

SAMPLE CHAPTER



Tuscany SCA

IN ACTION

Simon Laws
Mark Combellack
Raymond Feng
Haleh Mahbod
Simon Nash

 MANNING



Tuscany SCA in Action

by Simon Laws, Mark Combellack, Raymond Feng,
Haleh Mahbod and Simon Nash

Chapter 8

Copyright 2011 Manning Publications

brief contents

PART 1 UNDERSTANDING TUSCANY AND SCA..... 1

- 1 ■ Introducing Tuscany and SCA 3
- 2 ■ Using SCA components 33
- 3 ■ SCA composite applications 71

PART 2 USING TUSCANY..... 107

- 4 ■ Service interaction patterns 109
- 5 ■ Implementing components using the Java language 132
- 6 ■ Implementing components using other technologies 175
- 7 ■ Connecting components using bindings 197
- 8 ■ Web clients and Web 2.0 232
- 9 ■ Data representation and transformation 257
- 10 ■ Defining and applying policy 284

PART 3	DEPLOYING TUSCANY APPLICATIONS	309
11	▪ Running and embedding Tuscany	311
12	▪ A complete SCA application	329
PART 4	EXPLORING THE TUSCANY RUNTIME	355
13	▪ Tuscany runtime architecture	357
14	▪ Extending Tuscany	382

Web clients and Web 2.0

This chapter covers

- Using SCA with servlets and JSPs
- Creating interactive web interfaces using SCA and JavaScript
- Integrating Atom and RSS feeds into SCA applications

In today's computing world, everything seems to be web-enabled and available over the internet, for example, banking, food shopping, business applications, your utility bills, and so on.

There are many ways to make applications web-enabled. These include the more traditional approaches of using a web-enabling framework, for example, the Java Platform, Enterprise Edition, and the less-traditional approaches encompassed by the term *Web 2.0*, such as Ajax, Atom, and RSS feeds, and the like.

In this chapter, we'll look at some of the ways that SCA and Tuscany can help to simplify web-enabling your applications. First, we'll look at how SCA can be used with Java Enterprise Edition by looking at Java Servlets (or servlets for short) and JavaServer Pages (JSPs). The advantage of this approach is that organizations that have already invested time in creating Java Enterprise Edition applications can reuse their existing code in new SCA applications.

We'll then move on to look at how you can use HTML and JavaScript to create interactive browser-based clients that make use of SCA services. The advantage of this approach is that interactive clients can be produced without requiring a full-featured Java Enterprise Edition server because it will run with a much lighter-weight web server. Not all elements within a web application need to be interactive, and some can be static, for example, static HTML pages, images, and logos. We'll also look at how Tuscany can help with this type of static content.

Finally, we'll look at Atom and RSS feeds. We'll show how to produce Atom and RSS feeds using Tuscany. People can subscribe to these feeds and be notified of updates. We'll then look at how to consume Atom and RSS feeds using Tuscany.

There's a lot of information to cover in this chapter, so let's get started and look at using servlets with SCA.

8.1 Servlets as SCA component implementations

In this section, we'll look at how you can create an interactive web application using SCA and servlets. Servlets are part of the Java Enterprise Edition specification and allow developers to create dynamic web content using the Java programming language. We'll examine how a servlet can be used as a component implementation and how it can invoke existing SCA services.

By using a servlet as a component implementation, a developer can create applications that combine existing servlets or Java Enterprise Edition code with new or existing SCA services. This hybrid approach maximizes reuse of existing code and allows SCA to be adopted incrementally.

8.1.1 Creating the currency converter user interface using a servlet

Suppose that as part of TuscanySCATours, you wanted to provide your customers with a web interface that they can use to calculate the conversion value of currencies, for example, convert \$100 U.S. into sterling.

The TuscanySCATours application already contains an SCA currency converter contribution that allows values in one currency to be converted to another. All you'll need to do is create a web interface that uses it. In this section, you'll create a servlet that will use the currency converter contribution to provide a web interface that can be used to convert currencies, as shown in figure 8.1.

The currency converter servlet runs within a Java Enterprise Edition container and uses the currency converter SCA component to convert currencies. A web browser can be used to invoke the currency converter servlet over HTTP.

USING TUSCANY IN A SERVLET

A servlet can look up and use an SCA service. How this is achieved depends on the level of support the servlet container provides for SCA. If SCA is fully supported, then the servlet code can use the `@Reference` annotation as it would in any other SCA code. But few servlet containers today contain this level of support for SCA. For these servlet

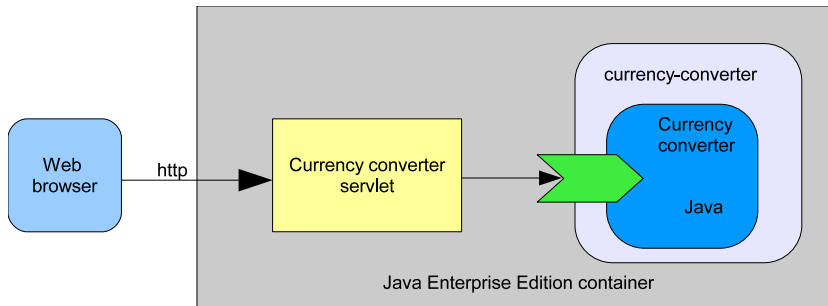


Figure 8.1 Overview of a currency converter servlet connected to the currency converter SCA component

containers, the `@Reference` annotation won't work, and the references need to be set up in the `init()` method of the servlet. Currently, support for SCA is available in Apache Geronimo (using a plug-in) and in IBM WebSphere.

The following listing shows the `CurrencyConverterServlet.java` class from `contributions/currency-servlet` that makes use of SCA to invoke the currency converter.

Listing 8.1 Currency converter servlet invoking SCA service

```
public class CurrencyConverterServlet extends HttpServlet {

    @Reference
    protected CurrencyConverter currencyConverter;

    public void init(ServletConfig config) {
        if (currencyConverter == null) {
            ComponentContext context =
                (ComponentContext)config.getServletContext()
                    .getAttribute("org.osoa.sca.ComponentContext");
            currencyConverter = context.getService(
                CurrencyConverter.class, "currencyConverter");
        }
    }

    protected void service(HttpServletRequest request,
        HttpServletResponse response) throws IOException {
        Writer out = response.getWriter();
        out.write("<html><body><h2>SCA Tours Currency Converter Servlet</h2>");
        out.write("Welcome to the SCA Tours Currency Converter Servlet<p>");
        out.write("<form method=post action="
            + "\"CurrencyConverterServlet\">");
        out.write("Enter value in US Dollars");
        out.write("<input type=text name=dollars size=15><p>");
        out.write("<input type=submit>");
        out.write("</form><p>");

        String dollarsStr = request.getParameter("dollars");
        if (dollarsStr != null) {
```

1 Container-supported reference injection

2 Containers not supporting reference injection

3 Currency input form

```

double dollars = Double.parseDouble(dollarsStr);
double converted = currencyConverter.convert(
    "USD", "GBP", dollars);
out.write(dollars + " US Dollars = "
    + converted + " GB Pounds");
}

out.write("</body></html>");
out.flush();
out.close();
}
}

```

4 Convert currency using SCA service

The servlet uses the `@Reference` annotation **1** to look up the currency converter service. This style of reference injection will work on servlet containers that support SCA. But because most don't at this time, the servlet has code **2** to look up the references using the SCA component context. The SCA `ComponentContext` is provided as a standard part of the SCA API. The servlet contains a form **3** that's used to enter the amount in U.S. dollars to convert to sterling. The final block of code checks whether an amount in U.S. dollars has been entered, and if so, it converts it into sterling using the currency converter service **4**.

