WebSphere. software

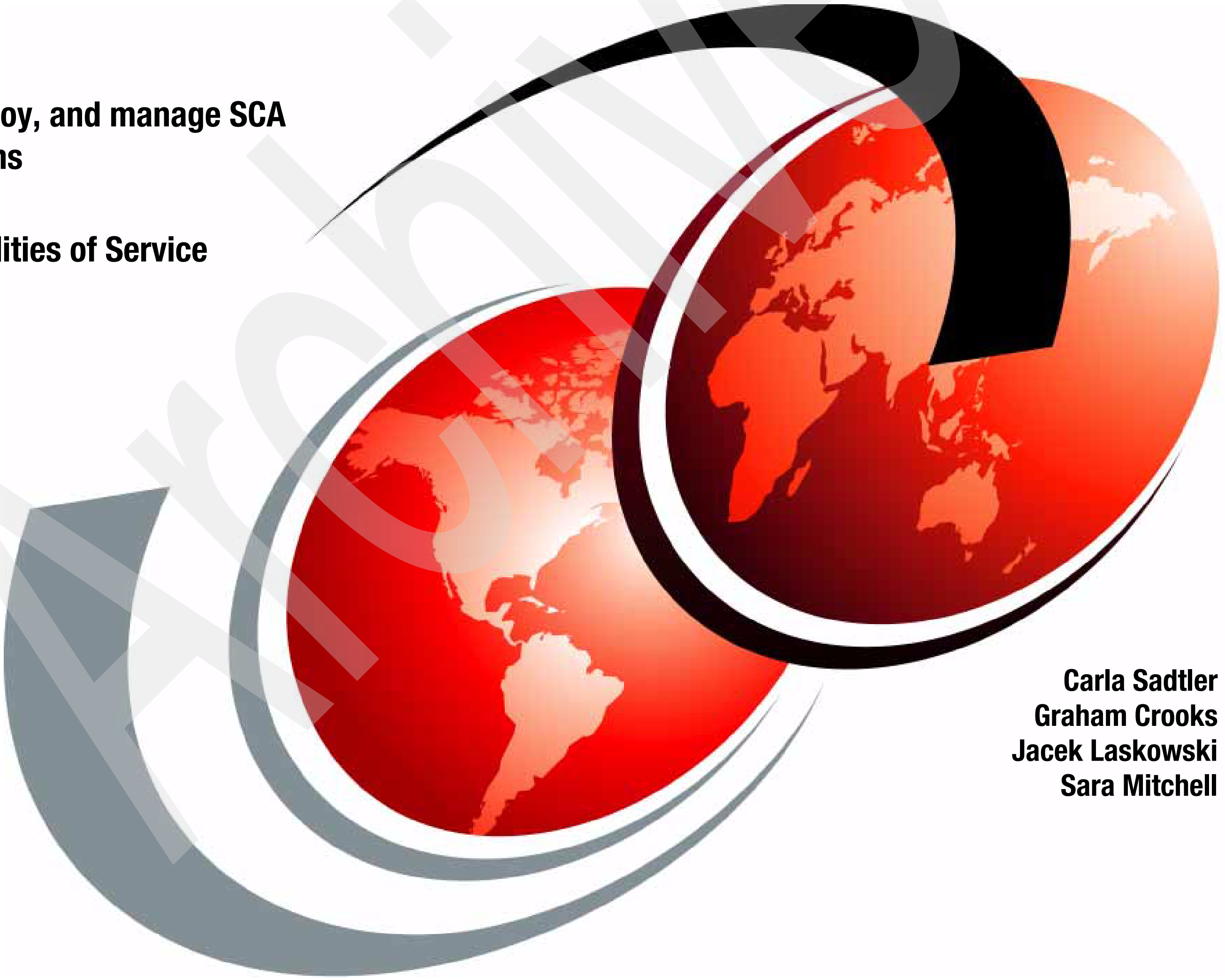
IBM

Getting Started with WebSphere Application Server Feature Pack for Service Component Architecture

Learn about the new features

Build, deploy, and manage SCA applications

Apply Qualities of Service



Carla Sadtler
Graham Crooks
Jacek Laskowski
Sara Mitchell

Redpaper

ibm.com/redbooks

International Technical Support Organization

Getting Started with WebSphere Application Server Feature Pack for Service Component Architecture

March 2010

REDP-4633-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (March 2010)

This edition applies to WebSphere Application Server V7 Feature Pack for Service Component Architecture Version 1.0.1.

Contents

Notices

Trademarks

Preface

Service Component Architecture (SCA) defines a service-based model for building business process applications using an SOA approach. This ability to drive a business process using individual, reusable services is the heart of the SOA concept. With IBM® WebSphere® Application Server Feature Pack for Service Component Architecture, you can deploy SCA applications to WebSphere Application Server.

This IBM Redpaper™ publication provides a starting point for using the Feature Pack for SCA. It provides an architectural view of SCA and of the Feature Pack. In addition, this paper explains how to create simple SCA components from existing Java™ and Spring implementations. It discusses how to apply quality of service to applications, and how to deploy and manage SCA artifacts in WebSphere Application Server. The examples in this paper use Rational® Application Developer to illustrate how to create and package SCA applications.

The team who wrote this paper

This paper was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Rochester Center.

Carla Sadtler is a Consulting IT Specialist at the ITSO, Raleigh Center. She writes extensively about WebSphere products and solutions. Before joining the ITSO in 1985, Carla worked in the Raleigh branch office as a Program Support Representative, supporting MVS customers. She holds a degree in mathematics from the University of North Carolina at Greensboro.

Graham Crooks is an Application Architect within the IBM Developer Relations architects team and is based in Melbourne, Australia. He has worked for IBM for 10 years, focusing on SOA, application integration, and more recently on integration of ISV applications into the IBM Industry Frameworks. He works across all of the IBM brands with specialities in WebSphere Enterprise Service Bus, WebSphere Message Broker, WebSphere Process Server, and associated technologies.

Jacek Laskowski is a IBM WebSphere IT Specialist and is responsible for IBM Business Process Management solutions in IBM Software Group, Poland. He is experienced with enterprise open source and commercial Java products and frameworks. He is an ASF committer (Apache Geronimo, Apache OpenEJB, and others), a leader and founder of Warszawa Java User Group (Warszawa JUG), and a NetBeans dream team member. He has authored other IBM Redbooks® publications. His area of expertise includes, but is not limited to, Java Enterprise Edition (JEE) and WebSphere BPM product stack. He holds a Master's degree in Computer Science from the University of Warsaw, Poland.

Sara Mitchell is a Web Services architect and team lead in the U.K. working from the Hursley Development Laboratory in Hampshire. She has 8 years of experience in IBM working on WebSphere Application Server. Her areas of expertise include Web services standards, especially WS-Addressing, WS-ReliableMessaging, and WS-Notification. She is a member of the W3C WS-ReliableMessaging Working Group and is a member of the Apache Web Services Project Management Committee. She also co-authored the developerWorks® article, *Using Spring and Hibernate with WebSphere Application Server*. She regularly presents at the WebSphere User Group in the UK. She holds a Bachelor of Science degree in Computer Studies from Brighton University.

Thanks to the following people for their contributions to this project:

- Scott Kurz
  IBM U. S.
- Louis Amodeo
  IBM U. S.
- Sean Zhou
  IBM U. S.
- Jennifer Thompson
  IBM U. S.
- Will Edwards
  IBM U. S.
- Greg Dritschler
  IBM U. S.

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks
- Send your comments in an e-mail to:

  redbooks@us.ibm.com
- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks publications

- Find us on Facebook:

  http://www.facebook.com/pages/IBM-Redbooks/178023492563?ref=ts

- Follow us on twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Feature Pack for SCA technical overview

Feature packs offer targeted, incremental new features for WebSphere Application Server. The WebSphere Application Server V7 Feature Pack for SCA takes advantage of core components of Apache Tuscany to deliver an implementation of the Open SOA Collaboration (osoa.org) SCA 1.0 programming model for composite application development and management in an SOA environment.

The Feature Pack for SCA can be downloaded from:

`http://www-01.ibm.com/software/webservers/appserv/was/featurepacks/sca/`

You can find information about Apache Tuscany at:

`http://tuscany.apache.org/`

This chapter provides an overview of SCA and how the WebSphere Application Server V7 Feature Pack for SCA implements it. It looks at this implementation from the architectural perspective as well as from the perspective of a developer using Rational Application Developer, and an administrator who deploys the application to WebSphere Application Server.

For information about further resources, see “Related publications” on page 133.

1.1 What is SCA?

Service Component Architecture (SCA) defines a model for the construction, assembly, and deployment of services using an SOA approach. With SCA, services are represented as components that are wired together in a logical flow, with each component providing a business function. Each component has its own implementation technology (for example, Java) and access technology (for example, Web services or JMS) that are suitable for the implementation. This ability to drive a business process using individual, reusable services is the heart of the SOA concept.

A classic example often used to show the capabilities of SOA is that of a banking application. Components that represent business functions, such as get balance, deposit funds, withdraw funds, transfer funds (withdraw then deposit), or loan application, can all be wired to a user interface that allows customers to perform online banking.

SCA is not new to IBM WebSphere products. WebSphere Enterprise Service Bus and WebSphere Process Server both provide users the ability to build SCA applications. Those products use what we refer to as a *classic* SCA implementation. In WebSphere Process Server, the component implementation is typically a BPEL process that consists of a flow of activities that manipulate business data to perform a function. Data flows among the components as business objects, based on the Service Data Objects (SDO) specification. In WebSphere ESB, the component implementation is a mediation module. Data flows among the components as service message objects (SMO), which are enhanced SDOs.

The WebSphere Application Server Feature Pack for SCA provides support based on the Open SOA Collaboration SCA 1.0 programming model. For a list of the portions of the specification that are not supported, see:

`http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/csca_spec_gaps.html`

For information about the Open SOA Collaboration SCA programming model, see:

`http://osoa.org`

If you are familiar with WebSphere Process Server, you will notice some differences in the architecture. For example, the classic SCA used by WebSphere Process Server wires Interfaces and References to Export and Import components respectively. Bindings that define the access methods are applied to the Export and Import Components. In Open SCA, there are no Import and Export components. Bindings are applied directly to the Interfaces and References. So, if you have a WebSphere Process Server background, read the specification and keep an open mind as you go along!

Interoperability with WebSphere Process Server: The SCA modules of WebSphere Process Server V7 use Import and Export bindings to interoperate with Open SCA services that are developed in a Rational Application Developer environment and that are hosted by the WebSphere Application Server Feature Pack for SCA.

1.2 Getting started with the Feature Pack for SCA

You can install the Feature Pack for SCA on a WebSphere Application Server V7.0.0.7 or later system. WebSphere Application Server customers can download the Feature Pack from:

http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg27008534

You can also install the IBM Rational Application Developer V7.5 SCA Tools to provide support for SCA development. You can install the tools using the Update option of the Installation Manager.

You can find more information about the installation process Chapter 2, "Installation tips" on page 13.

This paper uses samples that ship with Rational Application Developer and the Feature Pack for SCA. The following sections describe these samples.

1.2.1 Samples that ship with the Feature Pack for SCA

To find the samples that ship with the Feature Pack for SCA, navigate to the following directory:

```
app_server_root/samples/SCA
```

For example, if you installed the Feature Pack on an instance of WebSphere Application Server, the path might be:

```
C:\Program Files\IBM\WebSphere\AppServer\samples\SCA
```

If you installed the Feature Pack on the WebSphere Application Server test environment in Rational Application Developer, the path might be:

```
C:\Program Files\IBM\SDP\runtimes\base_v7\samples\SCA
```

The samples that ship with the Feature Pack for SCA come with instructions on how to build the sample for deployment and how to test the sample. Although specific instructions are not provided, you can import the samples into a Rational Application Developer workspace.

Typically, you need to follow the instructions that come with the sample to generate the target code. Then, review the documentation that comes with the sample to understand the structure of the artifacts in the sample. For example, if the artifact is a Web application, you need to import a WAR or EAR file into the workspace. If it is an SCA composite, you need to import the JAR file as an SCA archive file.

1.2.2 Samples that ship with the SCA tools for Rational Application Developer

To find the samples that ship in with the SCA tools, open a workspace, and select **Help** → **Help Contents**. SCA application samples are in the **Tutorials** → **Do and Learn** section, as well as in the **Samples** → **Technology samples** → **SCA** section.

The samples that ship with Rational Application Developer come with instructions about how to import the sample into your workspace and how to use them.

1.3 Feature Pack for SCA architecture

SCA composites are the deployment unit for SCA in WebSphere Application Server. Composites consist of SCA components that implement business functions in the form of services. To deploy SCA composites, you import them as assets to the product repository and add the assets to business-level applications.

Figure 1-1 shows the elements of an SCA composite.

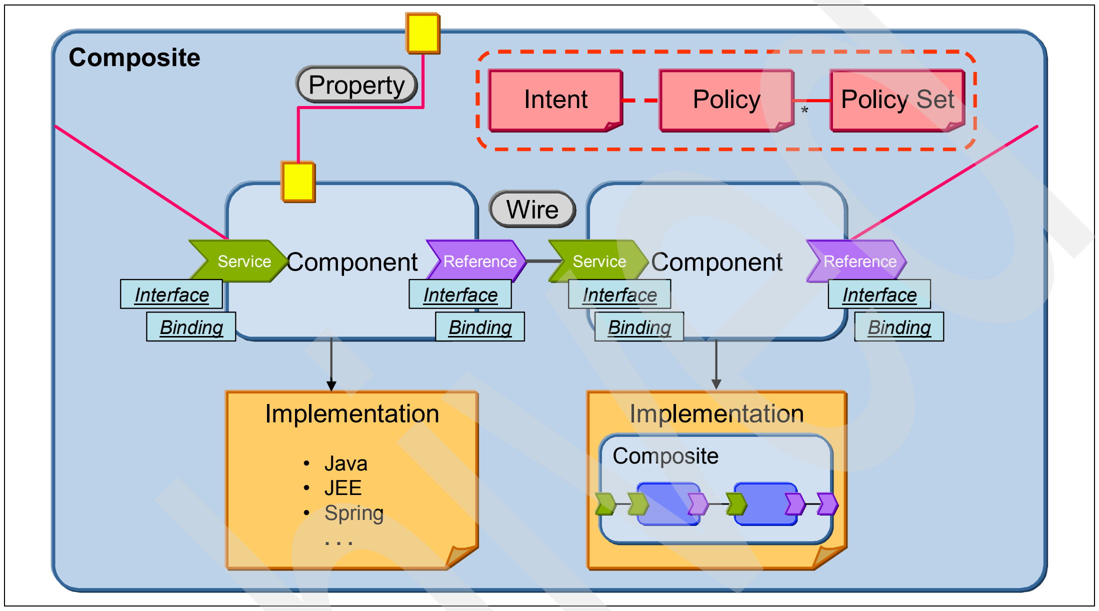


Figure 1-1 Feature Pack for SCA architectural view

As illustrated in the figure, an SCA composite includes the following elements:

- An *SCA component* (or simply *component*) is the basic element in an SCA application. A component exposes an *SCA service* (or simply *service*). Components point to the code that implements their service. The supported implementation types for a service vary with the middleware environment. In the Feature Pack for SCA, the service implementation includes the following components:
  - Java POJO
  - Java Platform, Enterprise Edition (Java EE) integration
  - Other SCA composites
  - Spring 2.5.5 containers
- When the SCA service requires the use of another service (SCA or otherwise), a *reference* provides the point for the application to call the next service.
- Services and references have an *interface* definition that points to the location of the WSDL or Java that describes the interface. The *service interface* provides the information that clients use to call the service. A *reference interface* provides the information the component needs to call another service.
- Services and references are bound to specific protocols through the usage of *bindings*. The use of bindings removes the details of connection from the business logic. The bindings supported in the Feature Pack for SCA V1.0.1 are SCA, Web service, EJB, JMS, Atom, and HTTP.
- SCA components are assembled into an *SCA composite file*. A composite can be reused to provide the implementation for other components in a nested fashion. The SCA

composite file is the deployable unit for WebSphere Application Server. SCA composites are deployed to an SCA domain. The SCA domain is typically the cell on multiple-server installations and the server scope on single-server installations.

