Pro Jakarta Commons

HARSHAD OAK

Apress™

ISBN (pbk): 1-59059-283-2

Printed and bound in the United States of America 10987654321

Technical Reviewers: Henri Yandell, John Zukowski

Editorial Board: Steve Anglin, Dan Appleman, Gary Cornell, James Cox, Tony Davis, John Franklin, Chris Mills, Steve Rycroft, Dominic Shakeshaft, Julian Skinner, Jim Sumser, Karen Watterson, Gavin Wray, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Tracy Brown Collins

Copy Manager: Nicole LeClerc

Copy Editor: Kim Wimpsett

Production Manager: Kari Brooks

Production Editor: Janet Vail

Compositor: Kinetic Publishing Services, LLC

Indexer: Valerie Perry

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States: phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com. Outside the United States: fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section. You will need to answer questions pertaining to this book in order to successfully download the code.

About the Author

Harshad Oak has a master's degree in computer management from Symbiosis, India, and is a Sun Certified Java Programmer and a Sun Certified Web Component Developer. He is the founder of Rightrix Solutions (`http://www.rightrix.com/`), which is primarily involved in software development and content management services. Harshad has been part of several J2EE projects at i-flex Solutions and Cognizant Technology Solutions. Furthermore, he has written several articles about Java/J2EE for CNET Builder.com (`http://www.builder.com/`). He is also a guest lecturer on Java and J2EE.

Harshad wrote the book *Oracle JDeveloper 10g: Empowering J2EE Development* (Apress, 2004) and coauthored *Java 2 Enterprise Edition 1.4 Bible* (Wiley & Sons, 2003).

Harshad is passionate about all kinds of writing and has published articles about a wide array of subjects. He is an avid nature lover and enjoys reading nonfiction books. He, however, hopes to soon try his hand at writing fiction and providing some serious competition to the Lords of the Rings and the Harry Potters of the world.

Feel free to send all comments and suggestions to `harshad@rightrix.com`.

CHAPTER 1

Introducing Jakarta Commons: The Uncommon Commons

It is likely that you are reading this because you want to do more with less and you believe in working smarter rather than harder. Working smart is the guiding principle for this book; every topic it covers is a smart solution to a specific need. Each topic will make you more effective and efficient, saving you a lot of time and money.

Deciding to adopt new technologies and techniques today boils down to economics. No longer can technology be marketed just on the basis of being the next big thing that is capable of doing fascinating tasks for you. If you want a new technology to be used and to prosper, it has to make economic sense. Adopting Apache Jakarta Commons, apart from the fun involved in using it, makes a lot of economic sense. All software that is part of Jakarta Commons is free and open source.

The Jakarta Commons project Web site is at `http://jakarta.apache.org/commons/`. From this page, you can easily find links to all the other Commons projects.

The Jakarta Commons project consists of easily reusable components that you can quickly put to good use in any type of Java development. The components are not big applications but sleek little bits of code that do a definite task very well.

The tasks that most of these components undertake are of a general nature, something that many applications might require. Strangely enough, few efforts such as the Commons exist where code is written once and then one piece of well-tested code is reused. I dread to think of how many times the same piece of code to implement something such as connection pooling or to perform simple date formatting operations has been written in the Java world.

It is uncommon to see an effort being made to find a common bit of functionality and to reuse it. Often, even within the same company, people are reluctant to find the code they need and reuse that code. The causes are many; lethargy for finding and understanding somebody else's code and a lack of faith in that code top the list. Another problem is that developers often prefer to copy, paste, and adapt code from an existing application rather than reuse that same piece of code.

The key to understanding and using Commons components is knowing when and how to use a particular method or component. As such, the components do not require much understanding of theory or other core fundamentals. In this book, although I delve into the theory behind the need for a particular component and the way it is designed, the focus is on using the components. The book is meant to serve as a ready reference to the components, enabling you to find solutions quickly.

Examining the Apache Factor

Open-source software is becoming mainstream rapidly; however, not everybody is comfortable with the idea of using open-source code. Big businesses do not mind blowing money on software as long as they have someone to hold responsible for it. I have been part of projects in which the client explicitly asked that no open-source software be used apart from anything that comes from Apache.