CONFIGURING THE SCA CURRENCY CONVERTER SERVICE

Our servlet looks up the currency converter SCA service, but we haven't yet configured it. Let's do that now by creating a `META-INF/sca-deployables/web.composite` file that contains the composite definition. Tuscany will automatically scan the `META-INF/sca-deployables` directory for composites. This `web.composite` file can be found in the `contributions/currency-servlet` directory of the TuscanySCATours application and is shown here.

Listing 8.2 Currency converter servlet web.composite file

```

<composite xmlns="http://www.oxa.org/xmlns/sca/1.0"
  xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.0"
  targetNamespace="http://scatours.com"
  name="CurrencyConverterServlet">

  <component name="WebClient">
    <implementation.web web-uri="" />
    <reference name="currencyConverter"
      target="CurrencyConverter" />
  </component>

  <component name="CurrencyConverter">
    <implementation.java class=
      "com.tuscanyscatours.currencyconverter.impl.
      CurrencyConverterImpl" />
    <service name="CurrencyConverter" />
  </component>
</composite>

```

1 WebClient currency converter reference

2 Currency converter component

In this composite file the `WebClient` component is defined so that it has a reference to the currency converter service **1**. We set `web-uri` to `""` to indicate that the whole of

the WAR is an SCA contribution. The composite file also defines the currency converter component that exposes the currency converter service ②.

CONFIGURING THE SERVLET FILTER TO ROUTE REQUESTS TO TUSCANY

The final file that we'll need to define for the servlet is the WEB-INF/web.xml file that configures a servlet filter that will direct requests via Tuscany. It can be found in the contributions/currency-servlet directory of the TuscanySCATours application and is shown in the following listing.

Listing 8.3 Servlet filter to direct requests via Tuscany

```
<web-app version="2.4"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" >

  <display-name>SCA Tours Currency Converter Servlet</display-name>

  <filter>
    <filter-name>tuscany</filter-name>
    <filter-class>
      org.apache.tuscany.sca.host.webapp.
        ↪ TuscanyServletFilter
    </filter-class>
  </filter>

  <filter-mapping>
    <filter-name>tuscany</filter-name>
    <url-pattern>*</url-pattern>
  </filter-mapping>

  <servlet>
    <servlet-name>CurrencyConverterServlet</servlet-name>
    <servlet-class>com.tuscanyscatours.currencyconverter.
      ↪ servlet.CurrencyConverterServlet</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>CurrencyConverterServlet</servlet-name>
    <url-pattern>/CurrencyConverterServlet</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>CurrencyConverterServlet</welcome-file>
  </welcome-file-list>
</web-app>
```

① Tuscany servlet filter

② Currency converter servlet

③ URL mapping

④ Default web page

The Tuscany servlet filter ① is used to start and stop Tuscany when the web application starts and stops. This ensures that the `CurrencyConverterServlet` ② can make use of SCA services. The `servlet-mapping` ③ is used to define the URI to access the servlet. Finally, you'll configure the `CurrencyConverterServlet` to be displayed by default ④.

At this point, you've defined all the files that you'll need:

- META-INF/sca-deployables/web.composite
- WEB-INF/web.xml

Let's move on to deploying and running the currency converter servlet.

DEPLOYING AND RUNNING THE CURRENCY CONVERTER SERVLET

Before you can run the currency converter servlet, you'll need to create the WAR file and deploy it on a web server that can host servlets. In our sample we'll use the Maven build to produce the WAR file. Please refer to your web server's documentation for information on how to deploy a WAR file. Once the currency converter WAR file is deployed, you can access the currency converter servlet using a web browser: <http://localhost:8080/scatours-contribution-currency-servlet/>.

You should see the currency converter servlet, as shown in figure 8.2.

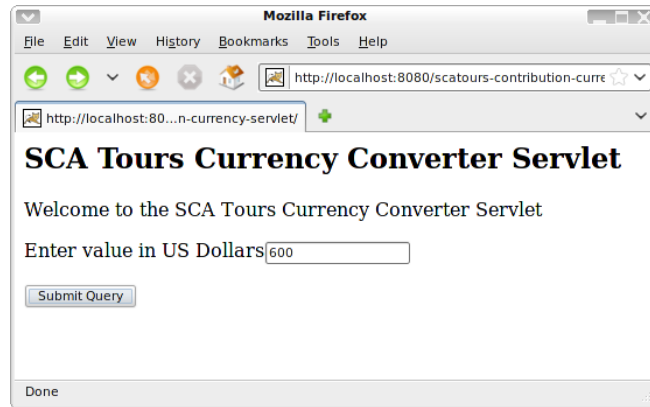


Figure 8.2 Currency converter servlet viewed through a web browser

Entering 600 in the field and clicking the Submit Query button will cause the currency converter servlet to convert \$600 U.S. into sterling using the SCA currency converter service and display the result, as shown in figure 8.3.

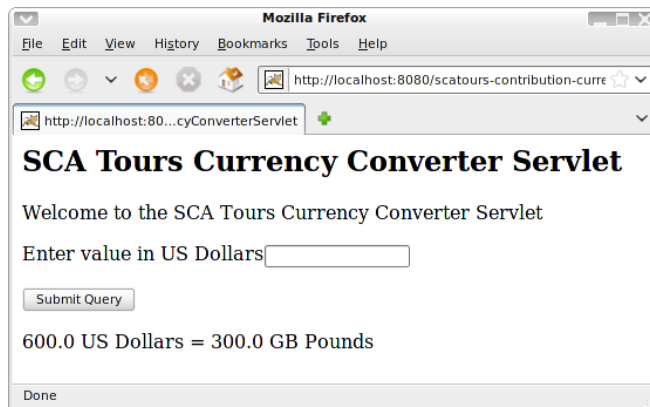


Figure 8.3 Currency converter servlet converted \$600 U.S. to sterling

In this section, you've seen how you can create applications that combine SCA and servlets. With this combination, you'll use the servlet to expose a web service that can be invoked from a web browser. When the servlet is invoked, it can invoke SCA services by using SCA references. In the next section, we'll look at using a JSP instead of a servlet.

8.2 **Writing web component implementations using JSPs**

In this section, we'll show how you can use JavaServer Pages (JSPs) to create web component implementations that make use of SCA. This approach is similar to using servlets as component implementations, but you'll use a JSP instead of a servlet. The advantage is that a JSP tends to be easier to write than a servlet.

One of the core differences between JSPs and servlets is that JSPs attempt to separate the application code from the web page. This doesn't mean that there will be no code in the web page. It means that there'll be a clear separation between the two (unlike with servlets where there's only code). Don't worry if this doesn't make much sense now because it'll become clear once you move on to writing your JSP.