- Multiple services are *wired* into an orchestrated flow. For example, a business process that processes loans can be assembled from a collection of SCA and non-SCA components that perform the activities required to process a loan (get credit check, query accounts, and so forth). The wiring provides the information that is needed to find the next service.

  There are four wiring options for a reference in a composite:

  – The reference is wired to a service in the same composite. (Note that this wiring is the only instance where you will see a representation of a wire in a composite in a Rational Application Developer work area.) The composite file has a target attribute or wire element on the reference element.

  – The reference is wired to a service in a different composite, also accomplished with a target attribute or wire element.

  – The reference is wired by the autowire option, which is defined at the composite level.

  – Wiring is done at the binding level by specifying the endpoint URI for a Web services binding or by specifying the invocation URI for an EJB binding.

- Policy and quality of service intents can decorate services or references (called *interaction intents*) and can decorate components (called *implementation intents*.)
- The components, assemblies, internal wires, and service and reference definitions are written in an open XML language called *Service Component Definition Language* (SCDL).

1.3.1 Component implementation types

You can implement components as Java, JEE, or Spring applications or as other composites. Note that prior to the Feature Pack for SCA V1.0.1, Java and Composite were the only selections. JEE and Spring are new with V1.0.1.

Java implementation

You can implement SCA components using Java Plain Old Java Object (POJO) code using the following methods:

- Top-down development

  For top-down development, you create a new service from an existing WSDL or Java file. This method is considered best practice. For more information, see:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/tsca_devscawsdl_topdown.html`

  With top-down development, keep in mind the following considerations:

  – You must use WSDL Version 1.1 definitions that also conform to the WS-I Basic Profile Version 1.1 and Simple SOAP Binding Profile 1.0 standards, and you must use the document literal style.

  – The Feature Pack for SCA does not support using a WSDL file when the Java mapping requires holder classes.

  The following tools are also available:

  – Use the ***`app_server_root`*****`/bin/wsimport`** command to generate Java classes that you can use to write a Java implementation. The result is a POJO implementation of the generated interface using the generated JAXB data types. Add the `@Service` annotation to the Java implementation to define the Java implementation as an SCA service implementation. This process is detailed in the previous information center reference.

  – Use the Rational Application Developer New SCA Component wizard to create the component and Java classes.

- Bottom-up development

  You can create a new service from existing Java code that uses Java Architecture for XML Binding (JAXB) data types. For more information, see:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/tsca_devsca_bottomup.html`

JEE implementation

You can use JEE applications as SCA component implementations. Java EE applications can take advantage of SCA programming model with little or no change to its implementation code. JEE applications provide the following benefits:

- An SCA client can invoke an EJB stateless session bean in an enterprise application as an SCA service.
- JEE components, such as session beans, message-driven beans, or Web components, can use SCA annotations to invoke an SCA service.
- An SCA service that is implemented as an EJB bean can invoke another SCA service.

JEE applications used as SCA component implementations can be enhanced for SCA or non-SCA enhanced as follows:

- You can use an enterprise application as an SCA component implementation without requiring any SCA annotations or composite files in the application. Such an application is denoted as a *non-SCA enhanced* enterprise application. The EJB business interfaces of stateless session beans that are configured in EJB modules are exposed as SCA services. The EJB business interfaces of EJB references that are configured in EJB and Web modules are exposed as SCA references. For more information, see:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/fep/topic/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/tsca_jee_nonenhanced.html`

- You can also use a JEE application as an SCA component after enhancing the application with SCA annotations. SCA annotations are used to enable Java EE components such as stateless session beans, message-driven beans, servlets, listeners, filters, and JavaServer Pages (JSP) files to consume SCA services and properties. In an enhanced application, you can rewire EJB references in EJB and Web modules to SCA references. For more information, see:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/fep/topic/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/tsca_jee_enhanced.html`

The Feature Pack for SCA includes a sample application, HelloJEE, that illustrates these uses. You can find the sample application in the following directory:

app_server_root`\samples\SCA\HelloJee\`

Spring implementation

You can use beans that follow the Java SE programming model in a Spring 2.5.5 container as component implementations.

The Feature Pack for SCA supports only the Spring Framework that uses the Java SE programming model. Local JNDI lookups are not supported, thus the product does not support the Spring Framework using the JEE 5 programming model. Spring does not ship with Rational Application Developer or the Feature Pack for SCA.

1.3.2 Service and reference interfaces

Interfaces are added to components to enable other applications or components to invoke the service the component provides. An *interface* describes the services offered by the component; however, an interface does not specify the technology that used to access the component. You specify the technology by applying a binding to the interface. A component can have multiple interfaces, and an interface can have multiple bindings.

References are added to components to enable the implementation to invoke another service, either within the same SCA domain or externally. A component can have multiple references, and a reference can have multiple bindings.

Both references and interfaces can be written as WSDL 1.1 or Java.

The Feature Pack for SCA supports the following binding types:

- SCA binding

  Used by an SCA service to invoke another SCA service in the same SCA domain.

- Web service binding

  Used for SOAP-based Web Services-Interoperability (WS-I) compliant Web services. This binding defines the manner in which an SCA service is made available as a Web service and in which a reference can invoke or access a Web service. Only WSDL Version 1.1 is supported. The Web service binding either references an existing WSDL binding or enables you to specify enough information to generate a WSDL file.

- EJB binding

  Used by an SCA service to invoke session beans as an SCA service and to expose SCA services as stateless session beans to external clients. Support is provided for the EJB binding when using both 2.x and 3.0 EJB styles for both the SCA service and reference.

- JMS binding

  Used by an SCA service to invoke messaging applications using the JMS programming interface, and to expose SCA services to applications using JMS. SCA services using the JMS binding are exposed using the Java EE Connector Architecture (JCA)-based messaging provider in WebSphere Application Server. The Feature Pack for SCA supports the default messaging provider or WebSphere MQ as the messaging engine.

- Atom binding

  Used to work with SCA services that provide or consume entries described in the Atom Syndication Format and Atom Publishing Protocol. An SCA service can reference existing external Web feeds defined using the Atom protocol and work with them inside a Java implementation. Also, you can use the Atom binding to compose new SCA services and expose them as an Atom feed.

- HTTP binding

  Used to expose SCA services for consumption by remote JavaScript-enabled Web browser clients. Using this binding, clients can invoke Remote Procedure Calls (RPC) to server-side SCA components.

You can find examples of these bindings in the various samples that ship with the Feature Pack for SCA and with Rational Application Developer.

1.4 From a developer's perspective

As an example, assume that you need to provide a service to a business application. Using Rational Application Developer, you would perform the following typical sequence of tasks:

1. First, use the New SCA Project Wizard to create an SCA project to hold the composite and its elements.
2. Next, use the new SCA Component wizard to create a new SCA composite and component.
   a. Select the type of interface to use for the component and whether to use an existing interface or create a new one.
   b. Select the type of implementation for the component and whether an existing implementation should be used or a new one created. Note that you can change the implementation of a component after it is created.

   Specific combinations of these options are supported by the new SCA component wizard. For example, for top-down development, you select an existing WSDL file as the interface and specify that you want a new Java implementation created based on the WSDL file.

   The wizard also creates a new composite to hold the component.
3. Add a binding for the component's service interface (for example, a Web service binding).
4. Create the implementation code that provides the business function. For Java components, for example, open the new Java implementation file, and enter the code.
5. Add references to the component for each service that it will invoke. Configure the references by adding the interface information that tells how to invoke the service and by adding binding information that tells what protocol to use.
6. Create a contribution file for the composite in the project, and package the project as an SCA archive.

Now, the new service is ready to deploy to WebSphere Application Server as an asset for a business-level application.

1.4.1 Looking at the elements in the composite

Note: The Rational Application Developer tools provide support for building and working with SCA applications. The example we provide here is intended to give you a feel for the actual structure of an SCA composite and does not highlight the use of those tools. Later examples will show how those tools can simplify the task of developing applications.

Our example looks at the construction of the files in an SCA application, which includes the following components:

- A composite with one component that is implemented with Java code.
- A service interface with a Web service binding.
- A reference to an external Web service using a Web service binding.

The example uses a Web service from the Address Book sample that is provided as part of the Rational Application Developer v 7.5 tutorials as the external service. You can find the Web service example by opening the Welcome page (**Help** → **Welcome**) in the workspace. You will find it listed as the "JAX-WS Address book secured with the RSP policy set" under the Samples category. You might need to tab to a second page of samples to find it.

The following steps illustrate how the sample SCA application is developed and the composite file that results:

1. First, create an SCA project in the Rational Application Developer workspace.
2. Then, in the SCA project, create the `RemoteAddressBook` interface for the component's service, and add the `@Remotable` annotation to the interface as shown in Example 1-1.

   Example 1-1 Interface code

   ```
   package bankutils;
   import org.osoa.sca.annotations.Remotable;
   import com.addressbook.AddressType;
   import com.addressbook.PersonType;
   @Remotable
   public interface RemoteAddressBook {
         public boolean saveRemoteAddress(PersonType person);
         public AddressType findRemoteAddress(String name);

   }
   ```

3. Using the Rational Application Developer tools, create an SCA component from the interface class, which generates an empty implementation class automatically.

   The method implementations call the relevant methods on the Web service, delegating the functionality to the Web service.

   Notice in Example 1-2 that we added the `@Reference` declaration to an instance of `AddressBook`, which is the generated Web Service class. Other types are also needed and are imported.

   Example 1-2 Component implementation

   ```
   package bankutils;
   import org.osoa.sca.annotations.Reference;
   import org.osoa.sca.annotations.Service;
   import com.addressbook.AddressBook;
   import com.addressbook.AddressType;
   import com.addressbook.FindAddressFault;
   import com.addressbook.PersonType;
   @Service (RemoteAddressBook.class)
   public class RemoteAddressBookImpl implements RemoteAddressBook {
        @Reference
        public AddressBook AddressBookReference;
        public boolean saveRemoteAddress(PersonType person) {
               return AddressBookReference.saveAddress(person);
            }
        public AddressType findRemoteAddress(String name) {
               try {
                  return AddressBookReference.findAddress(name);
             } catch (FindAddressFault e) {
                  System.out.println("problem finding address   ...");
                  e.printStackTrace();
   ```

```
            }
            return new AddressType();
        }
}
```

4. For the external AddressBook Web service that the component references, copy the WSDL for the Web service to the SCA Project and generate a JAX-WS interface from the WSDL to give the classes that you need to use the referenced service.

   The SCA composite file describes the components in the composite. It contains the name of the component, the location of the implementation, the service interface name, the reference name, and the binding information.

   The reference element in the file must contain a name that matches the variable name of the `@Reference` annotation shown in Example 1-2. In this example, the reference points at an external Web service so that it points to the WSDL file and so that it contains a Web services binding (`binding.ws`). As you build your components in Rational Application Developer, the composite file (shown in Example 1-3) is updated automatically with the elements of the composite.

Example 1-3 Composite XML file

```
 <?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" autowire="false"
name="RemoteAddressComposite" targetNamespace="http://mybank.bankutils">
  <component name="RemoteAddressComponent">
     <implementation.java class="bankutils.RemoteAddressBookImpl"/>
     <service name="RemoteAddressBook">
        <interface.java interface="bankutils.RemoteAddressBook"/>
        <binding.ws/>
     </service>
     <reference name="AddressBookReference">
        <interface.wsdl
           interface="http://addressbook.com/#wsdl.interface(AddressBook)"/>
        <binding.ws name="AddressBook"
           uri="http://localhost:9081/jwsAddressBook/AddressBookService"

wsdlElement="http://addressbook.com/#wsdl.service(AddressBookService)"/>
     </reference>
  </component>
</composite>
```

1.5 From an administrator's perspective

The Feature Pack for SCA extends the business-level application functionality of WebSphere Application Server V7. A *business-level application* is an administrative concept that defines applications at a business level. The business-level application can encompass both WebSphere and non-WebSphere artifacts.

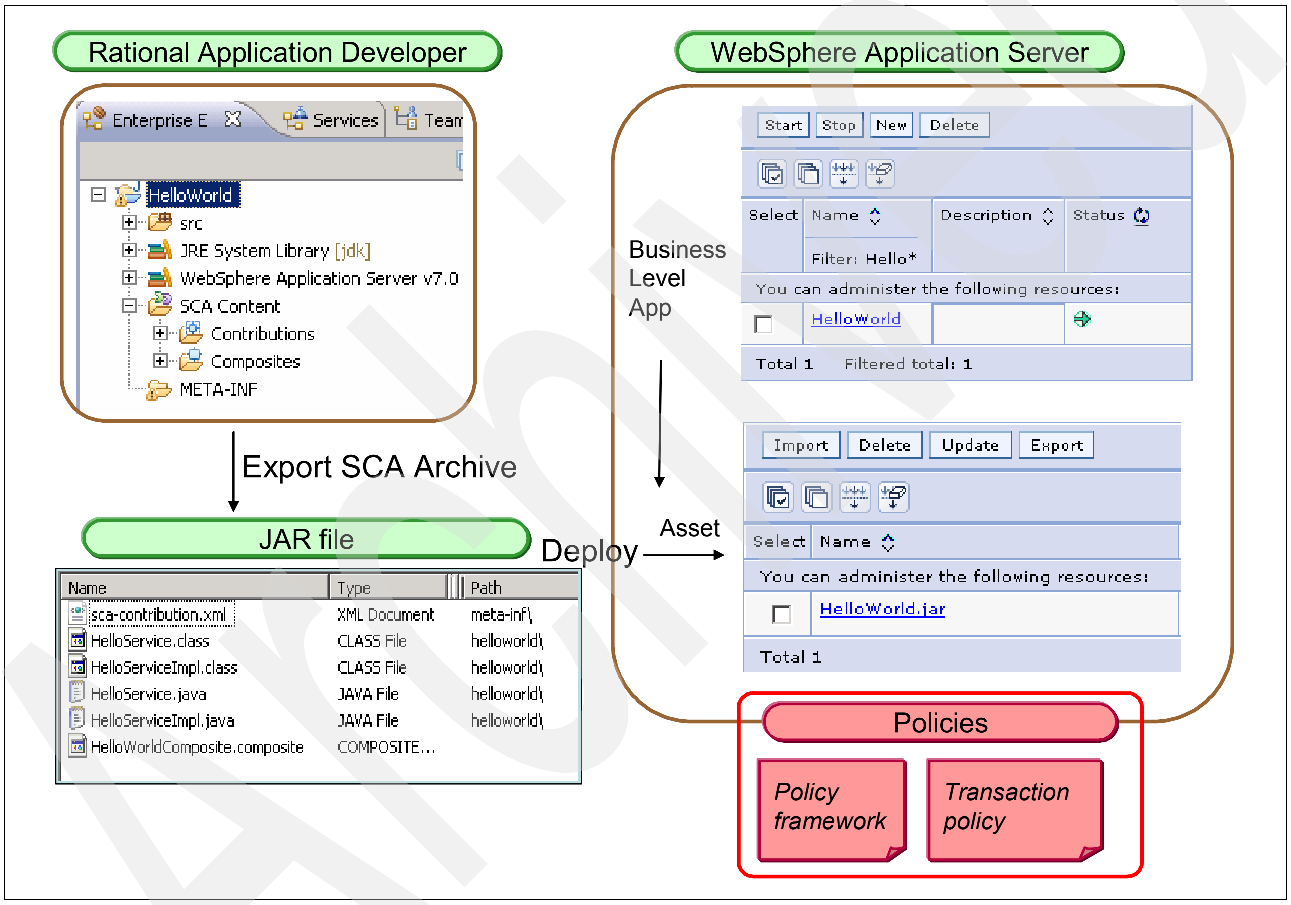
The business-level application itself does not represent or contain application files. Rather, it contains composition units that are linked to deployable assets such as JAR, WAR, or EAR files. A composition unit contains configuration information for the asset. For example, the server the asset will run on and references other resources that the asset uses.

