

**Chapter  
2**

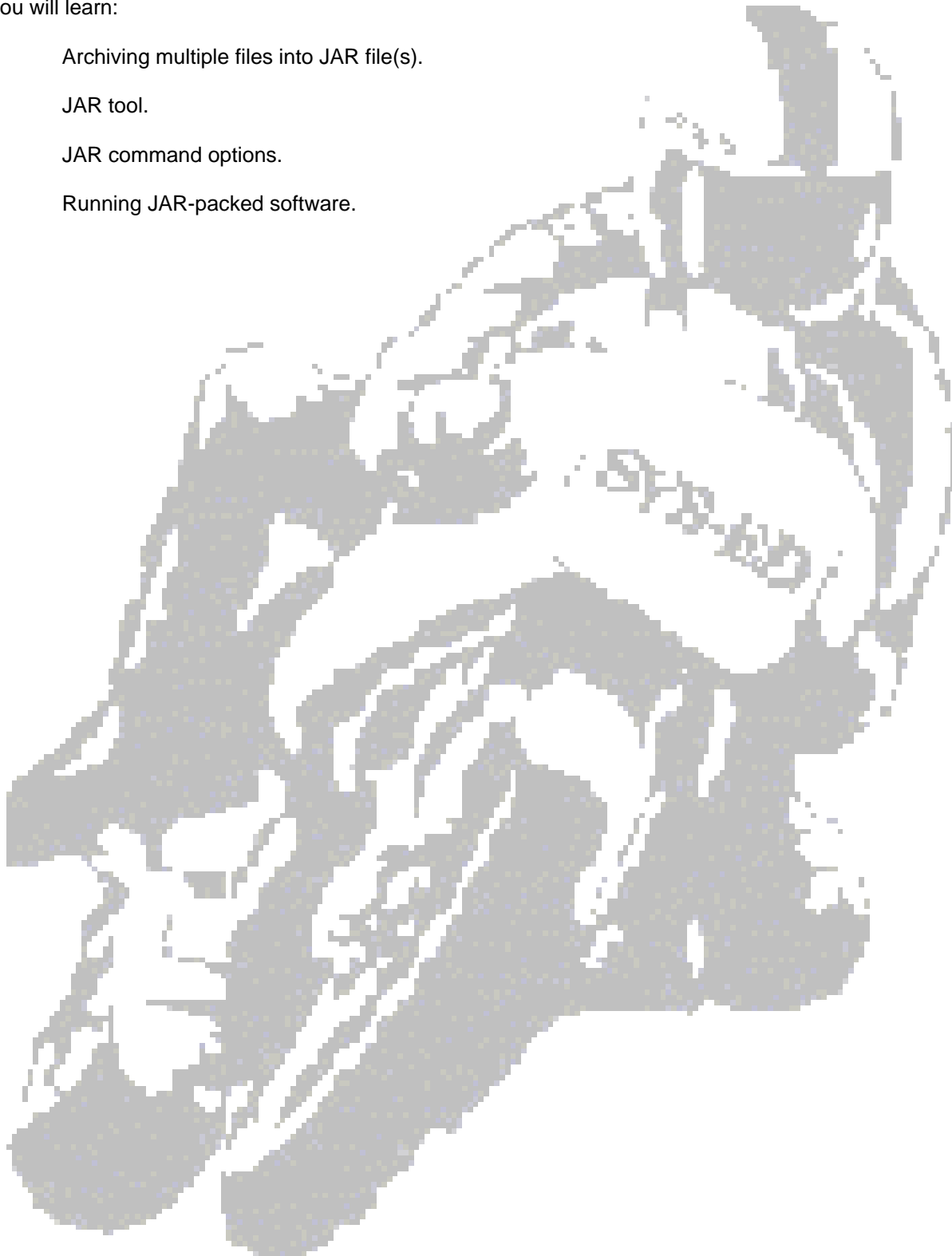
**JAVA ARCHIVE  
FILE FORMAT**

**SYS-ED/  
COMPUTER  
EDUCATION  
TECHNIQUES, INC.**

**Objectives**

You will learn:

- C Archiving multiple files into JAR file(s).
- C JAR tool.
- C JAR command options.
- C Running JAR-packed software.



