

# Selecting COBOL for Java and Web

---

## CONTENTS

EXECUTIVE OVERVIEW.. 1

ENTERPRISE JAVABEANS  
..... 2

COBOL AND THE BROWSER3  
QUESTIONS TO CONSIDER.. 5

## Executive Overview

The interplay between Java and COBOL for web applications is exciting and can work together to deliver Java capabilities for COBOL applications. Two focus areas are server side applications most notably Enterprise JavaBeans or Servlets and the client such as the browser applet plug-in.

## Enterprise JavaBeans

An Enterprise JavaBean (EJB) is a server-side component, which is very much like a graphical component on a tool palette. The EJB is encapsulated functionality used on a server. The EJB exposes the business functionality to the external clients and other EJB components. These execution objects are then combined or configured within an EJB container. The EJB container then can manage multiple instances of multiple EJB components simultaneously within one or more Application Server(s) on one or more platforms.

Several considerations must be considered as you determine which of the COBOL alternatives can be used in the creation of Enterprise JavaBeans. The Java 2 Enterprise Edition (J2EE) and Application Server environment define execution requirements and capabilities that to be followed in order to have a well behaved application. J2EE restrictions focus on the technical requirements that can impact performance, stability, security, and scalability. EJB containers are designed to verify enforce requirements, and, while not all EJB Containers “verify and report” on non-compliant Enterprise JavaBeans, it is critical that any new or existing EJB adhere to environmental requirements.

The use of native code is forbidden and a very important consideration while moving the COBOL application into the J2EE environment. The EJB specification forbids access or use of native code<sup>1</sup> and states: “*The enterprise bean must not attempt to load a native library*”<sup>2</sup>. The implication for the COBOL application is that these applications cannot use the Java Native Interface (JNI) to access a native COBOL process. All I/O is delegated to the EJB container precluding any direct file access which is analogous to the CICS sub-system managing memory or I/O.

Native libraries access to be reserved for the EJB Container not the Enterprise JavaBean. Any usage of native code within an Enterprise JavaBean component always loads a native library and the native code can circumvent Java's security features. Native code can have stack overflows, underflows, bad pointers, etc. which can disruptively modify the Java State, voluntarily or involuntarily. Fed bad information, native code can be hijacked into supporting viruses or other information.

If automatic verification of EJB compliance is the rule rather than the exception, these applications will cease to function. The component which looks like an Enterprise JavaBean, but which does not follow the specification may have an adverse effect on other EJBs that have a built-in dependency. Like any program relying upon undocumented features, it will not necessarily function in any particular environment, present or future.

Loading native code prevents the Enterprise JavaBean from being used in EJB containers that may support Hot Deploy. Hot deploy is a function where the EJB Component may be deployed into an already running container without rebooting the container. In Java, a ClassLoader is responsible for loading the executable classes; there is a system ClassLoader and then there are custom ClassLoaders. Only the system ClassLoader (for classes known when the Java Virtual Machine starts) may load native functions. Custom ClassLoaders, such as those used for Hot Deploy, will not load native functionality.

An Enterprise JavaBean implemented using native functionality cannot be loaded into a public EJB container. An EJB container provided by an outside service such as an ISP would not allow native code to be executed in their container, because of the possible security issues and restricting Hot Deploy. Any public EJB container will need to be restarted to deploy, and reliability of the public container and be at risk with the use of native components.

---

<sup>1</sup> Mastering Enterprise JavaBeans™ and the Java™ Platform Enterprise Edition published by John Wiley & Sons, Inc. New York, NY, chapter 11, pp. 333-334

<sup>2</sup> Enterprise JavaBean™ v1.1, Final Release, published by Sun Microsystems Inc., November 24, 1999, page 274. accessed at <http://java.sun.com/products/ejb/docs.html>

The native EJB component can abnormally terminate and cause the host EJB Container environment to terminate as well. EJB Components are implemented as threads within only a single process. Using a single process, Java security prevents malicious or accidental tampering of data. Meaning data owned by one component is not exposed to another component. A native component could access another client's private data or other sensitive information. No reputable public EJB container would ever knowingly accept for deployment an EJB component using native functionality.

Enterprise JavaBeans implemented in native code would not be subject to full container control. EJB containers are allowed to implement failure recovery and load balancing by shifting components from one platform to another. An EJB container can legitimately shift an EJB component automatically from running on a Sun platform to an IBM platform to a HP platform as each machine is brought down for maintenance or when a particular platform is overloaded. A process would fail with a native code JavaBean implementation in the defined environment; the EJB container could not move native code safely to an alternate machine. The scalability and fault-tolerance of the EJB specification would also be lost.

Loading native code for an EJB Component is inherently non-portable. While this should be obvious, it should also be noted. The portability advantage and cost savings in moving EJB Components from one platform to another would be lost by using a native compiler.