With the Feature Pack for SCA, the composition units are extended to include SCA archives as follows:

- SCA composites are packaged as SCA archives for deployment.
- An SCA archive is imported into the WebSphere Application Server environment as an asset.
- The asset is then added to a business-level application as a composition unit that includes the configuration information that is required to execute it.

Note that despite the similarity between the terms *composition unit* and *composite*, the term similarities are a coincidence. An SCA composite packaged as a JAR file can become an asset and can become a composition unit after it is imported into a business-level application.

Figure 1-2 illustrates the deployment aspects of an SCA composite.

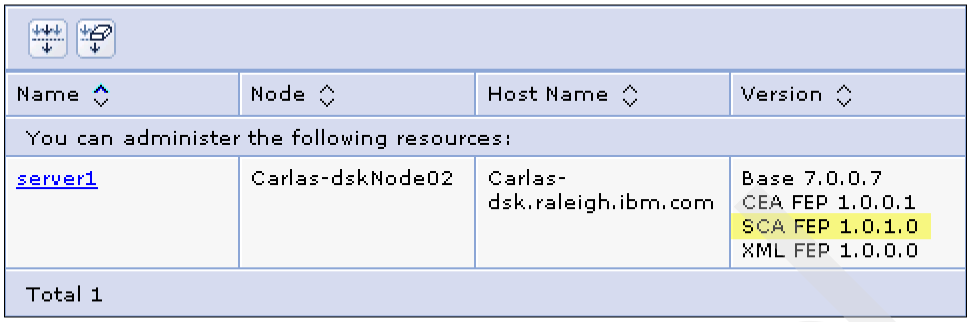


Name	Type	Path
sca-contribution.xml	XML Document	meta-inf\
HelloService.class	CLASS File	helloworld\
HelloServiceImpl.class	CLASS File	helloworld\
HelloService.java	JAVA File	helloworld\
HelloServiceImpl.java	JAVA File	helloworld\
HelloWorldComposite.composite	COMPOSITE...	

Figure 1-2 Deploying SCA composites

The SCA archive contains a JAR file that includes the Java implementation and a special XML file called `sca-contribution.xml`, referred to as a *contribution*. The contribution contains information about which composites in the deployable JAR file are executable, the namespaces of those composites, and which composites can be used by other contributions.

SCA archives must be installed in a server or cluster that is enabled for the Feature Pack for SCA. You can verify that the server is enabled by displaying it from the administrative console. Select **Servers** → **Server Types** → **WebSphere application servers**. Then, find the server name in the list of servers, and make sure that *SCA FEP* is listed in the Version column as shown in Figure 1-3.



Name	Node	Host Name	Version
You can administer the following resources:			
server1	Carlas-dskNode02	Carlas-dsk.raleigh.ibm.com	Base 7.0.0.7 CEA FEP 1.0.0.1 SCA FEP 1.0.1.0 XML FEP 1.0.0.0
Total 1			

Figure 1-3 Verifying a server is enable for the Feature Pack for SCA

Notes on contributions:

When a contribution does not contain the `sca-contribution.xml` file, it can contain a deployable composite under the `META-INF/sca-deployables` directory. The composite file must be named `default.composite`. The `default.composite` file is automatically the deployable composite.

You can use the `sca-contribution.xml` file to create a contribution with multiple deployable composites. When you import the asset, each composite is identified as a deployable unit. However, when you add the asset to a business-level application, you can select only one of the composites.

For more information about SCA contributions, see:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/csca_contribution.html

2

Installation tips

You install and update the WebSphere Application Server Feature Packs using the IBM Installation Manager. This process is different from the installation and maintenance processes used for WebSphere Application Server. Although those familiar with installing eclipse-based IDEs from IBM will be familiar with using the IBM Installation Manager, this process might be new to WebSphere Application Server administrators. For this reason, this chapter provides a quick outline of the steps needed to install the Feature Pack for SCA. You can use this information along with the documentation that comes with the products and maintenance packages.

2.1 On Rational Application Development systems

The primary development tool used to create applications for WebSphere Application Server is Rational Application Developer. When you develop applications in Rational Application Developer, you can deploy them to the test environment, which is a fully functional WebSphere Application Server that is integrated into the workspace with an interface for developers.

You can install the Feature Packs on the test environment as well as to a stand-alone WebSphere Application Server environment.

Installing the Feature Packs on Rational Application Development systems involves the following basic steps:

1. To get started, install Rational Application Developer. You can start the installation by executing the `launchpad.exe` utility in the RADSETUP directory.
2. First, the IBM Installation Manager installs. You will have to respond to a series of screens to complete this installation. If you already have the Installation Manager, the installation wizard checks for updates, and you are asked to update to the latest release. When the installation process is complete, you must restart the Installation Manager.
3. When the Installation Manager starts, click **Install**.

   Select Rational Application Developer 7.5.4 and WebSphere Application Server test environment 7.0 to install. Complete the installation following the wizard panels.

   Take the defaults for the packages to install. Do *not* install the Feature Packs to the test environment at this time. You need to go back through the update process later to bring the test environment up to WebSphere Application Server 7.0.0.7 before installing the Feature Packs. Installing them now gives you a lower code level than what is currently available.

   Clear the option to create a profile. Creating a profile at this stage gives you an application server that is not augmented for the Feature Packs.

   The same goes for any additional tools that you think might be needed for the Feature Packs. Delay the installation, because you need to update to 7.5.5 first. If you find that you are missing something later, you can go back through the installation process and install it.
4. Select **Update** from the Installation Manager main menu. If you do not have the latest level of the Installation Manager, you will be prompted to perform that update first.
5. After you update the Installation Manager, select **Update** again to update to Rational Application Developer 7.5.5 and WebSphere Application Server 7.0.0.7.

   When you select to update the WebSphere Application Server test environment to 7.0.0.7, select the option to install the Feature Pack for SCA. The Feature Pack for SCA has an additional option, the Service Data Object feature. This feature is optional and requires the Feature Pack for XML to be installed as a prerequisite. (We did not use the Service Data Object feature with the examples in this paper.).

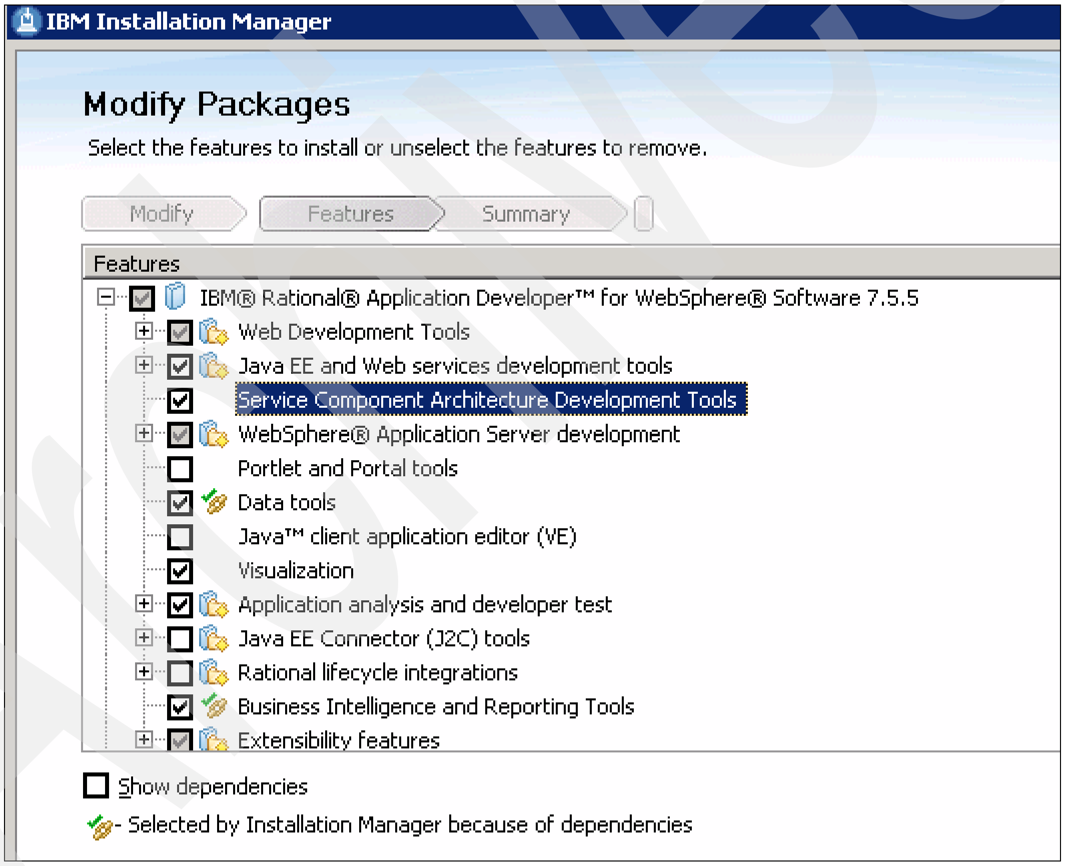
   Allow the update process to create an application server profile.

Note: If the Installation Manager does not find the WebSphere Application Server or Feature Pack updates when it scans for available updates, be sure that you have the following repositories defined to the Installation Manager (select **File** → **Preferences** → **Repositories**). When you add each repository, you are prompted to provide an IBM user name and password. Make sure that you also look for firewall blocking from your system and allow the Installation Manager to access the Web.

- `http://public.dhe.ibm.com/software/websphere/repositories/repository.config`
- `https://www.ibm.com/software/rational/repositorymanager/repositories/websphere/repository.config`

If you have WebSphere Application Server 7.0.0.7 installed and did not install the Feature Pack for SCA, select the Installation Manager Modify option to install the Feature Packs (instead of selecting the Update option).

6. Select **Modify** to update Rational Application Developer to add the Service Component Architecture Development Tools package, see Figure 2-1.



Modify Packages

Select the features to install or unselect the features to remove.

Features

- ✓ IBM® Rational® Application Developer™ for WebSphere® Software 7.5.5
  - ✓ Web Development Tools
  - ✓ Java EE and Web services development tools
  - ✓ Service Component Architecture Development Tools
  - ✓ WebSphere® Application Server development
  - ☐ Portlet and Portal tools
  - ✓ Data tools
  - ☐ Java™ client application editor (VE)
  - ✓ Visualization
  - ✓ Application analysis and developer test
  - ☐ Java EE Connector (J2C) tools
  - ☐ Rational lifecycle integrations
  - ✓ Business Intelligence and Reporting Tools
  - ✓ Extensibility features

☐ Show dependencies

✓ - Selected by Installation Manager because of dependencies

Figure 2-1 Installation Manager Modify Packages

2.2 Systems with Rational Application Developer and WebSphere Application Server

In this section, we provide an overview of the process to use when you have Rational Application Developer (with or without the WebSphere Application Server V7 test environment) and a separate WebSphere Application Server installation on the same system. The general steps are as follows:

1. Use the process that we describe in 2.1, “On Rational Application Development systems” on page 14 to install Rational Application Developer 7.5.5 and the WebSphere Application Server V7.0.0.7 test environment.
2. Install and update the WebSphere Application Server environment:
   a. Install WebSphere Application Server from the installation disk using the `launchpad.exe` utility. Note that this is the traditional method for installing WebSphere. You will not use Installation Manager to install it.

   b. Download the latest fixes for WebSphere Application Server and the JDK (7.0.0.7 currently) and the latest Update Installer from the support site:

      `http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg27004980#ver70`

      Note: The key to successfully installing Fix Pack 7 as a prerequisite for the Feature Pack for SCA installation is to download and install *both* the WebSphere Application Server updates and the JDK updates. These are two separate installations.

   c. Install the WebSphere Application Server Update Installer.

   d. Install both WebSphere Application Server and the JDK fix packs using the Update Installer.
3. Open the Installation Manager, and select **Import** to import the WebSphere Application Server installation. Note that the Import button is available only when the Installation Manager detects a package that requires this feature.
4. Update the Installation Manager repositories (see the Note box on page 15).
5. Select the Installation Manager **Install** option, and install the Feature Pack for SCA.

2.3 On WebSphere Application Server systems (no Rational Application Developer installed)

This section describes the basic process when you have only WebSphere Application Server installed on the system (no Rational Application Developer or other product that uses the Installation Manager). The general steps are as follows:

1. Install WebSphere Application Server using the `launchpad.exe` utility. Note that this is the traditional method. You will not use the IBM Installation Manager for this process.
2. Download the latest fixes for WebSphere Application Server and the JDK (7.0.0.7 currently) and the latest Update Installer from the support site:

   `http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg27004980#ver70`

   Note: The key to successfully installing Fix Pack 7 as a prerequisite for the Feature Pack for SCA installation is to download and install *both* the WebSphere Application Server updates and the JDK updates. These are two separate installations.