The Apache name has a big role to play in open-source software. Considering the ubiquitous nature of the Apache HTTP server, Apache is perhaps the biggest name in open source. Apache code has come to be used widely and trusted by everyone in the industry. Therefore, the Apache Jakarta Commons project benefits greatly from the Apache name. The wide acceptance of the Commons components has as much to do with the Apache name as with the quality of the components. Jakarta Commons is a small piece in the overall Jakarta and Apache structure and has a limited and focused scope of operation.

Understanding the Scope of Commons

The charter of any Jakarta project is a good place to learn more about that project. The charter states what the project is meant to achieve and what the scope of the project will be. The Commons charter states, "The subproject shall create and maintain packages written in the Java language, intended for use in server-related development, and designed to be used independently of any larger product or framework."

This conveys what the Commons project is about. The key point is that the components are to be designed in such a way that they exist independently of any larger framework. So most, if not all, components are very small downloads that do not exceed 1 megabyte (MB). Although the charter does say that the components are intended for server-related development, most components are not restricted to server-side development and are quite useful in all kinds of Java development. The Commons charter does not as such put any restrictions

on the kind of components that can be part of Commons. So the current set of components cover various functionalities ranging from quickly processing Extensible Markup Language (XML) to database connection pooling.

Getting an Overview of the Components

The Commons project already has more than three dozen components; some have been developed from scratch, and others have been pulled out of bigger projects under Jakarta.

The components that are part of Jakarta Commons are categorized into two parts:

Commons Proper: This section holds components that are in a production-ready state. Experimentation and changes to interfaces will be minimal and systematic with these components. So components in the Commons Proper can be used on projects.

Sandbox: The Sandbox components are those that hold a lot of promise but are still in an incomplete stage. Many of the Sandbox components are still on the drawing board or in beta and alpha releases. Lots of tweaking and experimentation is likely to happen to these components, so using them on live projects is not advisable. Sandbox components, when ready, either can move into the Commons Proper or can be absorbed by some other project.

Software that is still in beta and alpha versions generally does not sit well with most project decision makers. A final release is the least that is expected of open-source projects to even be considered for adoption.

The following are the various components currently featured in the Commons Proper:

BeanUtils: JavaBeans adhering to standard naming conventions as specified in the JavaBeans specification and having private member variables with a public `get` method to fetch the value of the variable and a `set` method to set it are often used in Java development. The BeanUtils component greatly simplifies working with these JavaBeans and getting things done while having to write minimal code.

Betwixt: The Betwixt component provides a way to easily convert JavaBeans into XML.

CLI: This component makes working with command line arguments simple and systematic. It provides a methodology to specify arguments, their usage, and access to arguments.

Codec: Codec provides implementations for common encoders and decoders.

Collections: The Collections component goes beyond what is offered by the Java Collections Framework and provides many classes that offer useful collections and utilities. Many collections build on existing core Java Application Programming Interface (API) collections, and others are new creations.

DBCP: The Database Connection Pool (DBCP) component provides the functionality of pooling database connection and optimally utilizing them. Database connections are precious resources that need to be created and used wisely.

DbUtils: The DbUtils component is a handy set of classes capable of easily handling most of the routine tasks that need to be performed while accessing a database using Java Database Connectivity (JDBC).

Digester: Creating, parsing, and using XML files can be quite a pain if only the basic APIs are used. Digester makes working with XML quite simple and is of great use, especially while retrieving information from configuration files.

Discovery: The Discovery component is meant to ease discovery of implementations for various interfaces.

EL: An important addition to Java Server Page (JSP) 2.0 is the support for an expression language. The EL component is an expression language interpreter.

FileUpload: The FileUpload component can provide file upload capability to Web applications.

HttpClient: The HttpClient component goes beyond Hypertext Transfer Protocol (HTTP) features offered by the Net component and provides many features required by a Java client communicating using HTTP.

Jelly: Jelly is a component that can convert XML into executable code. Jelly has many subelements that use the core concept to achieve different ends.

Jexl: Jexl stands for *Java expression language*. The Jexl component provides an expression language engine that implements an extended version of the expression language of JSP Standard Tag Libraries (JSTL).

JXPath: JXPath is an interpreter for XPath. XPath uses simple path expressions to identify nodes in an XML document. You have most likely used XPath to refer to nodes in an XML document while creating an XSL Transformations (XSLT) document. JXPath, however, applies XPath expressions not just to XML but even to objects such as JavaBeans and maps.