When writing a JSP you can make use of tag libraries (sometimes also called taglibs), which provide libraries of code that can be used with your JSP. The JSP specification defines a standard core tag library for tasks such as loops and variables. To simplify using SCA within your JSPs, SCA defines a tag library. The SCA tag library contains a single `reference` tag that can be used to look up SCA services, as shown by the following code fragment:

```
<sca:reference name="myService" type="myservice.MyService"/>
```

This code fragment will look up the SCA reference called `myService` that's defined by the `myservice.MyService` Java interface.

Enough of the theory, let's move on and show how it all works.

8.2.1 **Exposing the currency converter using a JSP**

In this section, we'll show how the currency converter can be exposed as a web interface using a JSP. It will be based on the example in section 8.1.1, which used a servlet to do the same task. To save duplicating the code, we'll show the differences between using a JSP and a servlet and refer back to the servlet example when the two approaches are the same.

USING TUSCANY IN A JSP

To make use of the SCA taglib, you'll need to include its definition in your JSP. You can then use its tags to look up SCA services. The following listing shows the `currency-converter.jsp` that makes use of the SCA taglib to invoke the currency converter. It can be found in the `contributions/currency-jsp` directory of the `TuscanySCATours` application.

Listing 8.4 Currency converter JSP invoking SCA service

```
<%@ page contentType="text/html; charset=UTF-8"
    language="java" %>
<%@ taglib uri="http://www.oesa.org/sca/sca_jsp.tld"
    prefix="sca" %>
```

1 Use SCA taglib

```

<sca:reference name="currencyConverter"
  type="com.tuscanyscatours.currencyconverter.
  CurrencyConverter"/>

<html>
<body>
<h2>SCA Tours Currency Converter JSP</h2>
Welcome to the SCA Tours Currency Converter JSP<p>

<form method=post action="currency-converter.jsp">
Enter value in US Dollars
<input type=text name=dollars size=15><p>
<input type=submit>
</form><p>

<%
  String dollarsStr = request.getParameter( "dollars" );
  if ( dollarsStr != null ) {
    double dollars = Double.parseDouble(dollarsStr);
    double converted = currencyConverter.convert(
      "USD", "GBP", dollars);
    out.println(dollars + " US Dollars = "
      + converted + " GB Pounds");
  }
%>
</body>
</html>

```

2 Look up currency converter SCA service

3 Currency input form

4 Convert currency using SCA service

The currency converter JSP defines itself as a JSP and includes the SCA taglib ①. It then uses the reference tag from the SCA taglib to look up the currency converter SCA service ②. The JSP contains a form that's used to enter the amount in U.S. dollars to convert to sterling ③. The final block of JSP code checks whether an amount in U.S. dollars has been entered, and if so, it converts it into sterling using the currency converter service ④.

CONFIGURING THE SCA CURRENCY CONVERTER SERVICE

Your JSP looks up the currency converter SCA service but you haven't yet configured it. Let's do that now by creating a META-INF/sca-deployables/web.composite file that contains the composite definition. You can use the same file as you used for the servlet in listing 8.2 in section 8.1.1.

CONFIGURING THE SERVLET FILTER TO ROUTE REQUESTS TO TUSCANY

The final file that you'll need to define for the JSP is a servlet filter that will direct requests via Tuscany. Once again, you can use the same file as you used for the servlet in listing 8.3 in section 8.1.1, but you'll need to make two changes. You'll need to remove the `<servlet>` and `<servlet-mapping>` tags because you aren't using servlets and change the `<welcome-file-list>` to refer to your JSP. It can be found in the contributions/currency-jsp directory of the TuscanySCATours application and is shown here.

Listing 8.5 Fragment of the servlet filter to direct requests via Tuscany

```

<welcome-file-list>
  <welcome-file>currency-converter.jsp</welcome-file>
</welcome-file-list>

```

Now that you've defined all the files that you need, let's move on to deploying and running the currency converter JSP.

DEPLOYING AND RUNNING THE CURRENCY CONVERTER JSP

Before you can run the currency converter JSP, you'll need to create the WAR file and deploy it on a web server that can host JSPs. Please refer to your web server's documentation for information on how to deploy a WAR file. Once the currency converter WAR file is deployed, you can access the currency converter JSP using a web browser:

```
http://localhost:8080/scatours-contribution-currency-jsp/
```

You should see the currency converter JSP, as shown in figure 8.4.

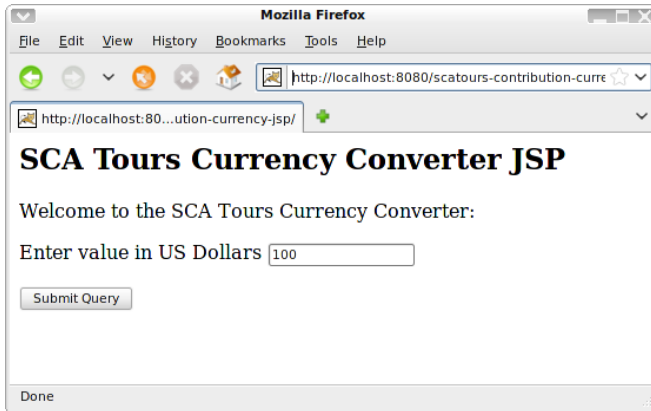


Figure 8.4 Currency converter JSP viewed through a web browser

Entering 100 in the field and pressing the Submit Query button will cause the currency converter JSP to convert 100 U.S. dollars into sterling using the SCA currency converter service and display the result, as shown in figure 8.5.

As you can see, the JSP has used the currency converter SCA service to do the currency conversion.

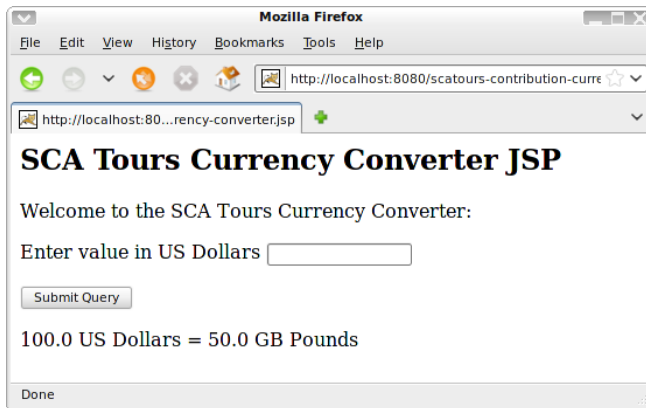


Figure 8.5 Currency converter JSP converted 100 U.S. dollars to sterling.

Let's now move on to looking at how you can use an HTML page as a component implementation.

8.3 HTML pages as SCA component implementations

An alternative approach to using a servlet or JSP to create a web interface is to use an HTML page with JavaScript.

Whereas servlets and JSPs run on the server, HTML and JavaScript run in the browser on the client machine. This has implications for how the Tuscany infrastructure treats these web application artifacts as SCA components.

Regardless of whether you're using servlets, JSPs, or HTML/JavaScript, the artifacts can only be configured with SCA references. This is reasonable because there's no sense that these user interface artifacts provide an SCA service for other computers to use. The service they provide is to the human user.

When an SCA-enabled web container reads the `implementation.web` element, it's fairly easy to appreciate that the server is able to interact with the servlet or JSP and ensure that the proxy that represents the configured reference is injected into the right place. This is less obvious when using HTML/JavaScript on the client. How does the Tuscany SCA runtime inject reference proxies across the network into the client browser that's running the HTML/JavaScript code?