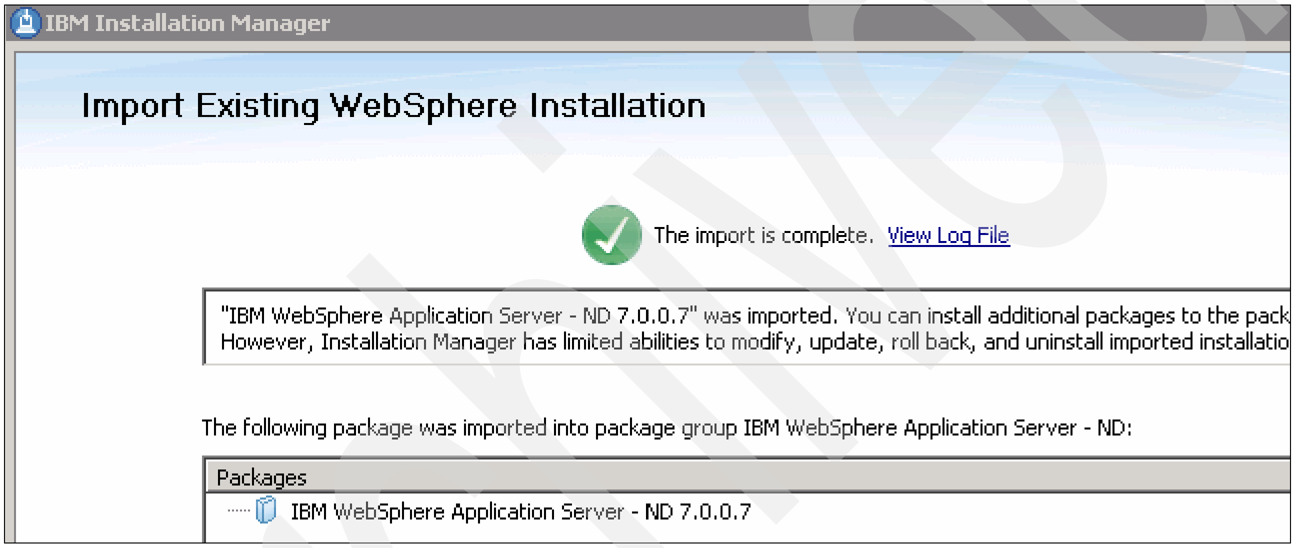
3. Install the WebSphere Application Server Update Installer.
4. Install both WebSphere Application Server and the JDK fix packs using the Update Installer.
5. Download the Installation Manager for WebSphere from the following Web site and install it:

   `http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg24023498`

6. Start the Installation Manager, and select **Import** to import the WebSphere Application Server installation. Then, follow these steps:

   a. Enter the location of the WebSphere installation.
   b. Select a location for the shared resources directory.
   c. Review the summary information, and select **Import**.

7. Follow the import panels until you receive the Import complete panel with a green check, as shown in Figure 2-2.


IBM Installation Manager
Import Existing WebSphere Installation
The import is complete. View Log File
Packages

"IBM WebSphere Application Server - ND 7.0.0.7" was imported. You can install additional packages to the pack
However, Installation Manager has limited abilities to modify, update, roll back, and uninstall imported installatio

The following package was imported into package group IBM WebSphere Application Server - ND:

IBM WebSphere Application Server - ND 7.0.0.7

Figure 2-2 WebSphere Installation Import complete

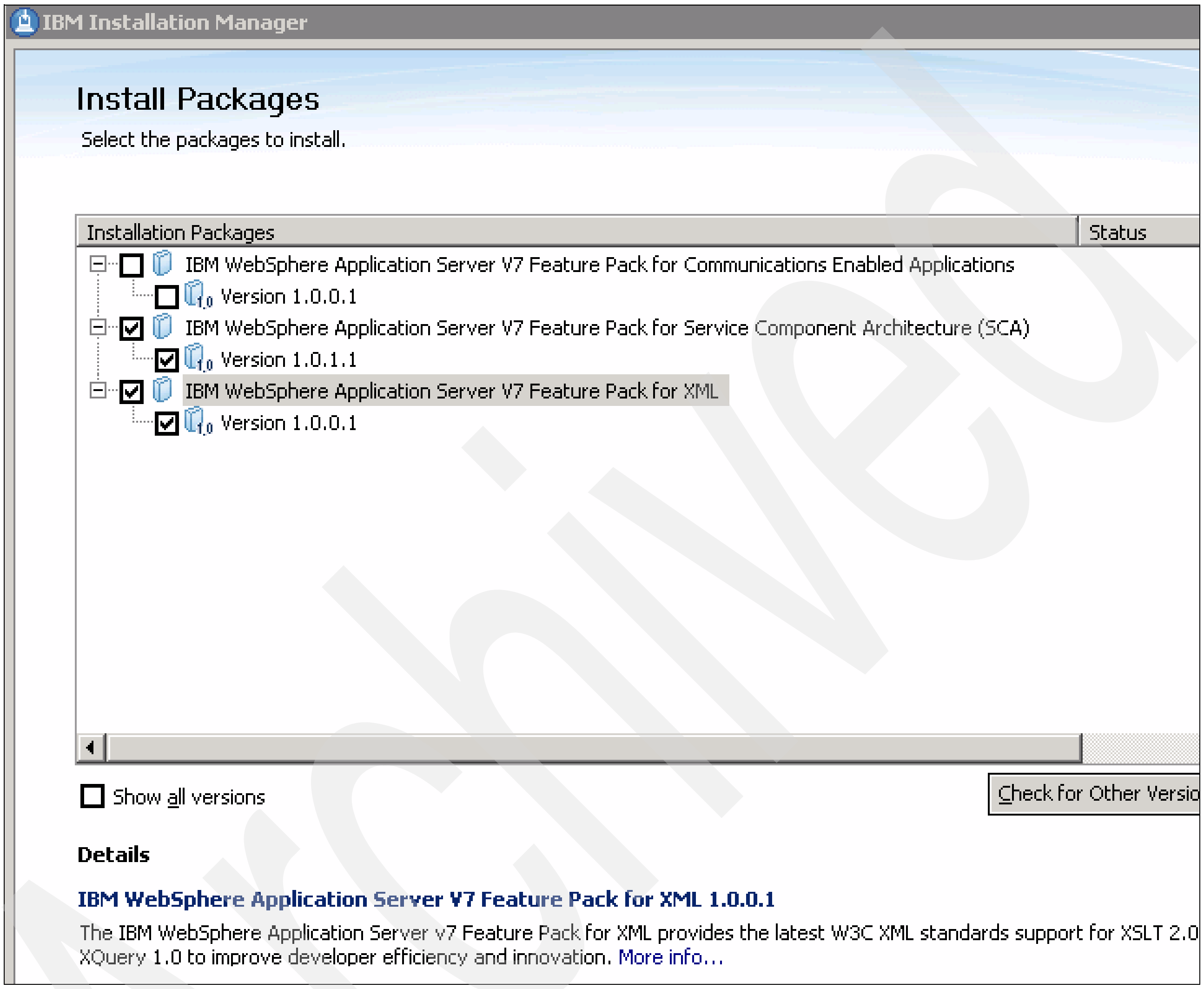
8. Update the Installation Manager repositories to include the WebSphere repository (select **File** → **Preferences** → **Repositories**)

   Note that accessing these repositories requires an IBM login. You are prompted for this login the first time that you attempt to access them:

   `http://public.dhe.ibm.com/software/websphere/repositories/repository.config`

9. Select **Install** in the Installation Manager to install the Feature Pack for SCA.

   Select the Feature Pack for SCA and any prerequisites that are called out (Figure 2-3). Note that if you are also installing the optional SDO feature, WebSphere Application Server V7 Feature Pack for XML is a prerequisite.



Install Packages

Select the packages to install.

Installation Packages	Status
☐ IBM WebSphere Application Server V7 Feature Pack for Communications Enabled Applications	
☐ Version 1.0.0.1	
☑ IBM WebSphere Application Server V7 Feature Pack for Service Component Architecture (SCA)	
☑ Version 1.0.1.1	
☑ IBM WebSphere Application Server V7 Feature Pack for XML	
☑ Version 1.0.0.1	

Details

IBM WebSphere Application Server V7 Feature Pack for XML 1.0.0.1

The IBM WebSphere Application Server v7 Feature Pack for XML provides the latest W3C XML standards support for XSLT 2.0
XQuery 1.0 to improve developer efficiency and innovation. More info...

Figure 2-3 Installation Packages

10. Follow the panels until you get the "Packages Installed" message, and then click **Finish**. At this point you can launch the Profile Management Tool to create your profiles, if needed.

3

Creating an SCA application

Rational Application Developer is the IDE of choice for developing applications for WebSphere Application Server. The SCA tool support makes Rational Application Developer ideal for developing SCA applications as well.

In this chapter, we use the MyBank sample that ships with the WebSphere Application Server Feature Pack for SCA to illustrate how to create a simple SCA application using Rational Application Developer 7.5.5.

3.1 Application overview

The MyBank application consists of the following elements:

- The *client* portion of the application has a JSP that provides the interface to the user and an SCA composite that calls a service. The service is also an SCA composite.
- Each *composite* has one component that is implemented with Java.
- The client component references the *service component* and uses a Web services binding.

Figure 3-1 shows the MyBank application.

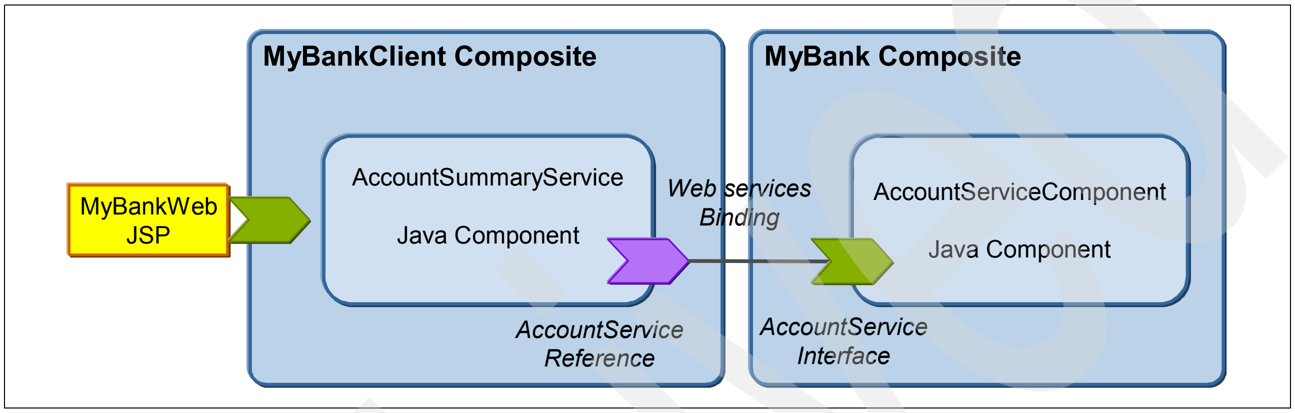


Figure 3-1 MyBank application overview

The remaining sections in this chapter show how each of the elements in the application are built.

3.2 Creating the MyBank composite

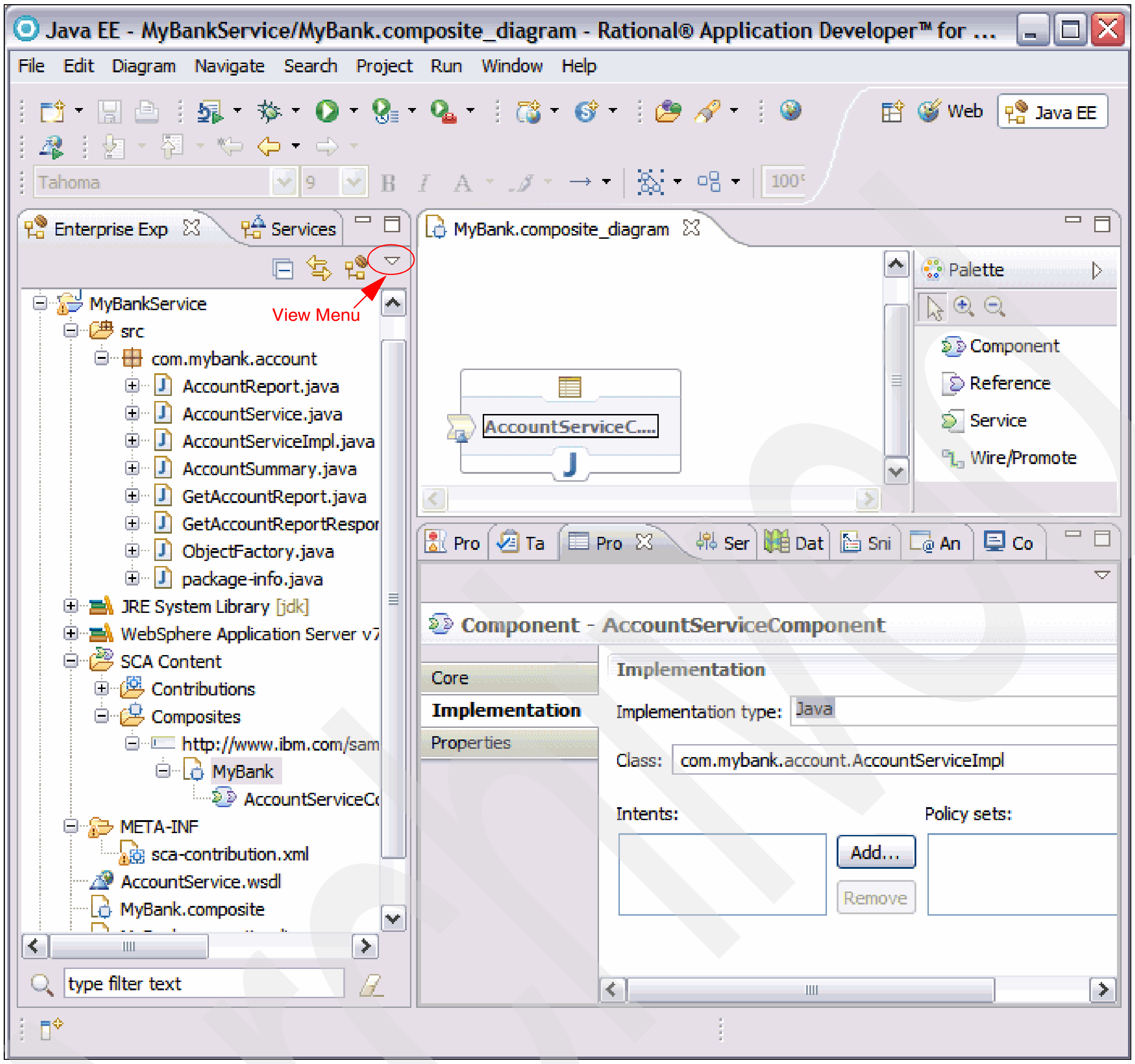
This section demonstrates how to create an SCA component using Rational Application Developer. As an example, we use the top-down development approach to create the new service from a WSDL file.

3.2.1 Overview of SCA service

Figure 3-2 shows the MyBank SCA application open in the Rational Application Developer workspace.

The MyBankService project and its files display on the left in the Enterprise Explorer view. To open any element in the project, double-click it in this view. The appropriate editor is used to open it in the workspace.

To the right, the MyBankComposite is open in the workspace (at the top), and the Properties view (at the bottom). Clicking any element in the Enterprise Explorer view or opening any element in the workspace displays its properties in the Properties view where you configure the elements.



Implementation

Implementation type: Java

Class: com.mybank.account.AccountServiceImpl

Figure 3-2 MyBank SCA application open in the workspace

The SCA Content section in the project structure is not a physical folder. It is a logical node that contains the composite. You will not find this folder in the compressed file when you export the WAR file.

Rational Application Developer presents a logical view of the SCA composites and contributions in this section. A composite file in the project is presented by namespace under the SCA Content node.

