http request.

We expect that jar files downloaded from other platforms will contain text data in ASCII (8859-1), and therefore have enabled the appletviewer to expect text files in a jar file to be in ASCII. This means however, that if the jar tool is used on z/OS to create a jar file, then text files must be converted to ASCII before being added. Of course this also means that any jar files you create on z/OS can be transported and used on other platforms. Equally, any text files extracted from a jar file will not be converted from their original encoding.

Topic 3 - Language Environment (LE)

The JVM uses LE for runtime language services. More information can be found at the LE website.