---

## 1 JAR Files

The Java Archive (JAR) file format enables multiple files to be bundled into a single archive file.

Typically, a JAR file will contain the class files and auxiliary resources associated with applets and applications.

The JAR file format provides the following:

<b>Benefit</b>	<b>Description</b>
Security	You can digitally sign the contents of a JAR file. Users who recognize your signature can then optionally grant your software security privileges it wouldn't otherwise have.
Decreased Download Time	If your applet is bundled in a JAR file, the applet's class files and associated resources can be downloaded to a browser in a single HTTP transaction without opening a new connection for each file.
Compression	The JAR format provides for file compression.
Package Sealing	Packages stored in JAR files can be sealed in order to enforce version consistency. Sealing a package within a JAR file means that all classes defined in that package must be found in the same JAR file.
Package Versioning	A JAR file can hold data about the files it contains, such as vendor and version information.

## 2 JAR Files: The Basics

JAR files are packaged with the ZIP file format. They can be used for "ZIP-like" tasks such as lossless data compression, archiving, decompression, and archive unpacking.

To take advantage of advanced functionality provided by the JAR file format such as electronic signing, you'll first need to become familiar with the fundamental operations.

To perform basic tasks with JAR files, the Java Archive Tool is provided as part of the Java Development Kit. The Java Archive tool is invoked by using the `jar` command, aka the "Jar tool".

The common JAR-file operations are:

Operation	Command
To create a JAR file.	<code>jar cf jar-file input-file(s)</code>
To view the contents of a JAR file.	<code>jar tf jar-file</code>
To extract the contents of a JAR file.	<code>jar xf jar-file</code>
To extract specific files from a JAR file.	<code>jar xf jar-file archived-file(s)</code>
To run an application packaged as a JAR file (version 1.1)	<code>jre -cp app.jar MainClass</code>
To run an application packaged as a JAR file (version 1.2 -- requires Main-Class manifest header)	<code>java -jar app.jar</code>
To invoke an applet packaged as a JAR file.	<code>&lt;applet code=AppletClassName.class archive="JarFileName.jar" width=width height=height&gt; &lt;/applet&gt;</code>

The basic format of the command for creating a JAR file is:

```
jar cf jar-file input-file(s)
```

Options and Arguments	Description
c option	Indicates that a JAR file is to be created.
f option	Indicates that output will be sent to a file rather than to stdout.
jar-file	Is the name that the resulting JAR file is to have. Any filename for a JAR file can be used. By convention, JAR filenames are given a .jar extension, though this is not required.
input-file(s)	Is a space-separated list of one or more files that will be placed in a JAR file. The input-file(s) argument can contain the wildcard * symbol.  If any of the "input-files" are directories, the contents of those directories are added to the JAR archive recursively.

The c and f options can appear in either order, but there must not be any space between them. This command will generate a compressed JAR file and place it in the current directory. The command will also generate a default manifest file for the JAR archive.

These additional options can be added to the cf options of the basic command:

Option	Description
v	Produces verbose output on stderr (in version 1.1) or stdout (in version 1.2) while the JAR file is being built. The verbose output tells you the name of each file as it's added to the JAR file.
0 (zero)	Indicates that you don't want the JAR file to be compressed.
M	Indicates that the default manifest file should not be produced.
m	Used to include manifest information from an existing manifest file. The format for using this option is: jar cmf existing-manifest jar-file input-file(s).
-C	To change directories during execution of the command. Version 1.2 only.

### 3 Viewing the Contents of a JAR File

The basic format of the command for viewing the contents of a JAR file is:

```
jar tf jar-file
```

Options and Arguments	Description
t	Indicates that you want to view the table of contents of the JAR file.
f	Indicates that the JAR file whose contents are to be viewed is specified on the command line. Without the f option, the Jar tool would expect a filename on stdin.
jar-file	Is the filename (or path and filename) of the JAR file whose contents you want to view.