Tip to display the SCA files: The physical composite file is hidden unless you explicitly tell the workspace to show it. To change the setting so that the composite file is displayed in the Enterprise Explorer view:

1. Click **View menu** (the down arrow in the upper right section of the Enterprise Explorer view).
2. Select **Customize view** and then go to the **Filters** tab.
3. Clear **SCA Resources**.

The remainder of this sections shows how we created this structure using top-down development.

3.2.2 Create the SCA project

The first step is to create a new SCA project in the workspace:

1. Open Rational Application Developer, and switch to the Java EE perspective.
2. In the Enterprise Explorer view, right-click and select **New** → **SCA Project**.

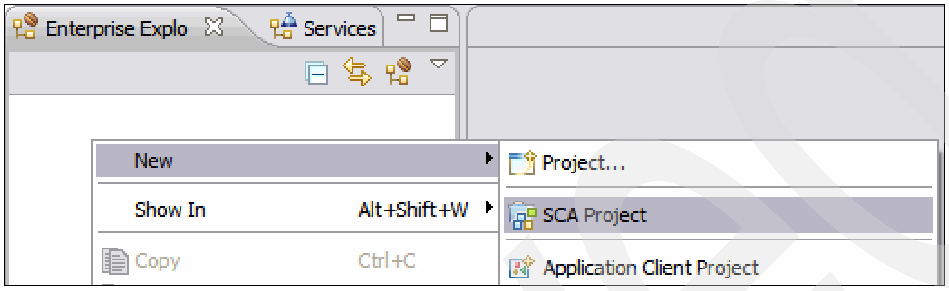
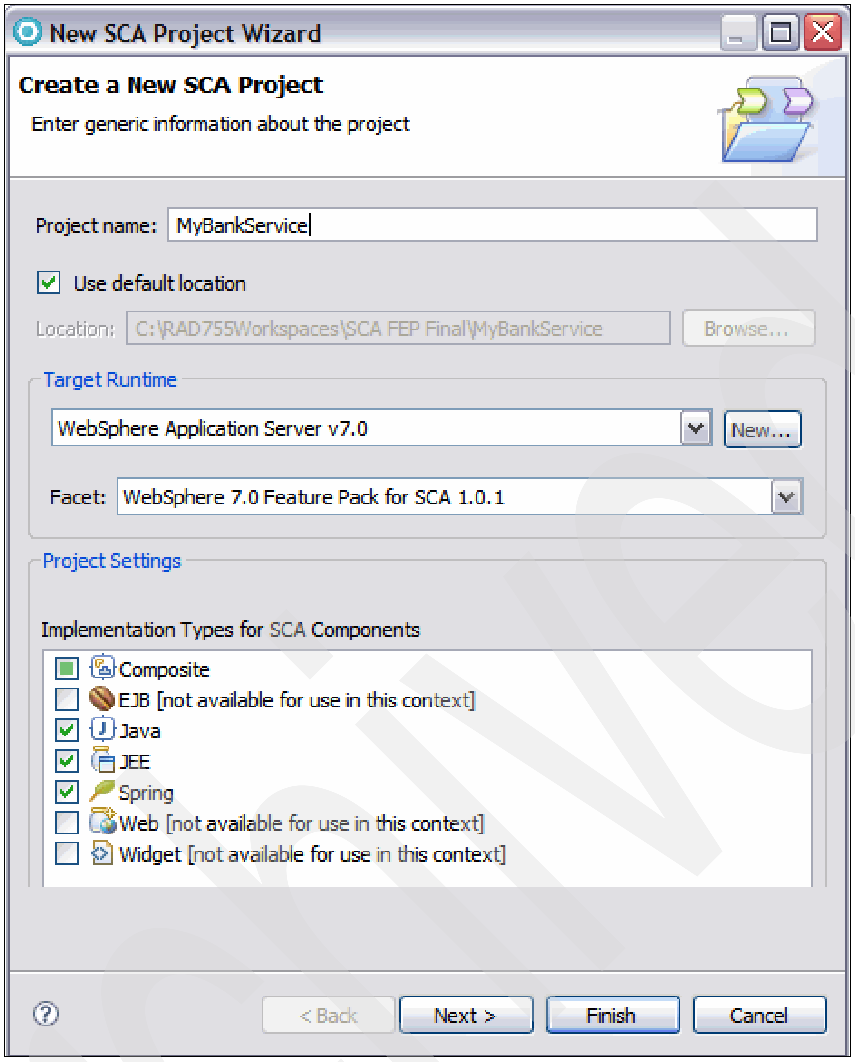


Figure 3-3 Create a new SCA project

3. In the first page of the New SCA Project Wizard, complete the following fields as shown in Figure 3-4 on page 23:
   a. Enter the project name as `MyBankService`.
   b. Ensure that the target run time is WebSphere Application Server v7.0.
   c. Ensure that the facet specifies the Feature Pack for SCA V1.0.1.
   d. Take the defaults for the implementation types.

> **Component implementation types:** Note that the implementation types that are available for the components that you will create in the project are determined when the SCA Project is created. The types vary depending on the facet that you choose. Figure 3-4 on page 23 shows the options that are available in the New SCA Project Wizard when the WebSphere 7.0 Feature Pack for SCA V1.0.1 facet is selected. The actual implementation for each component is determined when you create the component.



Create a New SCA Project

Enter generic information about the project

Project name: MyBankService

[✓] Use default location

Location: C:\RAD755Workspaces\SCA FEP Final\MyBankService

Target Runtime: WebSphere Application Server v7.0

Facet: WebSphere 7.0 Feature Pack for SCA 1.0.1

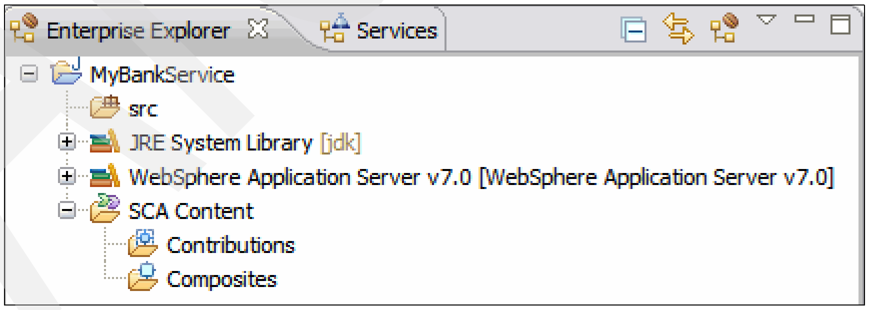
Project Settings

Implementation Types for SCA Components

- [■] Composite
- [ ] EJB [not available for use in this context]
- [✓] Java
- [✓] JEE
- [✓] Spring
- [ ] Web [not available for use in this context]
- [ ] Widget [not available for use in this context]

Figure 3-4 Provide the project information

4. Click **Finish**. The new project is created in the workspace, as shown in Figure 3-5.



```
MyBankService
    src
    JRE System Library [jdk]
    WebSphere Application Server v7.0 [WebSphere Application Server v7.0]
    SCA Content
        Contributions
        Composites
```

Figure 3-5 MyBankService project

3.2.3 Create an SCA composite

To create a composite from the Enterprise Explorer view:

1. Right-click the SCA Content/Composites folder. Select **New** → **SCA Composite** from the menu, as shown in Figure 3-6.

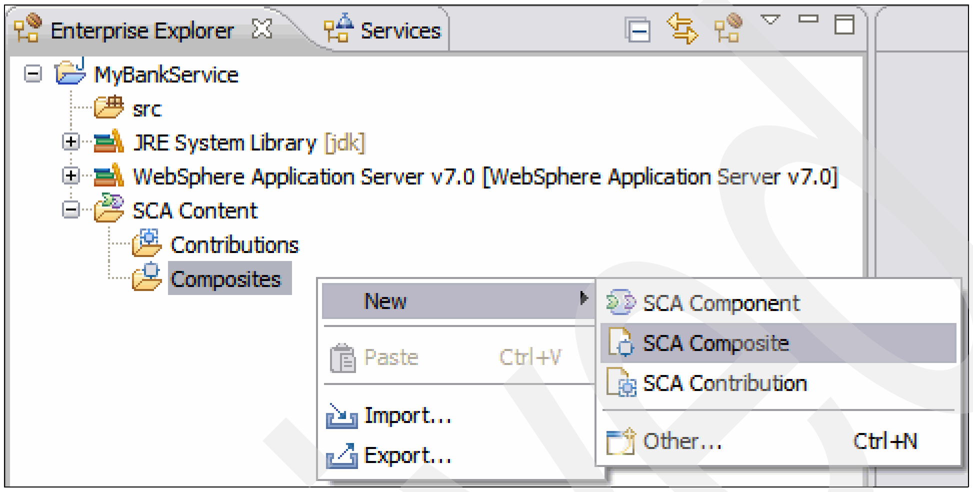
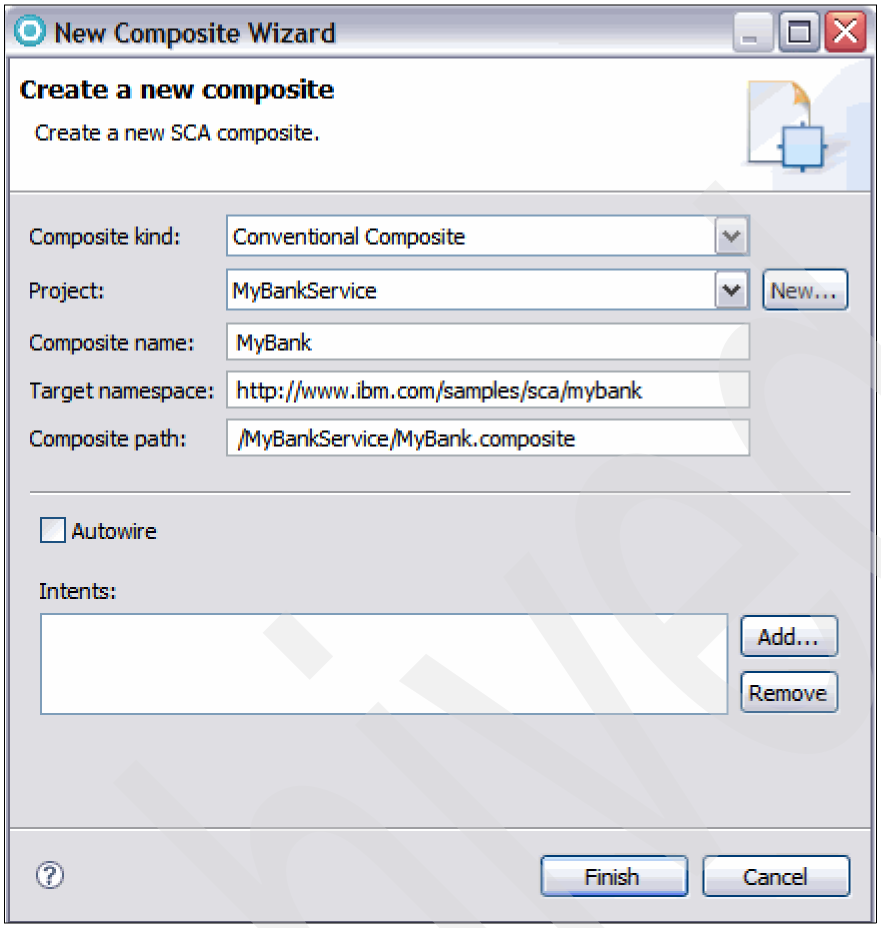


Figure 3-6 Create a new SCA composite

2. Complete the following fields in the New Composite Wizard, as shown in Figure 3-7 on page 25:
   a. Composite kind is Conventional Composite (the default).
   b. Ensure that the correct SCA project is selected.
   c. Enter the name for the composite as `MyBank`. This name is also the asset name when you deploy the composite to WebSphere Application Server.
   d. Enter the following target namespace for the composite:

      `http://www.ibm.com/samples/sca/mybank`



Create a new composite

Create a new SCA composite.

Composite kind: Conventional Composite

Project: MyBankService

Composite name: MyBank

Target namespace: http://www.ibm.com/samples/sca/mybank

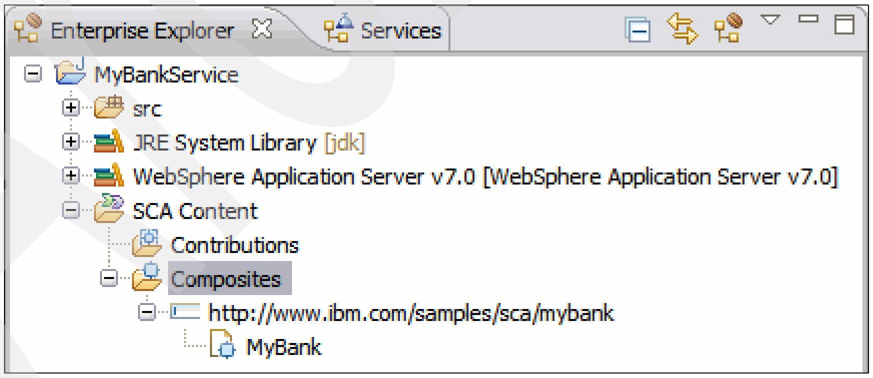
Composite path: /MyBankService/MyBank.composite

Autowire

Intents:

Figure 3-7 Provide the composite information

3. Click **Finish**. The new composite appears in the Enterprise Explorer view as shown in Figure 3-8.



```
MyBankService
  src
  JRE System Library [jdk]
  WebSphere Application Server v7.0 [WebSphere Application Server v7.0]
  SCA Content
    Contributions
    Composites
      http://www.ibm.com/samples/sca/mybank
        MyBank
```

Figure 3-8 New MyBank composite

3.2.4 Create a service WSDL file

For top-down development, add an existing WSDL file to the project or create a new WSDL file.

In this example, we used an existing WSDL file called `AccountService.wsdl` that defines a service called *AccountService*. We added the WSDL file to the project by simply dragging and dropping the file from a Windows® Internet Explorer window into the project. After you have the WSDL file, you can double-click the file name in the Enterprise Explorer view to open it in the workspace using the WSDL editor. Figure 3-9 shows the graphical interface of the WSDL editor with the `AccountService.wsdl` file open.

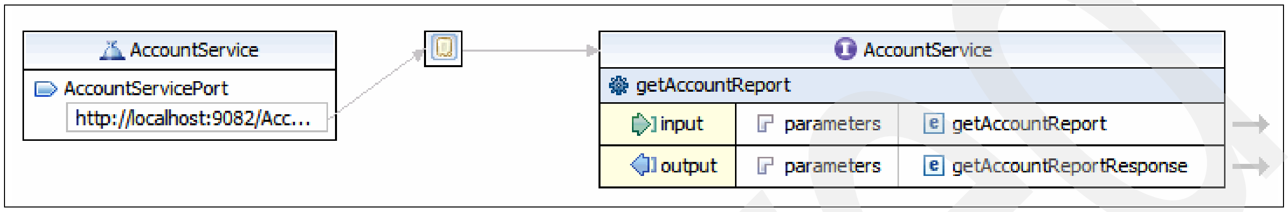


Figure 3-9 AccountService WSDL file open in the editor

You can also use the Rational Application Developer tools to create a new WSDL file in the workspace. Select **New** → **Web services** → **WSDL** to start the New WSDL File wizard.