Lang: Lang is a component relevant to almost any Java project. The Lang component is like an extension of the core Java API `java.lang` package. The component provides many utility methods to perform basic operations on strings, numbers, and so on.

Latka: Latka is a functional testing tool.

Logging: The Logging component is a simple wrapper over the various logging techniques in use such as Log4J, Java Software Development Kit (JSDK) logging, or just `System.outs`. Using this component is advisable to prevent code being locked into a certain methodology. The logging component can make switching between the various logging techniques a simple task.

Math: Math has been recently promoted to Commons Proper and provides mathematics and statistics features that are not available in `java.lang` or in Commons Lang.

Modeler: The Modeler component is meant to simplify the creation of Model management beans as defined by the Java Management Extensions (JMX).

Net: The Net component is a Java library meant to simplify communication using various protocols such as Telnet, Post Office Protocol 3 (POP3), File Transfer Protocol (FTP), Simple Mail Transfer Protocol (SMTP), and so on.

Pool: Pool is an object pooling API that can also be customized for specific pooling requirements.

Primitives: This component provides collections that can be used with Java primitives. So although you could never have created a `java.util.List` that was a collection of `int` primitives, such things are possible using the primitives component.

Validator: The Validator is a simple framework that provides for defining various validation rules in an XML file. The framework can then enable reuse and consistent application of these rules.

By the time you are reading this, there might be other components that have moved from the Sandbox to the Commons Proper. However, even in the Commons Proper, some components have truly matured and are already being used in many applications while others are not that popular. In the next section, you will look at the kind of Java development that is an ideal candidate for Commons usage.

Where Do You Use Commons?

Commons is useful to any brand of Java development in which you might be involved. It might be Web development using Servlets and JSP or Java Graphical User Interface (GUI) development using Swing. Commons can add a lot of value to your Java development and, most important, make your life easier and your development time shorter. That does not mean that all Commons components are useful to all kinds of applications. Commons components are primarily meant for server-side Java development, and using them in other kinds of development is more incidental.

The nature of the Commons project demands that the components do something specific for a certain need, so most components do just that: They focus on achieving something specific for a certain kind of development. Although using a database connection-pooling component for a Java Swing application that has nothing to do with databases does not make sense, it does make sense for a Web application that is using the database extensively.

So an important part of Commons adoption is that you first pick out components that can offer something useful to the development you have undertaken and then focus all efforts on those components. For instance, I have hardly ever gone beyond using three or four of the components at a time.

Before moving into the technicalities of the Commons packages and what each package does, you should know that the groundwork that has to be done to use all the components is quite similar. All the Commons packages that can boast of a proper release have a binary download available at the official Jakarta binary download site at `http://jakarta.apache.org/site/binindex.cgi`. A binary download is a compressed ZIP or TAR file that does not include any of the source code; as a result, the download file tends to be small.

TIP *I recommend you also download the source code for the Commons components that you intend to use. In most cases, this involves basic Java code that is quite easily understandable.*

The binary file in most cases will have a name in the format `commons-<ComponentName>-<VersionNumber>.ZIP`. If you have downloaded a TAR version instead of ZIP, the file extension will be `.TAR`.

This file will contain a Java Archive (JAR) file that holds all the class files for that component. The Javadocs also ship as part of the binary download. Not all components provide documentation beyond what is present in the Javadocs. However, the Javadocs are quite extensive and explain each method pretty well. So the best way to understand what a method does is to look at the Javadocs. If

that does not suffice, you can always look at the code that actually does the job for you—don't forget you are using open-source software!

To use any of the Commons components, all you now need to do is place the JAR file in a suitable location so that the class loader can easily find the classes in the JAR to be used in your application.

Developing a Web Application

Servlets and JSP-based Web applications have to adhere to a specific directory structure as warranted by the specification. As per the specifications, all Web applications have a directory named `WEB-INF` that holds the files that are meant to be private and not directly accessible by any clients accessing the application. Within the `WEB-INF` directory, there are two special directories named `classes` and `lib`. The `classes` directory is where compiled class files are kept during development when these files undergo frequent changes. Any Java 2 Enterprise Edition (J2EE)–compliant server can automatically detect any changes to the classes in this directory and load the newer class file. The `classes` directory is where, in all probability, you will keep the Java code that is using one or more of the Commons components.