All will be revealed in this section, where we'll describe how Tuscany supports invoking SCA services that have a JSON-RPC SCA binding using JavaScript contained within HTML pages. The user interface for the TuscanySCATours application uses exactly this approach, and we'll spend some time looking into how it works.

What are JSON and JSON-RPC?

JavaScript Object Notation-Remote Procedure Call (JSON-RPC) is a specification that defines how an application can perform remote procedure calls (RPCs) using HTTP. JSON-RPC will convert the RPC into a HTTP request from the client and send it to the server, where the server can process the request and send back a reply. This is done using standard HTTP. The details of the RPC, such as parameters, are encoded by JSON-RPC into the body of the HTTP request using JavaScript Object Notation (JSON).

JSON is a way of converting data into a text format that's lightweight and independent of a particular computer language. This allows data to be exchanged between applications that may be written in different languages. For example, a JavaScript application could use JSON to send data to a Java application.

8.3.1 Using an HTML page for the TuscanySCATours user interface

Tuscany provides the `widget` implementation type, `implementation.widget`, that allows an HTML page to be used as an SCA component implementation.

OVERVIEW OF THE TUSCANYSCATOURS HTML INTERFACE

The TuscanySCATours application makes use of an HTML page for the user interface using `implementation.widget`. Using this interface, users can search for and book their holidays. Listing 8.6 shows part of the composite XML file for the TuscanySCATours application user interface that uses this `implementation.widget` element. The full composite XML file can be found in the `contributions/fullapp-ui` directory of the TuscanySCATours application.

Listing 8.6 TuscanySCATours user interface with `implementation.widget`

```

<composite xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.0"
  name="fullapp-ui" ...>
  <component name="SCAToursUserInterface">
    <tuscany:implementation.widget location="scatours.html"/>
    <service name="Widget">
      <tuscany:binding.http uri="/scatours"/>
    </service>
    <reference name="scaToursCatalog"
      target="SCATours/SCAToursSearch">
      <tuscany:binding.jsonrpc/>
    </reference>
    ...
  </component>

  <component name="SCAToursComponent">
    <implementation.java class=
      "com.tuscanyscatours.impl.SCAToursImpl"/>
    <service name="SCAToursSearch">
      <tuscany:binding.jsonrpc/>
    </service>
    ...
  </component>
</composite>

```

The TuscanySCATours user interface composite defines an `SCAToursUserInterface` component that's implemented using the widget implementation type **1**. The location attribute is used to specify the name of the HTML file, in this case, `scatours.html`. Because this isn't a standard SCA implementation type, you'll need to import the Tuscany XML namespace. The `SCAToursUserInterface` component also defines a service that has an HTTP binding **2** on it. This is so the component can be accessed over HTTP.

The `SCAToursUserInterface` component also has a reference to the `SCAToursSearch` service **3** that uses the JSON-RPC binding. This reference will be used by the JavaScript in the HTML to look up the SCA service. There are several other similar references, but we haven't shown them here to keep the example short. The `SCAToursSearch` service is defined on the `SCATours` component **4** and has a JSON-RPC binding so it can be invoked from JavaScript in the HTML page.

Let's shift our focus to the `scatours.html` file to see how it looks up and invokes SCA services. The HTML in the following listing shows the key parts of the `scatours.html`

file that interacts with the SCAToursSearch service. The full `scatours.html` file can be found in the `contributions/fullapp-ui` directory of the TuscanySCATours application.

Listing 8.7 Looking up reference from `scatours.html` file

```

<html>
<head>
<title>SCA Tours</title>
<link rel="stylesheet" type="text/css" href="style.css" />
<script type="text/javascript" src="scatours.js"></script>

<script language="JavaScript">
  //@Reference
  var scaToursCatalog
    = new tuscanysca.Reference("scaToursCatalog");
  ...

  function searchTravelCatalog() {
    scaToursCatalog.search(getTripLeg(), search_response);
  }

  function search_response(items, exception) {
    if(exception) {
      alert(exception.javaStack);
      return;
    }
    ...
    for (var i=0; i<items.length; i++) {
      if (items[i].type == "Trip") {

        packagedHTML += '<td>' + items[i].name + '</td>';
        packagedHTML += '<td>' + items[i].description + '</td>';
        packagedHTML += '<td>' + items[i].location + '</td>';
        ...
      }
    }
  }
  ...
}

```

1 Include SCA JavaScript

2 Look up SCA reference

3 Invoke SCA service

4 Handle response

The `scatours.html` file is read by the browser using the URL <http://localhost:8080/scatours/scatours.html>. This URL is provided on the server by the HTTP binding associated with the Widget service of the component using `implementation.widget`. In effect, `implementation.widget` is presenting the component implementation to the browser and using the SCA composite file to configure the HTML file on the fly with appropriate JSON-RPC reference proxies.

The HTML includes the `scatours.js` JavaScript file 1 that provides code that can be used to look up SCA references. This JavaScript file is generated automatically by the widget implementation based on the reference configuration of the component using

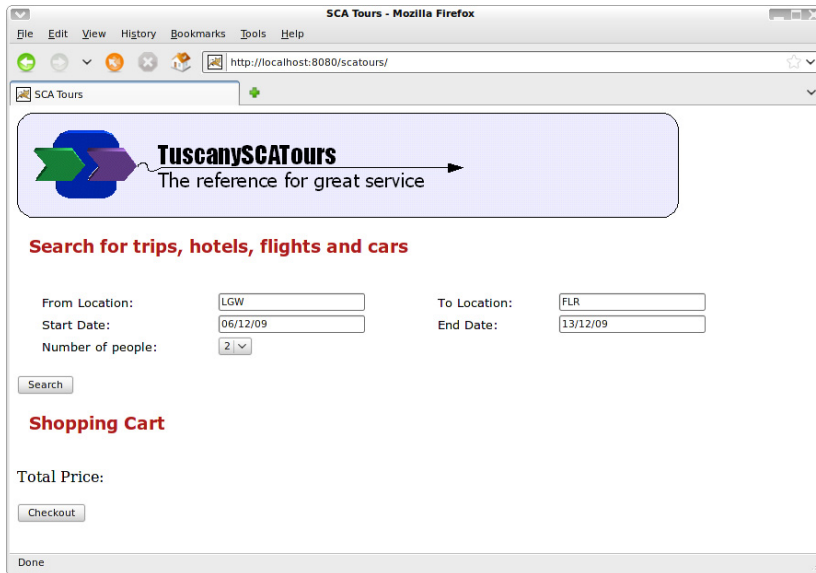


Figure 8.6 The TuscanySCATours HTML page as an SCA component implementation

implementation.widget. The name of the JavaScript file matches the name of the HTML file but with a .js extension rather than .html.

The HTML uses JavaScript to look up an SCA reference **2** marked with `//@Reference`, which is then used to invoke the `search` operation **3**. The response from the SCA service is returned via the `search_response` JavaScript function **4**, which checks to see if there was an exception and, if not, displays the results on the HTML page.

RUNNING THE TUSCANYSCATOURS HTML INTERFACE

Now that you have an idea of what's required to use an HTML page as an SCA component implementation, let's run the travel application user interface to see it in action using the `launchers/fullapp` launcher from the TuscanySCATours application. Once it starts, use your web browser to access it at <http://localhost:8080/scatours/>, and you should see a web page similar to the one shown in figure 8.6.