3.2.5 Create a component (top-down)

To create a new component from the WSDL file:

1. Right-click the composite, and select **New** → **SCA Component**.

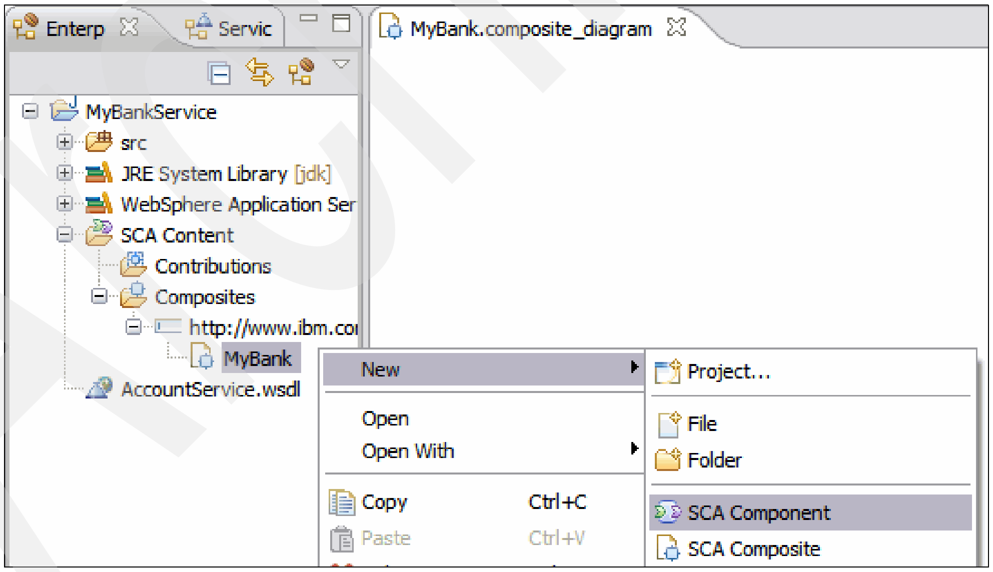
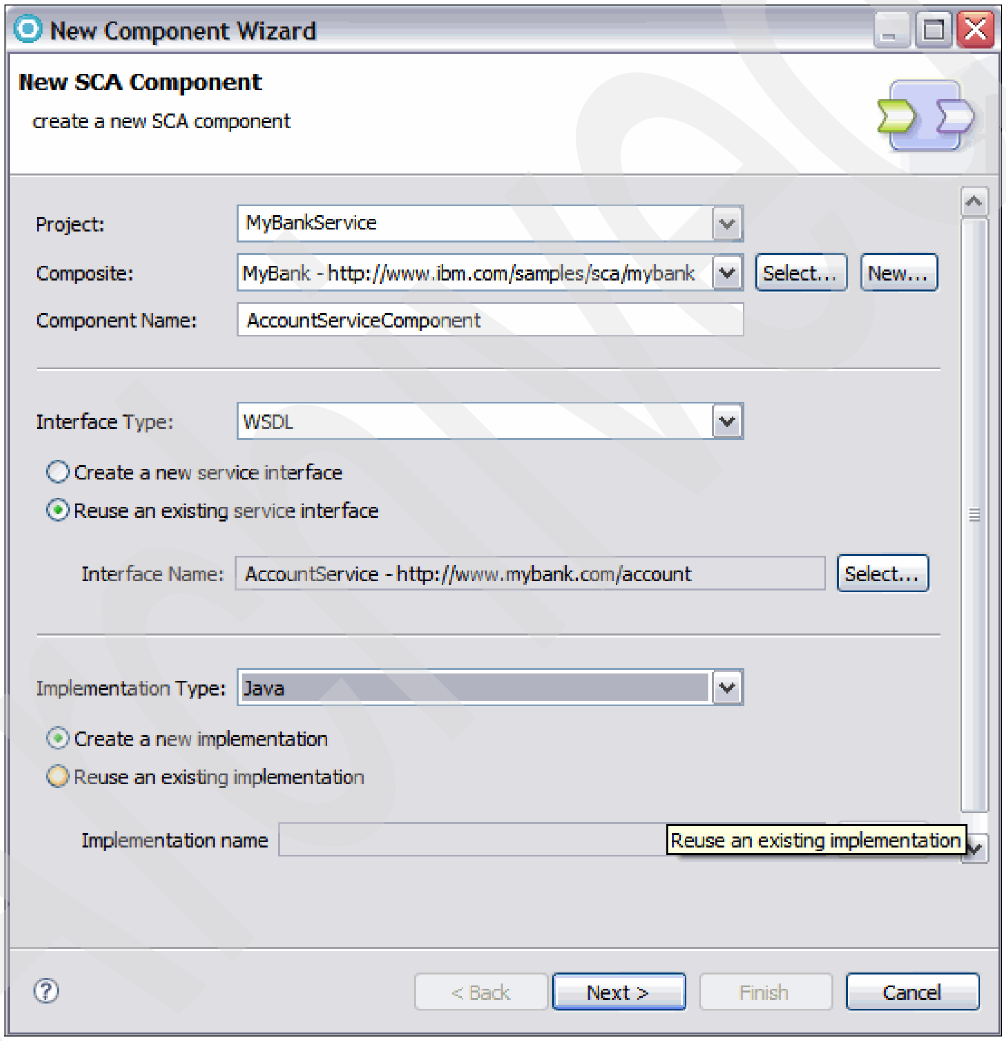


Figure 3-10 New SCA component, Step 1

2. Next, select how you want to create the component. In this top-down development scenario, we take the following options, as shown in Figure 3-11:
   a. Enter the name for the component as `AccountServiceComponent`.
   b. Select **WSDL** as the Interface Type.
   c. Select the “Reuse an existing service interface” option, and click **Select** to locate and select the WSDL file.
   d. Select the Implementation Type (**Java** in this case).
   e. Select the “Create a new implementation” option.
   f. Click **Next**.


New Component Wizard

New SCA Component

create a new SCA component

Project: MyBankService

Composite: MyBank - http://www.ibm.com/samples/sca/mybank

Component Name: AccountServiceComponent

Interface Type: WSDL

( ) Create a new service interface

(●) Reuse an existing service interface

Interface Name: AccountService - http://www.mybank.com/account

Implementation Type: Java

(●) Create a new implementation

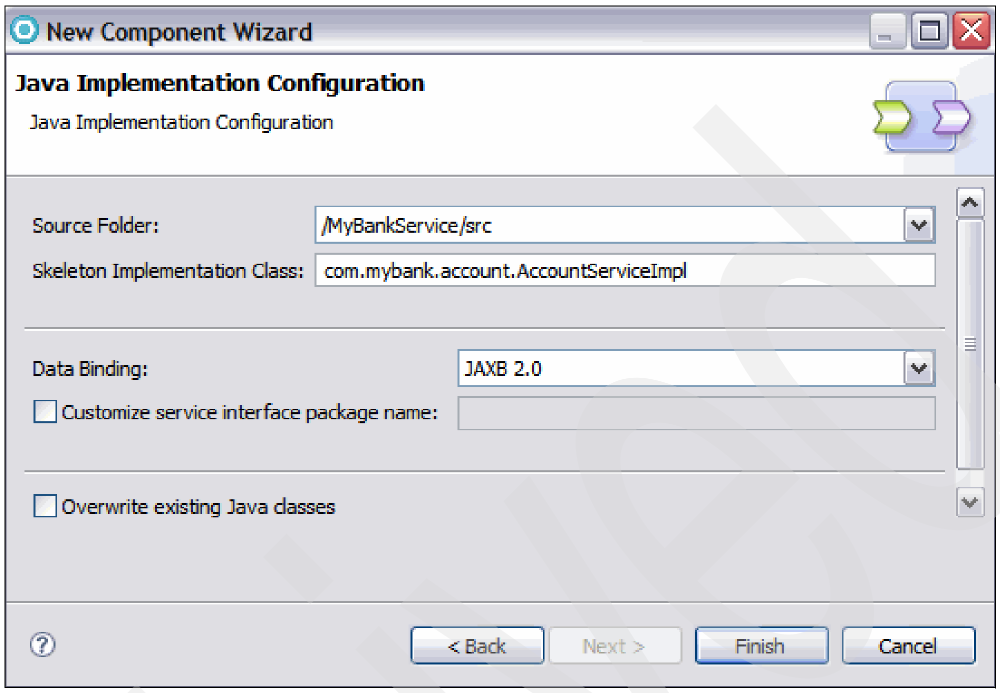
( ) Reuse an existing implementation

Implementation name

Reuse an existing implementation

Figure 3-11 New SCA component, Step 2

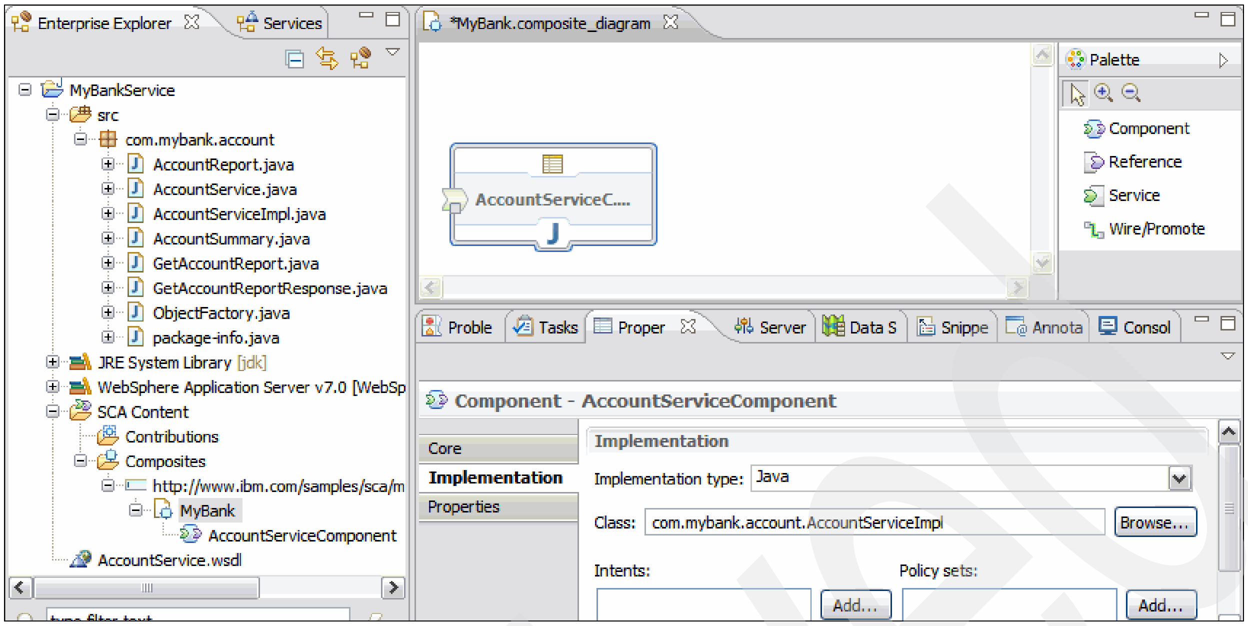
3. In most cases, you can take the defaults on the next window (Figure 3-12). Note that this window is where you specify SDO as the data binding if you are using SDO. Click **Finish**.


New Component Wizard
Java Implementation Configuration
Java Implementation Configuration
< Back
Next >
Finish
Cancel

Source Folder: /MyBankService/src
Skeleton Implementation Class: com.mybank.account.AccountServiceImpl
Data Binding: JAXB 2.0
☐ Customize service interface package name:
☐ Overwrite existing Java classes

Figure 3-12 New SCA component, Step 3

4. Double-click the composite. In the new component in the work area, note the following information, as shown in Figure 3-13 on page 29:
   - The service is added to the component and is denoted in the diagram as the arrow on the left border.
   - Java is set as the implementation type (note the letter “J” in the component diagram).
   - The implementation file is set for the component, as shown in the Properties view when you select the component on the diagram.
   - The new Java classes are created and placed in the `src` folder for the project (shown in the Enterprise Explorer view on the left).



```
MyBankService
  src
    com.mybank.account
      AccountReport.java
      AccountService.java
      AccountServiceImpl.java
      AccountSummary.java
      GetAccountReport.java
      GetAccountReportResponse.java
      ObjectFactory.java
      package-info.java
  JRE System Library [jdk]
  WebSphere Application Server v7.0 [WebSp
  SCA Content
    Contributions
    Composites
      http://www.ibm.com/samples/sca/m
        MyBank
          AccountServiceComponent
  AccountService.wsdl
```

Component - AccountServiceComponent

Implementation

Implementation type: Java

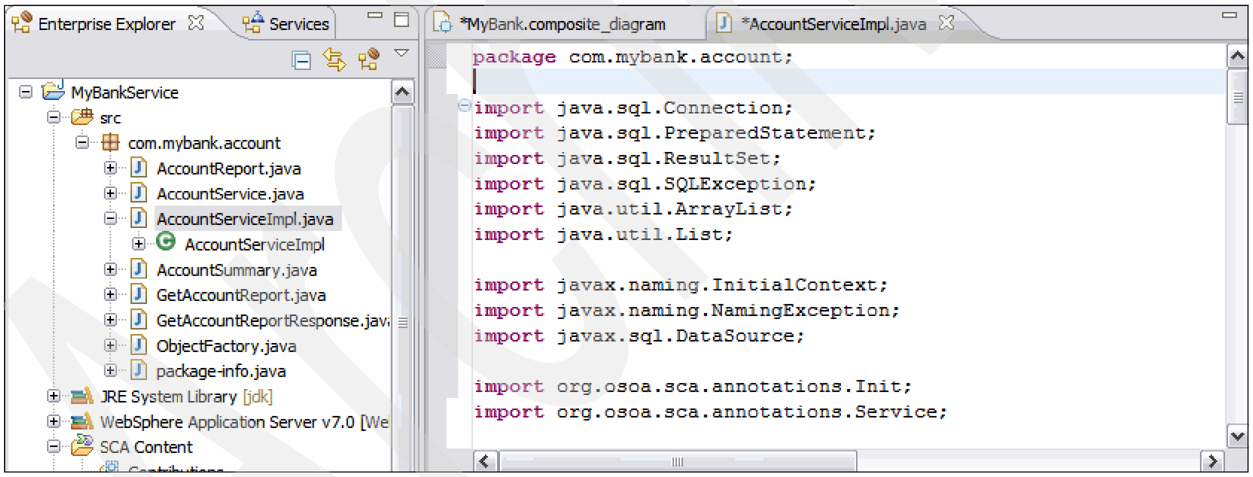
Class: com.mybank.account.AccountServiceImpl

Intents:

Policy sets:

Figure 3-13 New SCA component open in the editor

5. Implement the component by adding the Java code that will provide the business function to the implementation file, which is *AccountServiceImpl* in this case, as shown in Figure 3-14.



```
MyBankService
  src
    com.mybank.account
      AccountReport.java
      AccountService.java
      AccountServiceImpl.java
        AccountServiceImpl
      AccountSummary.java
      GetAccountReport.java
      GetAccountReportResponse.java
      ObjectFactory.java
      package-info.java
  JRE System Library [jdk]
  WebSphere Application Server v7.0 [We
  SCA Content
```

```
package com.mybank.account;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

import org.osoa.sca.annotations.Init;
import org.osoa.sca.annotations.Service;
```

Figure 3-14 Component implementation

Example 3-1 shows the code for this example.