The t and f options can appear in either order, but there must not be any space between them.

This command will display the JAR file's table of contents to stdout.

The verbose option can be added optionally, v, to produce additional information about file sizes and last-modified dates in the output.

#### JAR Example:

The Jar tool is used for listing the contents of the TicTacToe.jar file.

```
jar tf TicTacToe.jar
```

This command displays the contents of the JAR file to stdout:

```
META-INF/MANIFEST.MF
TicTacToe.class
audio/
audio/beep.au
audio/ding.au
audio/return.au
audio/yahoo1.au
audio/yahoo2.au
images/
images/cross.gif
images/not.gif
```

The JAR file contains the TicTacToe class file and the audio and images directory, as expected.

The output also shows that the JAR file contains a manifest file, META-INF/MANIFEST.MF, which was automatically placed in the archive by the JAR tool.

All pathnames are displayed with forward slashes, regardless of the platform or operating system you're using. For example, paths in JAR files are always relative; there will never be a path beginning with C.

The JAR tool will display additional information when the v option is used:

```
jar tvf TicTacToe.jar
```

---

## 4 Extracting the Contents of a JAR File

The basic command for extracting the contents of a JAR file is:

```
jar xf jar-file [archived-file(s)]
```

Options and Arguments	Description
x	Indicates that you want to extract files from the JAR archive.
f	Indicates that the JAR file from which files are to be extracted is specified on the command line, rather than through stdin.
jar-file argument	Is the filename (or path and filename) of the JAR file from which to extract files.
archived-file(s)	Is an optional argument consisting of a space-separated list of the files to be extracted from the archive. If this argument is not present, the Jar tool will extract all the files in the archive.

As usual, the order in which the x and f options appear in the command doesn't matter, but there must not be a space between them.

### Caveat:

When it extracts files, the Jar tool will overwrite any existing files having the same pathname as the extracted files.

---

## 5 Modifying a Manifest File

Version 1.2 of the Jar tool provides a `u` option which can be used for updating the contents of an existing JAR file, including its manifest.

The Jar tool automatically puts a default manifest with pathname `META-INF/MANIFEST.MF` into any JAR file you create. Special JAR file functionality can be enabled, such as package sealing, by modifying the default manifest. Typically, this involves adding special-purpose headers to the manifest that allow the JAR file to perform a particular desired function.

The Jar tool's `m` option allows information to be added to the default manifest during creation of a JAR file. First a text file containing the information must be prepared and then added to the default manifest. The Jar tool's `m` option is added to the information in a file to the default manifest.

The basic command has this format:

```
jar cmf manifest-addition jar-file input-file(s)
```

Options and Arguments	Description
<code>c</code>	Indicates that you want to create a JAR file.
<code>m</code>	Indicates that you want to merge information from an existing manifest file into the manifest file of the JAR file you're creating.
<code>f</code>	Indicates that you want the output to go to a file (the JAR file you're creating) rather than to stdout.
<code>manifest-addition</code> (or path and name)	Is the name of the existing text file whose contents you want included in the JAR file's manifest.
<code>jar-file</code>	Is the name that you want the resulting JAR file to have.
<code>input-file(s)</code>	Is a space-separated list of one or more files that you want to be placed in your JAR file.

The `c`, `m`, and `f` options can appear in any order, but there must not be any whitespace between them.

**Example:**

In version 1.2 of the Java platform, packages within JAR files can be optionally sealed, which means that all classes defined in that package must be archived in the same JAR file.

It might be beneficial to seal a package, for example, to ensure version consistency among the classes in your software.

A package can be sealed by adding the Sealed header beneath the header naming the package that's to be sealed.

```
1      Name: myCompany/myPackage/  
2      Sealed: true
```

The default manifest created by the Jar tool does not contain any Sealed headers, of course, because packages are not sealed by default.

To seal a package, you therefore have to add the Sealed header yourself.

To insert the Sealed header in a JAR file's manifest, you first need to write a text file containing the appropriate headers.