Clicking the Search button will cause the JavaScript in the HTML page to use the SCA reference to do a search for available flights by invoking the `search` method using JSON-RPC and then display a page of the results.

This concludes our time with the implementation.widget. Let's move on to look at exposing file system resources using Tuscany.

8.4 Exposing file system resources

Tuscany provides the ability to publish web resources using SCA components. The component is implemented using `implementation.resource`, which points at the directory holding the resources. The identified directory is the component implementation.

The HTTP binding, `binding.http`, can then be used to allow HTTP-based browser access to the resources provided by the component implementation. This can be useful when creating a web application using SCA that needs to serve static content such as images.

Here we'll dive straight in and use a set of help pages provided with the travel application to demonstrate how HTML pages on the filesystem resources are exposed to the user's browser via an SCA component.

8.4.1 Exposing the TuscanySCATours help pages

Suppose that, in the TuscanySCATours application, you have some help pages written in HTML. You could deploy a separate web server alongside your SCA application and make the pages available that way. However, that would be no fun because it doesn't use SCA, and this is a book about SCA. The advantage of using SCA to expose the HTML pages is that you can create a single converged application that contains SCA and HTML elements in a single server. If you have a large number of HTML pages, it may be better to use a traditional web server approach, but let's see how it can be done using SCA.

To expose your help pages so that the user can access them using a browser, you'll use the `implementation.resource` component implementation and `binding.http` in the `help-pages.composite` file. The `help-pages.composite` file can be found in the `contributions/help-pages` directory of the TuscanySCATours application and is shown in the following listing.

Listing 8.8 Exposing the help pages using `implementation.resource`

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
  xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.0"
  targetNamespace="http://www.tuscanyscatours.com"
  name="help-pages">

  <component name="Help">
    <tuscany:implementation.resource
      location="help_pages"/>
    <service name="Resource">
      <tuscany:binding.http
        uri="http://localhost:8085/help/" />
    </service>
  </component>
</composite>
```

The Help component uses `implementation.resource` as its implementation type **1** and defines that the resources to be exposed are in the `help_pages` directory. Because `implementation.resource` is a Tuscany implementation type, the Tuscany XML namespace needs to be defined. Finally, the Resource service **2** is defined with an HTTP binding that specifies that the resources should be published over HTTP at <http://localhost:8085/help/>.

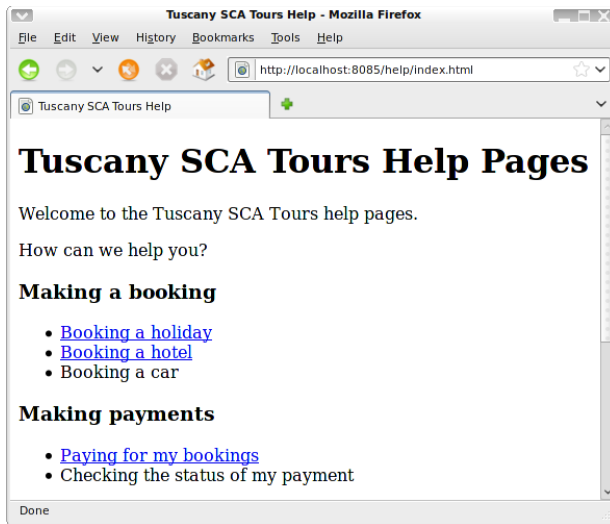


Figure 8.7 Screenshot of the TuscanySCATours help pages

When using `implementation.resource`, a single Resource service must be defined. This is used by the Tuscany implementation code of `implementation.resource` to expose the resources over an SCA service. This is why a Resource service is defined in the XML.

Now that you've defined the help pages composite, let's see what happens when you run it. When the `launchers/help-pages` launcher is run, the help pages will output the following messages to the console:

```
Node started - Press enter to shutdown.
To view the help pages, use your Web browser to view:
  http://localhost:8085/help/index.html
```

If you do as the output suggests and use a web browser to view the help pages at <http://localhost:8085/help/index.html>, then you should see a web page similar to the one shown in figure 8.7.

Clicking any of the links will result in your browser displaying another help web page. This page will have been served by the Tuscany `implementation.resource` component implementation. The contents of the HTML pages exist as files in the `help_pages` directory. If you look there, you'll see several files, including the `index.html` file displayed in figure 8.7.

These days, users expect to see more than static HTML pages when using web-based applications. They expect to be able to access applications in different ways, including being notified of changes using protocols such as Atom and RSS. Tuscany supports these as well, so let's take a look at these next.

8.5 Exposing component services as Atom and RSS feeds

The use of web feeds has become a popular way for users to be notified of content updates to websites. Users would subscribe to the web feed for the site(s) they're interested in, using their feed-reading software. When the website updates its content, it

updates its web feed listing the updated content. A user's feed-reading software would detect this feed update and inform the user of the updated content. From a user's perspective, using a web feed saves them from having to visit the websites regularly to see if any of the content has been updated.

The two main formats of web feeds are Atom and RSS. They're similar in concept in that they both allow content updates to be published containing information such as title, summary text, and publish date. They both use XML to format the information, but they have different schemas and are therefore not compatible with each other. In this section we'll look at how Tuscany and SCA can be used to publish web feeds in both formats.

8.5.1 Exposing the TuscanySCATours blog as an Atom feed

Suppose that TuscanySCATours wants to set up a corporate blog so that it can inform its customers of the latest news and special offers. Let's see how you can use SCA techniques to expose Atom and RSS feeds using Apache Tuscany.

Apache Tuscany provides support for exposing Atom feeds by adding the Atom binding to a service of a Java component implementation that exposes a feed. We'll examine the implementation code later. First, let's look at the composite file for the TuscanySCATours blog Atom feed. You can find it in the contributions/blog-feed directory of the TuscanySCATours application and in the following listing.

Listing 8.9 TuscanySCATours blog composite exposing an Atom feed

```
<composite xmlns="http://www.oxia.org/xmlns/sca/1.0"
  xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.0"
  targetNamespace="http://tuscanyscatours.com/"
  name="blogFeed">

  <service name="BlogAtom" promote="BlogFeed">
    <tuscany:binding.atom
      uri="http://localhost:8090/BlogAtom"/>
    </service>