Example 3-1 AccountServiceImpl

```
package com.mybank.account;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

import org.osoa.sca.annotations.Init;
import org.osoa.sca.annotations.Service;

import com.mybank.account.AccountReport;
import com.mybank.account.AccountService;
import com.mybank.account.AccountSummary;
import com.mybank.account.ObjectFactory;

@Service(AccountService.class)
public class AccountServiceImpl implements AccountService {

    private DataSource ds;

    @Init
    public void init() {
        initDataSource("mybank");
    }

   public com.mybank.account.AccountReport getAccountReport(int param0)  {
      final ObjectFactory objectFactory = new ObjectFactory();
      AccountReport accountReport = objectFactory.createAccountReport();
      accountReport.getAccountSummaries().addAll(getAccountSummaries(param0));
      return accountReport;
   }

   private List<AccountSummary> getAccountSummaries(int param0) {
       final ObjectFactory objectFactory = new ObjectFactory();
       ArrayList<AccountSummary> accountSummaries = new ArrayList<AccountSummary>();
        try {
            Connection conn = getConnection();
            PreparedStatement ps = conn.prepareStatement(
                           "SELECT accountNumber, accountID, accountType, balance FROM accounts 
where accountNumber = ?");
            ps.setInt(1, param0);
            ps.execute();
            ResultSet rs = ps.getResultSet();

            while ( rs.next() ) {
                AccountSummary summary = objectFactory.createAccountSummary();
                summary.setAccountNumber(rs.getInt(1));
                summary.setAccountID(rs.getString(2));
                summary.setAccountType(rs.getString(3));
                summary.setBalance(rs.getFloat(4));

                accountSummaries.add(summary);
            }
            conn.close();
        } catch (SQLException ex) {
            throw new RuntimeException(ex);
        }
        return accountSummaries;
    }

    private Connection getConnection() throws SQLException {
        Connection connection = null;
        try {
            connection = ds.getConnection();
            if (connection == null) {
                throw new RuntimeException(
                        "Could not obtain a Connection from DataSource");
            }
            connection.setAutoCommit(false);
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        return connection;
    }

    private void initDataSource(String database) {
        InitialContext ctx;
        try {
            ctx = new InitialContext();
        } catch (NamingException e) {
            throw new RuntimeException(e);
        }
        try {
            this.ds = (DataSource) ctx.lookup(database);
        } catch (NamingException e) {
            throw new RuntimeException(e);
        }
    }
}
```

3.2.6 Add bindings to the service interface

Although the service interface is added automatically to the component (because we created the component from the WSDL file), no bindings are attached yet. To add a binding:

1. Click the interface in the composite diagram. The interface is represented by the arrow on the left border of the component (*AccountServiceComponent*). Switch to the Properties view in the bottom pane.
2. Go to the Bindings tab, and click **Add**.

3. Select the binding type, and click **OK**. In this example, we use the *Webservice* binding, as shown in Figure 3-15.

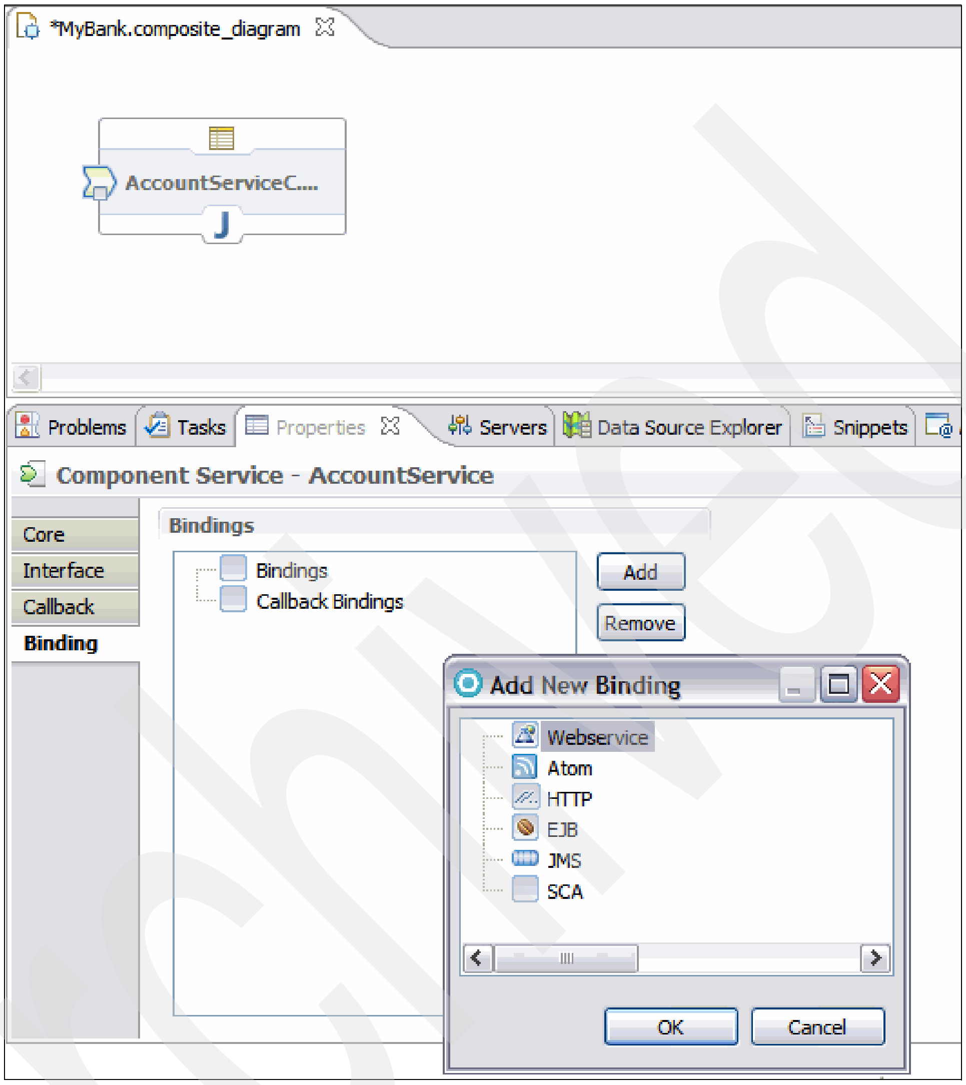
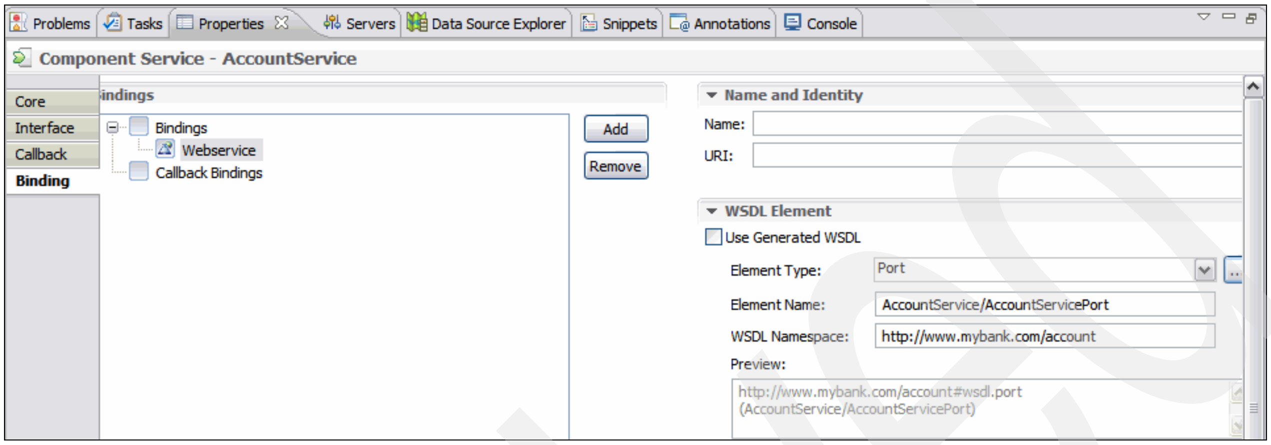


Figure 3-15 Add a Web service binding, Step 1

4. The new binding is added, and you can use the Properties view to provide the information that is required for the specific binding type. In this example, because we are adding a Web service binding, we clear the “Use Generated WSDL” option and specify the WSDL port element for the binding, as shown in Figure 3-16. From the Element Type drop-down menu, select **Port**. Then, the click the ellipses button (...) to the right of the Element Type field to browse for the specific port and to populate the Element Name field.



Name:

URI:

Use Generated WSDL

Element Type: Port

Element Name: AccountService/AccountServicePort

WSDL Namespace: http://www.mybank.com/account

Preview:
http://www.mybank.com/account#wsdl.port
(AccountService/AccountServicePort)

Figure 3-16 Add a Web service binding, Step 2

3.2.7 Add references for services to be invoked

You can use the composite diagram to add references to a component. Start by clicking the component in the diagram. A pop-up window appears with two icons. Clicking the left arrow icon adds a service to the component. Clicking the right arrow icon, a shown in Figure 3-17, adds a reference.

With the reference selected, switch to the Properties view to add the interface properties and the binding.

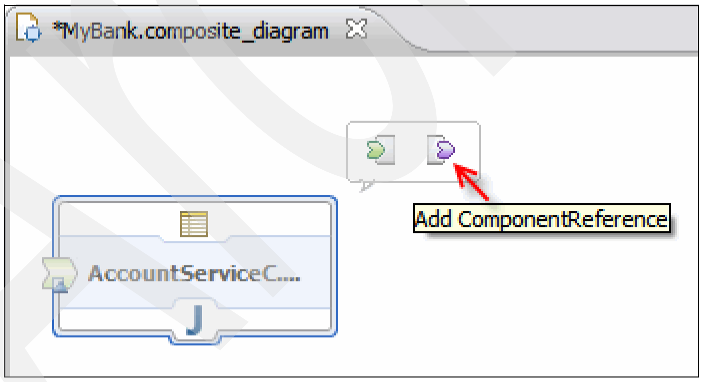


Figure 3-17 Add a reference

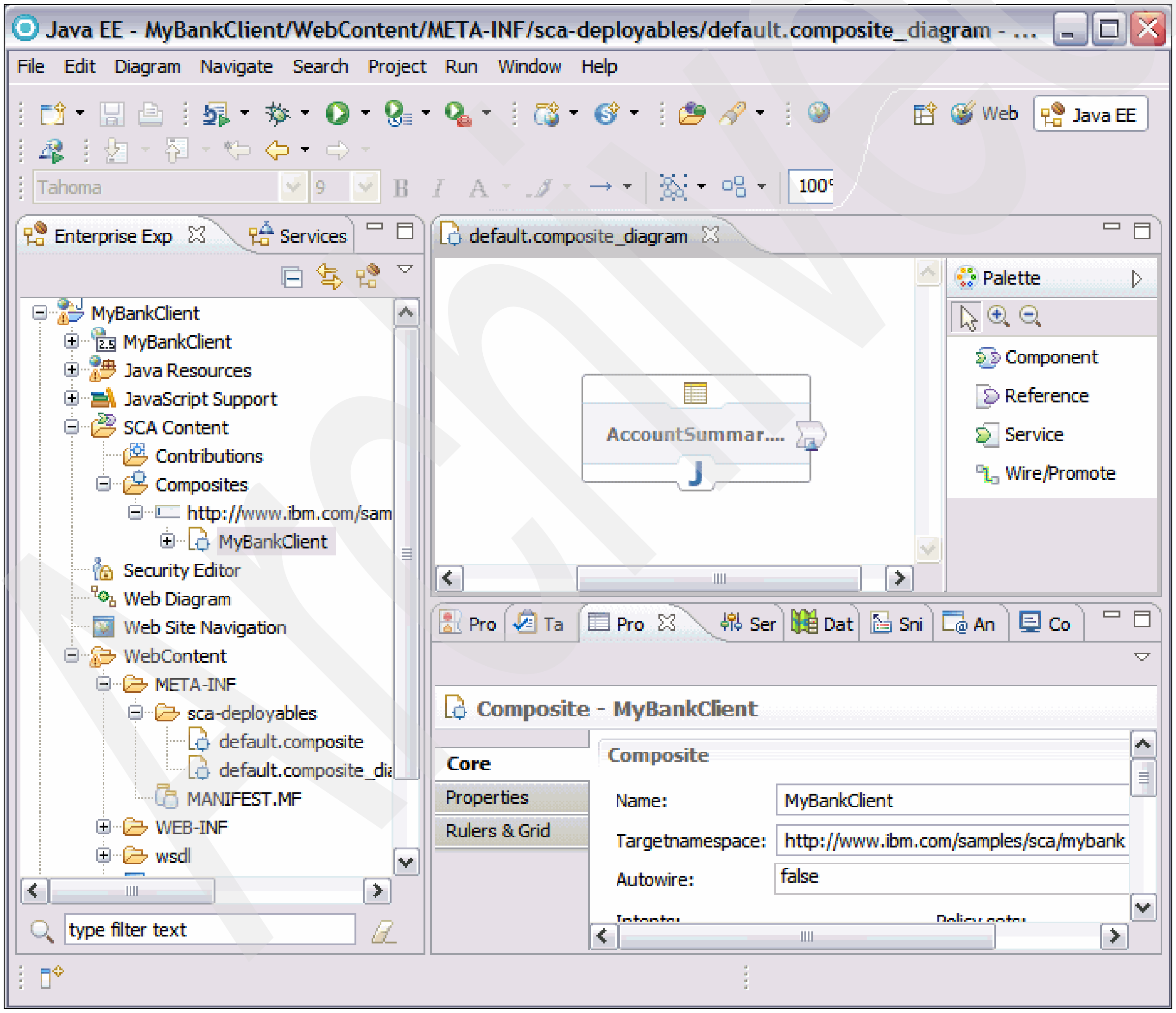
The AccountServiceComponent does not have any references, so we do not show the full process here. However, this service is referenced by the client portion of the application, which consists of a JSP and another composite. We describe how to add a reference later in “Add a reference to the composite” on page 45.

3.3 Creating the MyBankClient composite and JSP

The MyBank client application is a combination of a JSP and a composite that are used to invoke the MyBankService application. Because the client application is a combination of a Web module and SCA composite, it is created and packaged differently than the SCA service. The client is packaged as a WAR file that includes both the JSP and the SCA composite.

3.3.1 MyBankClient overview

Figure 3-18 shows the structure of the MyBankClient Web application in a Rational Application Developer workspace. Note that this is a Web application, but it also includes a composite. The component has a reference that allows it to invoke the MyBankService application. Note also that the client application is based on the `AccountService` WSDL file, taken from the MyBankService application.