The Enterprise JavaBean framework is an extremely powerful tool for distributed objects. However, this is a new tool for the COBOL programmer. Both the COBOL programmer and manager should be aware of these facts when selecting a COBOL solution for creating Enterprise JavaBeans.

Only LegacyJ PERCobol allows Enterprise JavaBeans to be written in COBOL with no native code. Providing an all Java, thread- and session-safe code base to the Enterprise JavaBean container, all PERCobol programs that follow the Enterprise JavaBean guidelines will be usable both now and in the future.

### ***COBOL and the Browser***

The other factor of the COBOL web equation is the browser -- notably the plug-in or applet. For COBOL to execute on the browser there must be some way of allowing the COBOL code to be downloaded from the web page and then execute safely within the browser.

Downloading the COBOL executable to the browser is the simple part. Conceptually, this is no more difficult than clicking on any download link, saving to a directory, and then running the result. The trouble with this approach, however, is the 'safely' or 'system security'; the program is a full-fledged application. This full-fledged application may read files on the client, search for information, send that information back to some third-party location, and the user may not even be aware of this occurring.

The issue of web plug-ins can be addressed in several ways: *One approach is to ignore security and hope that there are never any problems.* This is the approach taken by data format plug-ins and may be appropriate because pure data formats have no possibility for execution. (A plain text file viewed with nothing special is not going to be a security breach.) If used for executable content, capable of reading and transmitting sensitive information, this approach should certainly not be used.

*A second approach is to sign the code.* That is, digitally state the code is created and distributed by a certain entity. This approach is useful. Knowing that Company XYZ produced the code, and knowing that Company XYZ is a trusted entity, one which can be contact regarding bad code, is helpful. This doesn't prevent bad actions from occurring, but it does provide a level of responsibility. Nevertheless, it also makes one nervous when running something signed by FormatYourHardDrive. It also involves the additional step of giving the approval for the program to be run and it gives no real information about the content of the code to be executed. Java can code running as an applet or within the Java plug-in can use the signature approach; this is useful

for running as a full application automatically. ActiveX and other native code components use this approach.

*A third approach is the sandbox approach.* That is, the executable code can freely do certain things that are within the sandbox, but the code cannot play outside the sandbox without permission. This approach gives information about the actual activities of the code rather than a blind-faith approach to its content. Sandbox code is so well trusted that it generally runs in the browser without requiring permission to run first. A Java applet embedded in a page will start automatically without prompting the user for a go-ahead first. It won't be sending any private information back to the server without first asking. The Java applet sandbox has the eyes of Sun, IBM, and numerous others examining it constantly for any possible security flaws that would enable the code to harm the client. No other technology can make this claim. No other technology has automatically verifiable code that can run unmodified on such a large number of platforms.

The Java applet can take advantage of both the second and third method of browser client security. Java technology is the only technology that can.

The only COBOL that takes full advantage of browser client security is LegacyJ PERCobol. As a PERCobol applet is a Java applet, it is automatically verifiable, it runs within the Java sandbox as an applet, it can be signed to assign responsibility, and it runs as either a traditional applet or within the Java plug-in. A PERCobol applet cannot read a client's private information and send it to a third-party server without the permission of the user. The only COBOL program safe enough to run in any browser is a PERCobol program. PERCobol takes advantage of the security provided by Sun Microsystems, IBM, HP and others to allow the safest execution environment.

**Questions to Consider**

<b>Accessibility</b>	<ol style="list-style-type: none"> <li>1. Does the solution provide access to my COBOL business process?</li> <li>2. Will the solution require “glue” or external middleware to access existing processes?</li> </ol>	
<b>Reliability</b>	<ol style="list-style-type: none"> <li>1. Does the solution fit within a reliable supported infrastructure?</li> <li>2. Will the solution use established technology or does it require “home grown” extensions to fully utilize an advertised function.</li> </ol>	
<b>Extendibility</b>	<ol style="list-style-type: none"> <li>1. Does the solution allow for easy extensions using an industry accepted model?</li> <li>2. If Java, is it a standard Java capability or if a browser, does the plug-in adhere to browser guidelines?</li> </ol>	
<b>Maintainability</b>	<ol style="list-style-type: none"> <li>1. Does the solution fit within a defined coding and deployment process?</li> <li>2. Are the maintenance procedures consistent with like processes for deploying updates or enhancements?</li> </ol>	
<b>Security</b>	<ol style="list-style-type: none"> <li>1. Does the solution maintain or reduce the level of security established for the execution environment?</li> <li>2. Does the solution introduce a backdoor or other mechanism where security can be breached?</li> </ol>	
<b>Portability</b>	<ol style="list-style-type: none"> <li>1. Does the solution operate on multiple platforms?</li> <li>2. Will the solution be deployable in a public environment, such as a public EJB servers or public web page?</li> </ol>	