  <component name="BlogFeed">
    <implementation.java class="com.tuscanyscatours.
      ↪ blog.feed.impl.GenericBlogFeedImpl"/>
    </component>
  </composite>
```

1 Composite service with Atom binding

2 Blog feed implementation

You'll define a service called BlogAtom for the Blog component **1** and add an Atom binding to it. The Tuscany XML namespace needs to be defined because the Atom binding is a Tuscany binding type. The Atom binding is configured to publish the feed at <http://localhost:8090/BlogAtom> using the `uri` attribute. Finally, you'll define a Java component implementation called BlogFeed that provides the implementation for the feed **2**. You now have a composite file that can be used to expose the TuscanySCATours blog as an Atom feed. One thing we haven't looked at is the BlogFeed component implementation. Let's do that now.

Tuscany supports using the Atom binding on two different types of Java component implementations, namely ones that implement the following:

- `org.apache.tuscany.sca.binding.atom.collection.Collection`
- The `getAll()` method from the Tuscany `Data` API

The advantages and disadvantages of these two approaches are summarized in table 8.1.

Table 8.1 Comparison of the Atom component implementation types

Implements	Advantage	Disadvantages
Collection	You have full control over the Atom feed because you write the code that creates the Atom feed.	You have to write all the Atom feed creation code. Code is tied to Atom feed APIs.
getAll	Code is not tied to the Atom feed APIs. Tuscany creates the Atom feed for you.	You have less control over the Atom feed because Tuscany creates it on your behalf.

The two approaches are somewhat converse to each other. The simplest approach is to implement the `getAll` method and let Tuscany construct the Atom feed code. This has the benefit of increasing the reusability of the implementation code because other bindings can be applied to it later. However, Tuscany will have made some sensible default choices regarding how the feed will be constructed. These defaults may not match your requirements for how you want the Atom feed to look.

A slightly more complex approach is to implement the `Collection` interface because the implementation code will have to create the Atom feed itself using the Atom APIs. This gives you full control of how the Atom feed will look. But the implementation code is less reusable because it's tied to the Atom APIs.

Which approach should you use? It depends on your requirements. Generally, we'd recommend using the `getAll` approach because it's the simplest and most flexible in terms of SCA bindings. But if you know that you'll want to have complete control of how the feed looks, then use the `Collection` approach. For the purposes of this example, we'll use the `getAll` approach, as shown in the following listing. For completeness, there's an example of using the `Collection` approach in class `AtomBlogFeedImpl` in the `contributions/blog-feed` directory of the TuscanySCATours application.

Listing 8.10 `GenericBlogFeedImpl` exposes the TuscanySCATours blog

```
public class GenericBlogFeedImpl extends BaseBlogFeedImpl {
    public Entry<Object, Object>[] getAll() {
        final List<BlogPost> posts = getAllBlogPosts();
        final Entry<Object, Object>[] entries
            = new Entry[posts.size()];
        int i = 0;
        for (BlogPost post : posts) {
```

```

        entries[i++] = convertBlogPostToFeedItem(post);
    }

    return entries;
}

private Entry<Object, Object> convertBlogPostToFeedItem(
    BlogPost post) {
    final Item item = new Item(post.getTitle(),
        post.getContent(), post.getLink(),
        post.getRelated(), post.getUpdated());
    final Entry<Object, Object> entry
        = new Entry<Object, Object>(nextBlogID(), item);
    return entry;
}

```

2 Convert blog posts

3 Convert to feed entry

The `GenericBlogFeedImpl` class implements the `getAll` method from the Tuscany `Data` API. This means that the SCA Atom binding will manage the creation of the Atom feed and the conversion of the Tuscany `Data` items to Atom feed entries. The `getAll` method gets all blog posts using the `getAllBlogPosts` method ①, which returns the list of blog posts in some internal format. The implementation for the `getAllBlogPosts` method isn't shown here because it's implemented by the `BaseBlogFeedImpl` superclass. In this example, it returns a hardcoded list of blog posts. In a real-world application, it would load the blog posts from some persistent storage such as a database or filesystem.

Now that you have all the blog posts, you'll need to convert them to instances of the `Entry` class from the Tuscany `Data` API ② by calling the `convertBlogPostToFeedItem` method ③ on each blog post. The Tuscany `Entry` class provides a feed technology independent implementation of a feed item.

Now that you have your composite file and implementation code for the Tuscany-SCATours blog feed, run it using the `launchers/blog-feed` launcher from the Tuscany-SCATours application and see if you can access the Atom feed. When run, it will output the following messages to the console:

```

Node started - Press enter to shutdown.
To view the blog feed, use your Web browser to view:
    http://localhost:8090/BlogAtom

```

Follow the instructions and see if you can access the Atom feed by accessing the address <http://localhost:8090/BlogAtom> using a web browser. You'll see the web page shown in figure 8.8.

With Mozilla Firefox, you're presented with an option to subscribe to the feed. If you click the blog entry link, you'll be redirected to the *Tuscany SCA in Action* website.

How can you verify that this is indeed an Atom feed? The easiest way is to examine the source for the page (in Mozilla Firefox, right-click View Page Source). If you do, you'll see the Atom XML header, a fragment of which is shown here:

```

<?xml version='1.0' encoding='UTF-8'?>
<feed xmlns="http://www.w3.org/2005/Atom">

```

That confirms it; you've exposed the TuscanySCATours blog as an Atom feed.

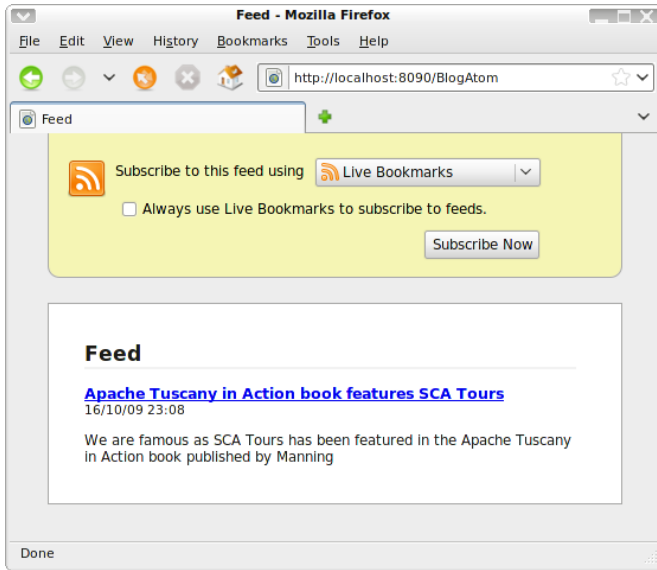


Figure 8.8 Screenshot of the TuscanySCATours blog Atom feed

8.5.2 Extending the TuscanySCATours blog with an RSS feed

Now that you've implemented the Atom feed for the TuscanySCATours blog, you'll add an RSS feed to it. Because you implemented your blog feed code by implementing the `getAll` method and are using Tuscany to convert the entries to feed entries, you'll need to add a new service with a Tuscany RSS binding to the `blog-feed.composite` XML file. It can be found in the `contributions/blog-feed` directory of the TuscanySCATours application and is shown here.

Listing 8.11 TuscanySCATours blog composite exposing an Atom feed and an RSS feed

```
<composite xmlns="http://www.oxa.org/xmlns/sca/1.0"
  xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.0"
  targetNamespace="http://goodvaluetrips.com/"
  name="blogFeed">

  <service name="BlogAtom" promote="BlogFeed">
    <tuscany:binding.atom
      uri="http://localhost:8090/BlogAtom"/>
  </service>
  <service name="BlogRSS" promote="BlogFeed">
    <tuscany:binding.rss
      uri="http://localhost:8090/BlogRSS"/>
  </service>

