Exhibit J

JAVAWORLD

NEWS

To jar or not to jar?

Get the lowdown on using Java Archive (jar) files -- including pros and cons

By Todd A. Webb

JavaWorld |

JUL 1, 1998 12:00 AM PST

With browsers that support Java 1.1.x gaining in market share, more developers will be exploring the use of Java Archive (jar) files. Before you open a jar of worms, you should know some of the *gotchas* involved in using these files. These *gotchas* can affect most aspects of your project -- from how you write your code, to the service your end-users get.

Jar files are an excellent tool to help overcome some of the hurdles that Java faces, such as packaging software and making a program *trusted*. They also have drawbacks -- including potentially longer download times, the need to put in extra work to retrieve resources, and a lack of universal support. You need to know the pros and cons and what you want to accomplish with your Java program before you decide whether or not to use jar files.

The history of jar

The Java language makes it easy for the developer to pull together a great many resources and objects for building software. A Java developer can end up with a heavily populated directory structure that often must be made available over the Internet. Anyone familiar with the HTTP protocol knows that a separate HTTP request must be made for every file. This small overhead becomes a big performance issue as the number of files that must be downloaded increases.

From the beginning, a mechanism was needed to simplify deployment of these classes and improve performance. Sun settled first on the zip file format as defined by PKWARE, and the core Java classes are still distributed in zip format. Additionally there was a need for small, modular software components (now known as JavaBeans) that could be packaged into a single file and imported into an integrated development environment (IDE) -- such as Symantec's Visual Café, IBM's Visual Age, or JBuilder by Inprise (the company formerly known as Borland).

These IDEs needed a little more than just a bundle of resources in a zip file, however. They also needed to know more about the classes -- for example, which classes were beans, and which provided support. It was decided that a *manifest* file would be used for this information, and that the name of the zip file should reflect the availability of the manifest file. A zip file containing the file /meta-inf/manifest.mf was used and dubbed the *Java Archive* file or jar -- the standard distribution format for JavaBeans.

Jar files also offered a solution to other vexing problems. Java was built on the philosophy that it's better to build an overly secure application and relax security as needed than it is to build a low-security system and try to patch it on demand. The *sandbox* model used by browsers is very restrictive, and doesn't allow developers to do some things that could be very useful, even very low risk things, such as reading and writing to a single file on the client machine. A mechanism was needed to allow certain code to perform these operations, so the idea of *trusted applets* was adopted. Since it would be extremely cumbersome to try to mark every class file as trusted, the logical choice was to wrap them all into one file and mark that one file as trusted. That file has the *sig* extension -- the digitally signed version of jar.

When and how to use jar

To decide whether or not jar is for you, consider the following:

- Your target market
- Security
- Performance

- Separate packaging

Target market

The first step in deciding whether to use the jar format is to know your target market. Jar already enjoys universal support from IDEs that deal with JavaBeans. However, jar files are not universally supported by Web browsers. Jar files were introduced with the 1.1 version of Java. A developer should assume that any version of a browser that doesn't run a 1.1 Java virtual machine (JVM) will not understand jar files. A significant number of Internet users still use these older browsers. Many software products also have integrated browsers based on older software -- such as versions of America Online (AOL) and PointCast that use Microsoft's Internet Explorer 3.0. The browsers that currently support jar -- and its digitally signed counterpart, the sig file -- include the latest versions of Netscape's Navigator 4.0x and Sun's HotJava browser 1.1. Even Microsoft appears to fully support jar in its latest incarnation of Internet Explorer 4.0x, even though it has created a proprietary *cab* file format that serves the same purpose. If you know you need to have backward compatibility with older browsers, you will have to forego using jar files. Sorry.

Security

The next consideration is security. If you're deploying an applet, your code will be restricted to the *sandbox* model. If you absolutely must have access to the client's file system, you'll have to run your applet as trusted. This requires you to apply a digital signature to a jar file. Your choice has been made.

Performance

The next important consideration is performance. Packaging your software in a jar file can either speed up performance or slow it down. Take, for example, Sun's popular `Swing` classes, a subset of the Java Foundation Classes (JFC), which are packaged in swingall.jar. Version 1.01 contains 1,305 files compressed to 3,657 kilobytes. Suppose you have an applet that uses swingall.jar -- even if you use only one class from swingall.jar, the entire jar file will be downloaded to the user before that one class will be extracted and loaded. In other cases, with small numbers of

classes, it may take more time to get the file, uncompress it, and extract the contents than to simply fetch each file individually. As a general rule, if you have a large number of files and you can keep them in tight packages (just the files you need at runtime) you are better off using jar files for better performance.

Separate packaging

Next, you need to decide if all of your classes should be packaged together. You'll probably want to make some of your code available to the world, release some of it only to a certain group, and restrict the rest of it to the administrator's use. So package your classes separately. You may find that some packages need to be in jar files and others don't.

Consider all of these things when you're deciding on jar usage. Syzygy Technologies Inc., for example, is using jar files to develop a breakthrough computerized time and attendance system. In this new product, a server provides access to an employee database using Java database connectivity (JDBC). The client software allows employees to securely log into the system across the Internet and submit their hours using a Web browser. The client interface requires a large number of files, but Syzygy wanted to keep the start-up time to a minimum. In addition, the company wanted the option of running the client as a trusted applet in a future version without having to change the architecture. Syzygy also used some visual JavaBeans in this product. So, what did Syzygy decide to do?

It decided to make the client classes accessible through a Web server, but not through the server classes. It would have to package pieces of the software separately, and make a decision on how to package each piece.

To run as a trusted applet in the future, the client classes would have to be digitally signed. This meant they would eventually have to be packaged in a signed jar file, a transition that would be greatly eased if the classes were packaged that way from the beginning. Add to that the fact that the software relied on some key technologies in Java 1.1x, meaning backward compatibility had already been given up, and the decision to package the client classes in a compressed jar file was clear.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.