The file you write doesn't have to be a complete manifest file; it can contain just enough information for the Jar tool to know where and what information to merge into the default manifest.

**Example:**

Consider a scenario where the JAR file is to contain these four packages:

```
1      myCompany/firstPackage/  
2      myCompany/secondPackage/  
3      myCompany/thirdPackage/  
4      myCompany/fourthPackage/
```

To seal firstPackage and thirdPackage, you would write a text file with contents that look like this:

```
1      Name: myCompany/firstPackage/  
2      Sealed: true  
  
1      Name: myCompany/thirdPackage/  
2      Sealed: true
```

## 6 Updating a JAR File

The Jar tool in version 1.2 of the Java Development Kit provides a `u` option which you can use to update the contents of an existing JAR file by modifying its manifest or by adding files.

The basic command for adding files has this format:

```
jar uf jar-file input-file(s)
```

Option	Description
<code>u</code>	Indicates that you want to update an existing JAR file.
<code>f</code>	Indicates that the JAR file to update is specified on the command line. If the <code>f</code> option is not present, the Jar tool will expect a JAR filename on stdin.
<code>jar-file</code>	The existing JAR file that's to be updated.
<code>input-file(s)</code>	Is a space-delimited list of one or more files that you want to add to the Jar file.

Any files already in the archive having the same pathname as a file being added will be overwritten. As when creating a new JAR file, you can optionally use the `-C` option to indicate a change of directory.

The `u` option can be combined with the `m` option to update an existing JAR file's manifest:

```
jar umf manifest jar-file
```

Option	Description
<code>m</code>	Indicates that you want to update the JAR file's manifest.
<code>manifest</code>	Is the manifest whose contents are to be merged into the manifest of the existing JAR file.

---

## 7 Running JAR-Packaged Software

Consider these three scenarios:

- C JAR file contains an applet that is to be run inside a browser.
- C JAR file contains an application that is to be invoked from the command line.
- C JAR file contains code that you want to use as an extension.

---

### 7.1 Applets Packaged in JAR Files

Use the APPLET tag to invoke any applet from an HTML file for running inside a browser.

If the applet is bundled as a JAR file, the only thing you need to do differently is to use the ARCHIVE parameter to specify the relative path to the JAR file.

The APPLET tag in the HTML file that calls the demo looks like this:

```
1 <applet code=TicTacToe.class
2     width=120 height=120>
3 </applet>
```

If TicTacToe was packaged in a JAR file named TicTacToe.jar, the APPLET tag could be modified with the simple addition of an ARCHIVE parameter:

```
1 <applet code=TicTacToe.class
2     archive="TicTacToe.jar"
3     width=120 height=120>
4 </applet>
```

The ARCHIVE parameter specifies the relative path to the JAR file that contains TicTacToe.class.

## 7.2 JAR Files as Applications - 1.2 Platform only

In version 1.2 of the JDK software, JAR-packaged applications can be run with the Java interpreter.

The basic command is:

```
java -jar jar-file
```

Option	Description
-jar flag	Tells the interpreter that the application is packaged in the JAR file format.
-jar option	Is not available for interpreters prior to version 1.2 of the Java Development Kit.

Before this command will work, however, the runtime environment needs to know which class within the JAR file is the application's entry point.

To indicate which class is the application's entry point, a Main-Class header must be added to the JAR file's manifest.

The header takes the form:

```
Main-Class: classname
```

The header's value, classname, is the name of the class that's the application's entry point.

To create a JAR file having a manifest with the appropriate Main-Class header, the Jar tool's m flag must be used as described in the Modifying a Manifest section. First a text file needs to be prepared consisting of a single line with the Main-Class header and value.

For example, if an application was the single-class HelloWorld application, the entry point would be the HelloWorld class, and the text file would have this line:

```
Main-Class: HelloWorld
```

Assuming that the text file was in a file called mainClass, it could be merged into a JAR file's manifest with a command such as this:

```
jar cmf mainClass app.jar HelloWorld.class
```

With a JAR file prepared in this way, the HelloWorld application can be run from the command line:

```
java -jar app.jar
```