  <component name="BlogFeed">
    <implementation.java
      class="com.tuscanyscatours.blog.feed.impl.GenericBlogFeedImpl"/>
  </component>
</composite>
```

1 Service with
RSS binding

2 RSS binding

All we needed to do was add another service for the RSS feed ❶ and add the RSS binding to it ❷. Simple as that—no code changes are required! But did you notice that we added another composite service rather than adding the RSS binding to the existing BlogAtom service? The reason for this is to show that it's simple to promote the same service with a different name and binding. We could have just added the RSS binding to the existing BlogAtom service and accepted that the BlogAtom service has both Atom and RSS bindings.

If we had decided to take the other approach and coded the TuscanySCATours blog so that it extended the `org.apache.tuscany.sca.binding.atom.collection.Collection` interface, we'd need to write a new Java component implementation that extended the `org.apache.tuscany.sca.binding.rss.collection.Collection` interface because the two interfaces aren't compatible. For completeness, there's an example of using the `Collections` approach in class `RSSBlogFeedImpl` in the TuscanySCATours application.

Running the TuscanySCATours blog feed again using the `launchers/blog-feed` launcher of the TuscanySCATours application will output the following messages to the console:

```
Node started - Press enter to shutdown.  
To view the blog feed, use your Web browser to view:  
  http://localhost:8090/BlogAtom  
  http://localhost:8090/BlogRSS
```

Follow the instructions and see if you can access the RSS feed by accessing the address <http://localhost:8090/BlogRSS> using a web browser. You'll see the web page shown in figure 8.9.

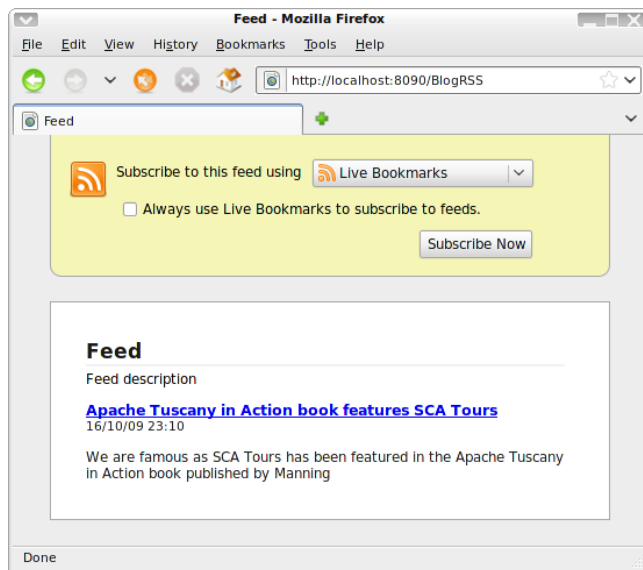


Figure 8.9 Screenshot of the TuscanySCATours blog RSS feed

With Mozilla Firefox, you're presented with an option to subscribe to the feed. If you click the blog entry link, you'll be redirected to the *Tuscany in Action* website.

How can you verify that this is indeed an RSS feed? The easiest way is to examine the source for the page (in Mozilla Firefox, right-click View Page Source). If you do, you'll see the RSS XML header, a fragment of which follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss xmlns:content="http://purl.org/rss/1.0/modules/content/"
```

This shows that you've exposed the TuscanySCATours blog as an RSS feed.

You've seen that it's straightforward to construct a component so that it can provide an Atom feed. It's similarly straightforward to expose a component service as an RSS feed. The Atom and RSS bindings can be used independently but will often be used together to provide access to a component service to the maximum number of users. You'll need to think about this because it has an impact on how you'll construct your component implementation.

Now that you know how to expose component services using Atom or RSS, let's look at how an SCA component can read an Atom or RSS feed.

8.6 *Referencing Atom and RSS feeds*

In the previous section, we looked at how you could expose component services as Atom and RSS feeds. We'll now move on to look at how you can use Atom and RSS feeds in your SCA applications. Fortunately, Tuscany provides support for accessing Atom and RSS feeds using `binding.atom` and `binding.rss`, respectively.

8.6.1 *Logging the TuscanySCATours blog Atom feed*

As you saw earlier, TuscanySCATours now has Atom and RSS feeds for their blog. Being the professional organization that they are, they decide they should keep a log of the Atom and RSS feeds so they can record what feed entries are posted. For this purpose, they decide to write the TuscanySCATours feed logger to log the feeds.

To simplify the running of this example, rather than running your own feeds, you're going to use the Atom and RSS feeds from the mocked-up TuscanySCATours blog that can be found at <http://scatours.wordpress.com>.

The first thing you'll need to access the TuscanySCATours Atom feed is the URL of where it's published: <http://scatours.wordpress.com/?feed=atom>. To make sure that it's working correctly, it's probably worth attempting to access the feed URL in your web browser. Now that you know the URL of the feed, let's have a look at the composite file for the TuscanySCATours feed logger. It can be found in the `feed-logger.composite` file in the `contributions/feed-logger` directory of the TuscanySCATours application and is shown in the following listing.

Listing 8.12 TuscanySCATours feed logger composite accessing an Atom feed

```
<composite xmlns="http://www.osea.org/xmlns/sca/1.0"
  xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.0"
  targetNamespace="http://tuscanyscatours.com/"
  name="feedLogger">
```

```

<service name="FeedLogger" promote="FeedLogger">
</service>

<component name="FeedLogger">
  <implementation java class="com.tuscanyscatours.
    ↳ feedlogger.impl.FeedLoggerImpl"/>
  <reference name="scaToursBlogAtom">
    <tuscany:binding.atom
      uri="http://scatours.wordpress.com/?feed=atom"/>
    </reference>
  </component>
</composite>

```

1 Define FeedLogger service

2 FeedLogger component

3 SCA Tours Atom reference

The XML then defines the FeedLogger service ① that exposes the FeedLogger component. You'll use this later to invoke the TuscanySCATours feed logger. The FeedLogger component is a Java component implementation ② and has a single reference with a `<binding.atom>` element ③ that uses the Atom binding to the TuscanySCATours Atom feed. Because `binding.atom` is a Tuscany binding, you must define the Tuscany XML namespace.

It's time to turn your attention to the TuscanySCATours feed logger implementation code. The previous composite used the `FeedLoggerImpl` class as the Java implementation. The code for this Java class is shown in the following listing. It can also be found in the `contributions/feed-logger` directory of the TuscanySCATours application.

Listing 8.13 Fragment of the `FeedLoggerImpl` class

```

public class FeedLoggerImpl implements FeedLogger {

    @Reference
    public org.apache.tuscany.sca.binding.atom.collection.Collection
        scaToursBlogAtom;

    public void logFeeds(int maxEntriesPerFeed) {
        System.out.println("Logging SCA Tours Blog Atom feed:");
        logAtomFeed(scaToursBlogAtom, maxEntriesPerFeed);
    }

    private void logAtomFeed(org.apache.tuscany.sca.binding.
        ↳ atom.collection.Collection atomFeed, int maxEntries) {
        final Feed feed = atomFeed.getFeed();
        System.out.println("Feed: " + feed.getTitle());
        final List<Entry> entries = feed.getEntries();

        for (int i = 0; i < entries.size() && i < maxEntries; i++) {
            Entry entry = entries.get(i);
            System.out.println("Entry: " + entry.getTitle());
        }
        System.out.println();
    }
}

```

1 Injected SCA Tours Atom feed reference

2 Get Atom feed entries

3 Log Atom feed entries

The `FeedLoggerImpl` class has an injected reference to the TuscanySCATours Atom feed ① that it uses to retrieve all the feed entries ② and log their titles ③ to the console.

