

Module 4 sample

XML parsing with Digester

4.1 The Digester component.....	1
4.2 The Digester stack	5
4.3 A match made in heaven?.....	6
4.4 Conforming to the rules!.....	7
4.5 Resolving conflicts with namespace-aware parsing.....	22
4.6 Rule sets: sharing your rules.....	23
4.7 Externalizing patterns and rules: XMLRules	24
4.8 Summary	26
Index	27

Configuration files are used in all sorts of applications, allowing the application user to modify the details of an application or the way an application starts or behaves. Since storing these configuration details in the runtime code is neither possible nor desirable, these details are stored in external files. These files take a variety of shapes and forms. Some, like the simple name-value pair, let you specify the name of a property followed by the current assigned value. Others, like those based on XML, let you use XML to create hierarchies of configuration data using elements, attributes, and body data.

For application developers, parsing configuration files and modifying the behavior of the application isn't an easy task. Although the details of the configuration file are known in advance (after all, the developers created the basic structure of these files), objects specified within these configuration files may need to be created, set, modified, or deleted. Doing this at runtime is arduous.

The Digester component from Jakarta Commons is used to read XML files and *act* on the information contained within it using a set of predefined rules. Note that we say XML files, *not* XML configuration files. Digester can be used on all sorts of XML files, configuration or not.

The Digester component came about because of the need to parse the Struts configuration file. Like so many of the Jakarta Commons projects, its usability in Struts development made it a clear winner for the parsing of other configuration files as well.

In this module, we'll look at the Digester component. We'll start with the basics of Digester by looking at a simple example. We'll then look at the Digester stack and help you understand how it performs pattern matching. This will allow us to tackle all the rules that are prebuilt into Digester and demonstrate how they're useful. We'll use this knowledge to create a rule of our own. We'll round out the module by looking at how the Digester component can be externalized and made namespace-aware.

4.1 The Digester component

As we said, the Digester component came about because of the need to parse the Struts Configuration file. The code base for the parsing the Struts configuration file was dependent on several other parts of Struts. Realizing the importance of making this code base independent of Struts led to the creation of the Digester component. Struts was modified to reuse the Digester component so as not to include any dependencies.

In essence, Digester is an XML → Java object-mapping package. However, it's much more than that. Unlike other similar packages, it's highly configurable and extensible. It's event-driven in the sense that it lets

you process objects based on events within the XML configuration file. *Events* are equivalent to patterns in the Digester world. Unlike the Simple API for XML (SAX), which is also event-driven, Digester provides a high-level view of the events. This frees you from having to understand SAX events and, instead, lets you concentrate on the processing to be done.

Let's start our exploration of the Digester component with a simple example. Listing 4.1 shows an XML file that we'll use to parse and create objects. This XML file contains bean information for the JavaBeans listed in listings 4.2 and 4.3. Listing 4.4 shows the Digester code required to parse this file and create the JavaBean objects.

Note: To be able to run these and all the other examples in this module, you need to have two supporting libraries in addition to commons-digester in your CLASSPATH. These libraries are common-logging.jar and commons-collections.jar.

Listing 4.1 A simple XML file (book_4.1.xml)

```
<?xml version="1.0"?>
<book title="Jakarta Commons in Action">
  <author id="1001">
    <name>Vikram Goyal</name>
  </author>
</book>
```

Listing 4.2 Book JavaBean

```
package com.manning.common.chapter04;
import java.util.Vector;

public class Book {

    private String title;
    private Vector authors; ← List of book's authors held in this Vector

    public String getTitle() { return this.title; }
    public void setTitle(String title) { this.title = title; }

    public Vector getAuthors() { return this.authors; }
    public void setAuthors(Vector authors) { this.authors = authors; }

    public void addAuthor(Author author) {
        if(authors == null) authors = new Vector();
        authors.add(author);
    }

    public String toString() { ← Book bean returns toString representation
        StringBuffer buffer = that includes title and all authors
            new StringBuffer("Title: " + getTitle() + "\r\n");

        if(authors != null) {
            for(int i = 0; i < authors.size(); i++) {
                buffer.append(
                    "Author " + (i + 1) + ": " + authors.get(i));
            }
        }
    }
}
```