The directory that is more relevant in the context of using Commons is the `lib` directory. All JAR files containing classes being used by the application are kept in the `lib` directory. The `lib` directory normally holds third-party JAR files, such as JDBC drivers or frameworks such as Struts. The Commons JAR files should also be placed into the `lib` directory, and once done, all other classes in the Web application can use all the classes in the JAR files.

Developing Other Applications

For non-Web applications, you do not have the easy option of using the special features of the `WEB-INF/lib` directory. You now need to specify the JAR file in the `CLASSPATH`.

If you are using an Integrated Development Environment (IDE) such as NetBeans, Eclipse, or JDeveloper, you have the option of setting the JAR file location into the `CLASSPATH` for a particular project or application. You then do not have to edit the `CLASSPATH` value at a system level. If your IDE of choice provides a feature of *clubbing* many JAR files into a single unit, use that to create a new library of all the JAR files for the various Commons components. I did just that when I began working on this book so that once that library was created, I could associate it with any project I created. Because all components are in distinct packages and some components also have dependencies on other Commons components, this is the best option to get things up and running in no time.

TIP *I am a great believer in using all tools that can make my life easier and make me work as little as possible. So if you are not using any Java IDE yet, choose any one and get going. The learning curve is not at all as steep as you might expect. All IDEs are pretty simple to use once you get the hang of them.*

Why Should You Use Commons?

Many of the features of the core Java API that you use regularly are things for which you could write code. You of course can create new data structures similar to those in the Java Collections API, or you can write something as simple as the `trim` method of the `String` class. So why don't you write a `trim` method specific to your application?

The answer is obvious enough: It makes more sense to use the `trim` method provided by the `String` class. No book is complete without quoting this phrase at least once, so I will go ahead and say it: "Why reinvent the wheel?"

You might be wondering what Java API usage has got to do with you using the Commons APIs. The answer is simple: Commons is nothing but Java on steroids. No, those using steroids are not thrown out of the Java Olympics, but instead they are the top contenders to get the gold.

NOTE *Do not expect Commons component methods to have complex logic or calculations. In most cases, the logic involved is pretty simple. However, even 50 lines of simple but well-tested code can be a big asset during development.*

You will now see a simple example to illustrate why you should choose the Commons approach over writing new code. The scenario is that you have to split some text you have received based on a certain separator such as a space or a # symbol and then store the split results in a `String` array. You would have to use the `StringTokenizer` class, handle null values, and then store the split contents into an array.

This is not a very difficult case, but it would require about 10 lines of code to be written and, more important, tested and maintained. Would it not be better if you had to write just one line of code and not have to worry about testing the logic or maintaining it?

The Java class is as follows:

```
package com.commonsbook.chap1;
import org.apache.commons.lang.StringUtils;
```

```
public class SplitString {
    public static void main(String[] args) {
        //Split a String into an Array using # as separator.
        String [] splitArr=StringUtils.split("AB#CD#EF#GH", "#");

        for(int i=0; i<splitArr.length; i++ ) {
            System.out.println( i + ") "+ splitArr[i]);
        }
    }
}
```

The only line that provides some action is the line where you call the static `split` method of the `StringUtils` class. That is all it will take if you use the `StringUtils` class that is part of the Commons Lang component and that provides many methods to make fiddling with strings a lot easier.

The output for this example is as follows:

```
0) AB
1) CD
2) EF
3) GH
```

Moving Beyond Jakarta Commons

A few other common components projects have been recently developed that are similar to Jakarta Commons. For example, the Apache DB project has a Commons section at `http://db.apache.org/commons/`, and the Apache XML project has a Commons section at `http://xml.apache.org/commons/`. The Apache DB Commons project hosts common components that are related to database development, and the XML project has XML-specific common components.

An Apache Commons project distinct from Jakarta Commons has also emerged at `http://commons.apache.org/`. Jakarta Commons is meant solely for Java components, and the Apache Commons project is language neutral. Apache Commons is still in its early stages. So, for the time being at least, Jakarta Commons is the only place to look for common Java components.

Summary

The ideal position to use Commons is when your boss pushes you to deliver a project in an extremely short time. Keep things simple, use as many Commons components as relevant, and then tell your boss you had to work really hard and

build everything from scratch. A promotion might just follow. It is all about working smart; working hard is just being stupid.

In this chapter, you got a quick overview of Jakarta Commons. In the next chapter, you will explore one of the most popular Commons components, the Lang component.