Now that your composite and Java component implementation are complete, try running the TuscanySCATours feed logger using the `launchers/feed-logger` launcher from the TuscanySCATours application. When run, the TuscanySCATours feed logger will output the following messages to the console:

```
Logging SCA Tours Blog Atom feed:
Feed: SCA Tours Blog
Entry: Apache Tuscany in Action
Entry: Hello and welcome to SCA Tours
```

From this output, you can see that the TuscanySCATours feed logger has pulled two entries off the Atom feed. Note that when you run the TuscanySCATours feed logger, the titles of the feed entries may be different if newer entries have been posted to the TuscanySCATours blog.

8.6.2 Logging the TuscanySCATours blog RSS feed

It's now time to turn your attention to adding the TuscanySCATours RSS feed to the TuscanySCATours feed logger. The URL for this RSS feed is <http://scatours.wordpress.com/?feed=rss>. To make sure that it's working correctly, it's probably worth attempting to access the feed URL in your web browser. Update the TuscanySCATours feed logger composite XML file to include the TuscanySCATours blog RSS feed. It can be found in the `feed-logger.composite` file in the `contributions/feed-logger` directory of the TuscanySCATours application and is shown here.

Listing 8.14 TuscanySCATours feed logger composite accessing Atom and RSS feeds

```
<composite xmlns="http://www.oesia.org/xmlns/sca/1.0"
  xmlns:tuscany="http://tuscany.apache.org/xmlns/sca/1.0"
  targetNamespace="http://tuscanyscatours.com/"
  name="feedLogger">

  <service name="FeedLogger" promote="FeedLogger">
  </service>

  <component name="FeedLogger"
    <implementation.java class="com.tuscanyscatours.feedlogger.impl.
      FeedLoggerImpl"/>
    <reference name="scaToursBlogAtom">
      <tuscany:binding.atom
        uri="http://scatours.wordpress.com/?feed=atom"/>
    </reference>
    <reference name="scaToursBlogRSS">
      <tuscany:binding.rss
        uri="http://scatours.wordpress.com/?feed=rss"/>
    </reference>
  </component>
</composite>
```

In this composite, you've added a new reference called `scaToursBlogRSS` **1** and added an RSS binding to it that points at the TuscanySCATours blog RSS feed **2**.

Let's update the TuscanySCATours feed logger implementation code to use the new reference to log the RSS feed. Unfortunately, Tuscany currently doesn't support

generic consumption of feeds via the Tuscany Data API. But support for it may be added in later versions of Tuscany. Therefore, you'll need to update the code in the `FeedLoggerImpl` class to handle the RSS feed, as shown here. The full source code can be found in the `contributions/feed-logger` directory of TuscanySCATours.

Listing 8.15 Updated the `FeedLoggerImpl` class with RSS logging added

```
public class FeedLoggerImpl implements FeedLogger {

    @Reference
    public org.apache.tuscany.sca.binding.atom.collection.Collection
        scaToursBlogAtom;

    @Reference
    public org.apache.tuscany.sca.binding.rss.collection.Collection
        scaToursBlogRSS;

    public void logFeeds(int maxEntriesPerFeed) {
        System.out.println("Logging SCA Tours Blog Atom feed:");
        logAtomFeed(scaToursBlogAtom, maxEntriesPerFeed);

        System.out.println("Logging SCA Tours Blog RSS feed:");
        logRSSFeed(scaToursBlogRSS, maxEntriesPerFeed);
    }

    private void logAtomFeed(org.apache.tuscany.sca.binding.
    ➤ atom.collection.Collection atomFeed, int maxEntries) {
        ...
    }

    private void logRSSFeed(org.apache.tuscany.sca.binding.
    ➤ rss.collection.Collection rssFeed, int maxEntries) {
        SyndFeed feed = rssFeed.getFeed();
        System.out.println("Feed: " + feed.getTitle());
        List<SyndEntry> entries = feed.getEntries();

        for (int i = 0; i < entries.size() && i < maxEntries; i++) {
            SyndEntry entry = entries.get(i);
            System.out.println("Entry: " + entry.getTitle());
        }
        System.out.println();
    }
}
```

1 Injected SCA Tours RSS feed reference

2 Get RSS feed entries

3 Log RSS feed entries

The updated `FeedLoggerImpl` class now has an additional injected reference to the TuscanySCATours RSS feed **1** that it uses to retrieve all the feed entries **2** and log their titles **3**.

Run the TuscanySCATours feed logger again using the `launchers/feed-logger` launcher from the TuscanySCATours application. This time, you'll get the following messages to the console:

```
Logging SCA Tours Blog Atom feed:
Feed: SCA Tours Blog
Entry: Apache Tuscany in Action
Entry: Hello and welcome to SCA Tours
```

```
Logging SCA Tours Blog RSS feed:  
Feed: SCA Tours Blog  
Entry: Apache Tuscany in Action  
Entry: Hello and welcome to SCA Tours
```

From this output, you can see that the TuscanySCATours feed logger has pulled the same entries from the TuscanySCATours blog over the Atom feed and the RSS feed. Simple, wasn't it?

8.7 *Summary*

In this chapter, you saw how you can create web clients in SCA using a variety of technologies including servlets, JSPs, JSON-RPC, Atom, and RSS feeds. The various SCA implementation types and bindings supported by Tuscany are able to simplify creating these web clients by hiding some of the complexities.

You've been able to develop both client-side (HTML/JavaScript) and server-side (servlet/JSP) user interface components that make use of the same backend SCA components. The SCA composite model presents a coherent picture of the client and server while hiding some of the complexities of how they're wired together.

Over the last two chapters, you've seen how different technologies can be wired together with SCA. It's time to move to the next chapter and see how SCA can help in transforming data between technologies that use different vocabularies.

Tuscany SCA IN ACTION

Laws • Combella • Feng • Mahbod • Nash



Tuscany SCA is a technology-neutral infrastructure for building composite applications based on the Service Component Architecture standard. It manages the protocols and other application plumbing, enabling you to focus on business logic and the relationship between services. The resulting applications are more flexible, scalable, and maintainable.

Tuscany SCA in Action is a comprehensive, hands-on guide for developing technology-agnostic, extensible applications. By following a travel-booking example throughout the book, you'll learn how to model, compose, deploy, and manage applications using SCA. The book emphasizes practical concerns, like effectively using Tuscany's supported bindings and protocols and integrating with standard technologies like Spring and JMS to save development time and cost.

What's Inside

- Introduction to Tuscany
- Coverage of Service Component Architecture
- Practical examples and techniques
- Written by core Tuscany committers

This book is for developers interested in service-oriented applications. No experience with Tuscany or SCA is required.

Simon Laws is a member of the IBM Open Source SOA project focused on building the Java runtime for Service Component Architecture (SCA). Coauthors **Mark Combella**, **Raymond Feng**, **Haleh Mahbod**, and **Simon Nash** are all Tuscany committers.

For online access to the authors and a free ebook for owners of this book, go to manning.com/TuscanySCAinAction

“A great resource.”

—Jeff Davis
Author of *Open Source SOA*

“A must-have guide.”

—Alberto Lagna, biznology.it

“The A-Z of SCA from the source.”

—Ara Ebrahimi
CityGrid Media

“Offers insights and techniques that help you put it all together.”

—Doug Warren
Java Web Services

“Start being productive with Tuscany SCA right away.”

—Jeff Anderson, Deloitte

ISBN-13: 978-1-933988-89-4
ISBN-10: 1-933988-89-4



9 781933 1988894