Composite - MyBankClient

Composite

Name: MyBankClient

Targetnamespace: http://www.ibm.com/samples/sca/mybank

Autowire: false

Figure 3-18 MyBankClient project structure

Tip:

When you include SCA support in a WAR file, be aware of the following considerations:

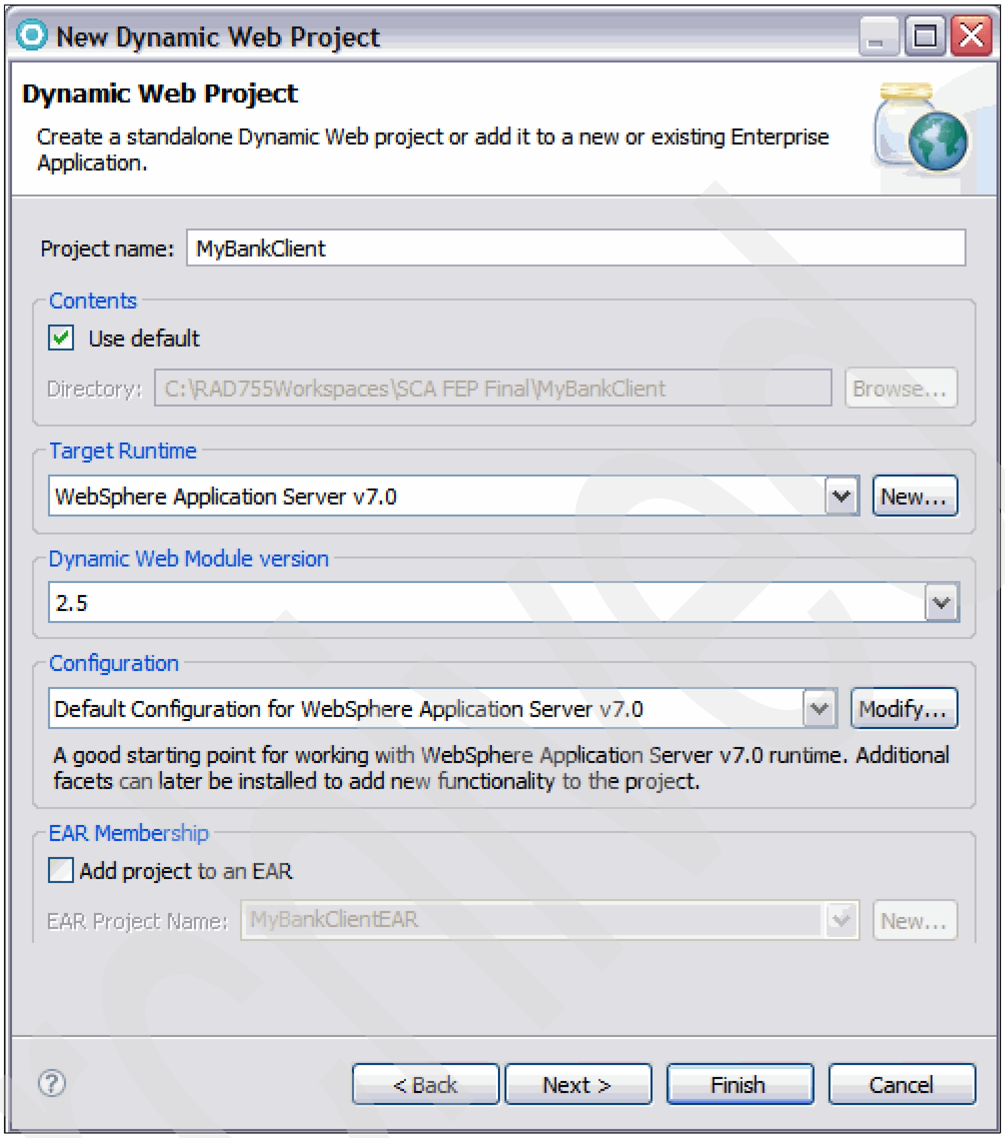
- When including SCA composites in a WAR file, there can be only one composite file, and it must be named `default.composite`. The composite does not need to be named `default`, but you need to ensure that the file name is correct before exporting the WAR file. If you create the composite file using the SCA tools, the name is *`composite_name`*`.composite`. You simply need to rename it.
- The `default.composite` file must reside in the `META-INF/sca-deployables` folder. The default is to store the file in the root directory of the workspace. You can set the default to the correct location by altering the preferences. Select **Window** → **Preferences** → **Service Component Architecture**. Change the default composite folder to `WebContent/META-INF/sca-deployables`.

For more information about the `default.composite` file and the `sca-deployables` directory, see the Note box on page 12.

3.3.2 Create a dynamic Web application

The project type for this combined application is a Web project. In the workspace, you create a new Web project as follows:

1. Right-click in the Enterprise Explorer view, and select **New** → **Other** → **Web** → **Dynamic Web project**. Click **Next**.
2. Enter the project name as `MyBankClient`, as shown in Figure 3-19 on page 36. In this example, we do not this Web module in an EAR file, although it can be. Click **Next**.


New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: MyBankClient

Contents

☑ Use default

Directory: C:\RAD755Workspaces\SCA FEP Final\MyBankClient

Target Runtime

WebSphere Application Server v7.0

Dynamic Web Module version

2.5

Configuration

Default Configuration for WebSphere Application Server v7.0

A good starting point for working with WebSphere Application Server v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

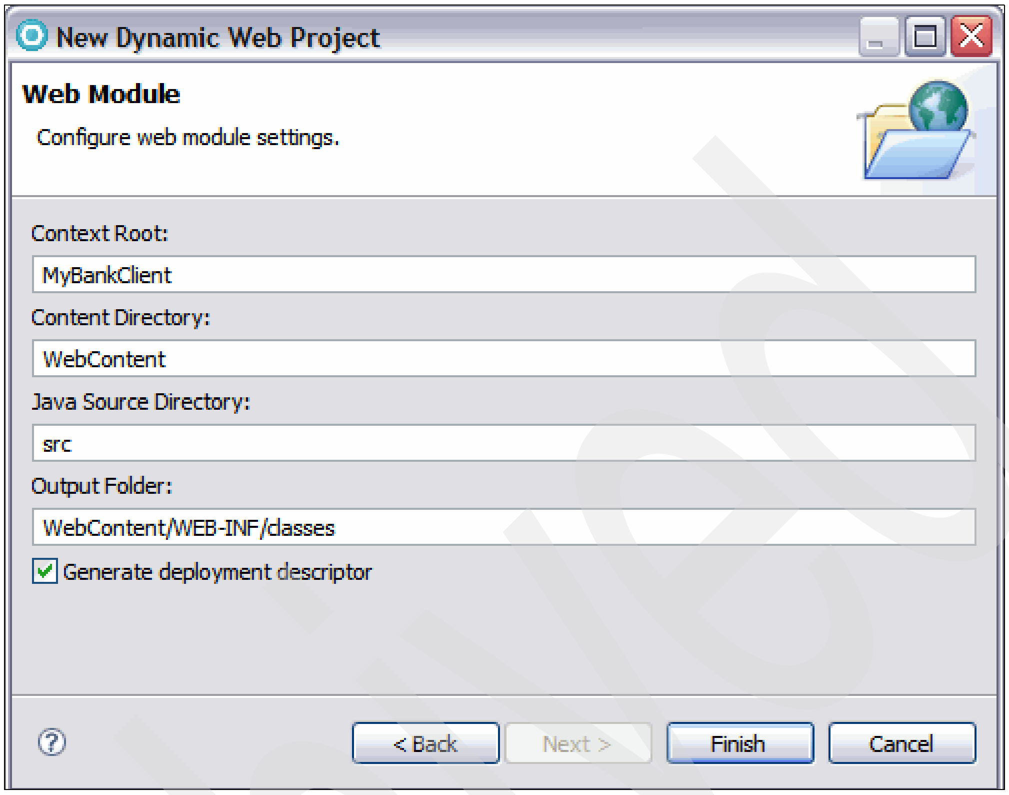
EAR Membership

☐ Add project to an EAR

EAR Project Name: MyBankClientEAR

Figure 3-19 Dynamic Web project settings

3. Provide a context root or accept the default as shown in Figure 3-20. Click **Finish**.


New Dynamic Web Project

Web Module

Configure web module settings.

Context Root: MyBankClient

Content Directory: WebContent

Java Source Directory: src

Output Folder: WebContent/WEB-INF/classes

✓ Generate deployment descriptor

Figure 3-20 Dynamic Web project settings, continued

The new project is created and displays in the Enterprise Explorer.

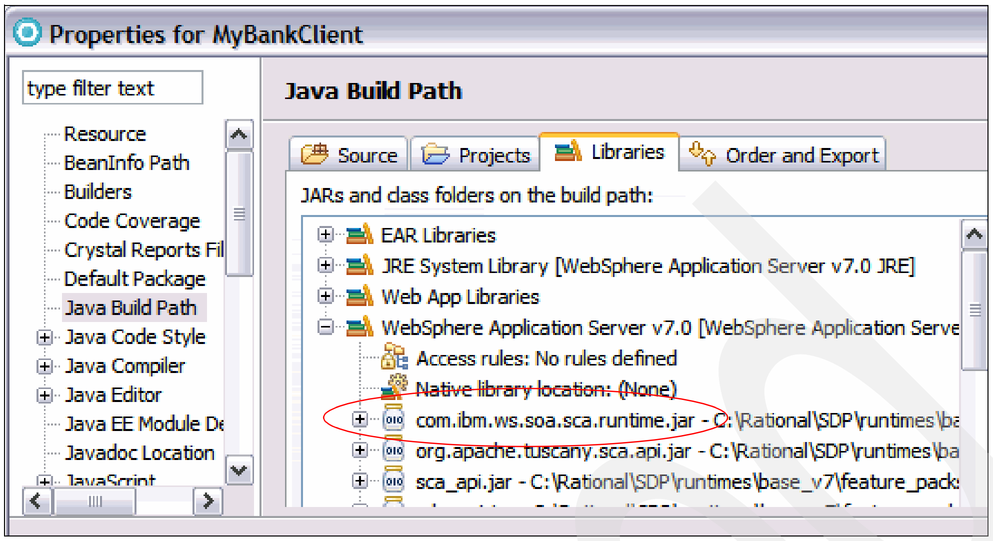
3.3.3 Prepare the environment for SCA

The client Web project will have both Web components (the JSP file) and SCA components. When the Web project was created, the run time libraries added automatically to the Java build path for the Web components. We need to also ensure that the run time libraries that are required for SCA are included in the build path. To do this, you need to add the following JAR file to the Java build path for the client project.

app_server_root/`feature_packs/sca/plugins/com.ibm.ws.soa.sca.runtime.jar`

Then, follow these steps:

1. In the Web project context menu, right-click the MyBankClient project, and select **Service Component Architecture** → **Add SCA support** to add the JAR file to the build path and to create a new folder in the project for SCA content.
2. Verify that the file is added to the build path, as shown in Figure 3-21 on page 38. Right-click the MyBankClient project and select **Properties**. Click the Libraries tab, and look for the SCA runtime JAR file.


Properties for MyBankClient

type filter text

- Resource
- BeanInfo Path
- Builders
- Code Coverage
- Crystal Reports Fil
- Default Package
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Java EE Module De
- Javadoc Location
- JavaScript

Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

- EAR Libraries
- JRE System Library [WebSphere Application Server v7.0 JRE]
- Web App Libraries
- WebSphere Application Server v7.0 [WebSphere Application Serve
  - Access rules: No rules defined
  - ~~Native library location: (None)~~
  - com.ibm.ws.soa.sca.runtime.jar - C:\Rational\SDP\runtimes\ba
  - org.apache.tuscany.sca.api.jar - C:\Rational\SDP\runtimes\ba
  - sca_api.jar - C:\Rational\SDP\runtimes\base_v7\feature_pack

Figure 3-21 Java Build Path settings

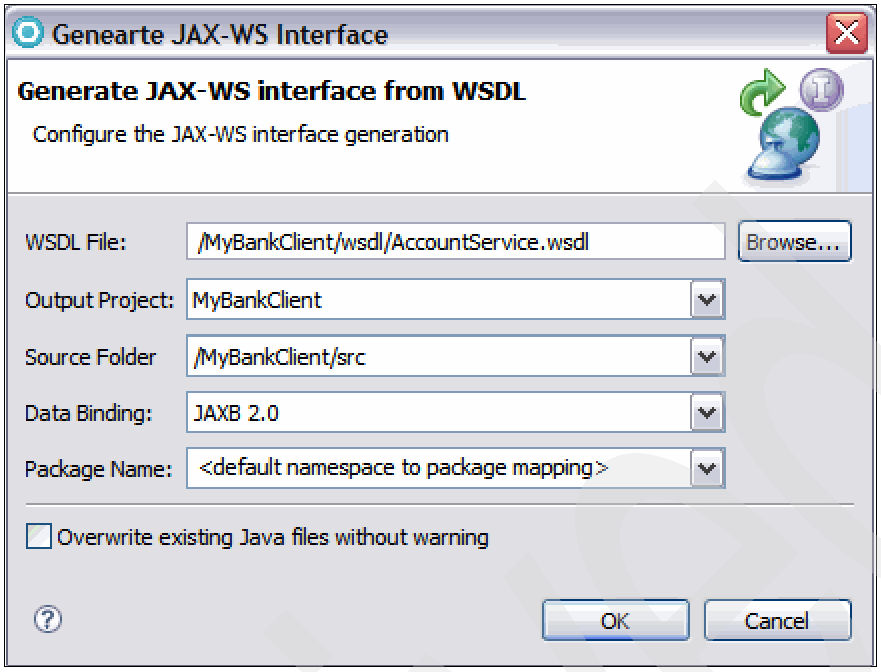
If you fail to do this for MyBankClient, you get the following errors:

```
composite context cannot be resolved to a type
```

3.3.4 Add the WSDL file to the project

The WSDL file provides the information that is required to access the MyBankService. To add the WSDL file to the project:

1. Copy the `AccountService.wsdl` file from the MyBankService project to the MyBankClient project. In this example, we create a new `wsdl` folder in the Web Content folder and place the WSDL file there.
2. Right-click the WSDL file, and select **Service Component Architecture** → **Generate JAX-WS interface**. Take the defaults shown in Figure 3-22 on page 39, and click **OK**.


Genearte JAX-WS Interface

Generate JAX-WS interface from WSDL

Configure the JAX-WS interface generation

WSDL File: /MyBankClient/wsdl/AccountService.wsdl

Output Project: MyBankClient

Source Folder /MyBankClient/src

Data Binding: JAXB 2.0

Package Name: <default namespace to package mapping>

☐ Overwrite existing Java files without warning

Figure 3-22 Generate the JAX-WS classes from the WSDL

These steps add the `com.mybank.account` package and classes to the `src` folder, as shown in Figure 3-23.

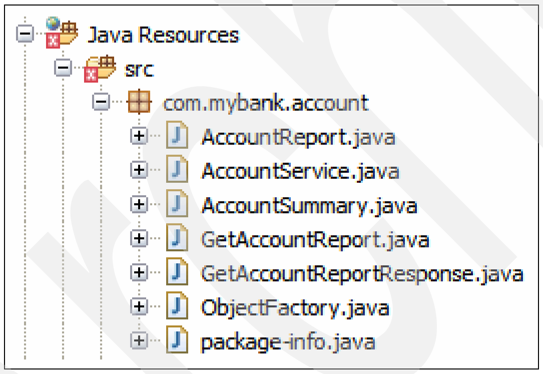
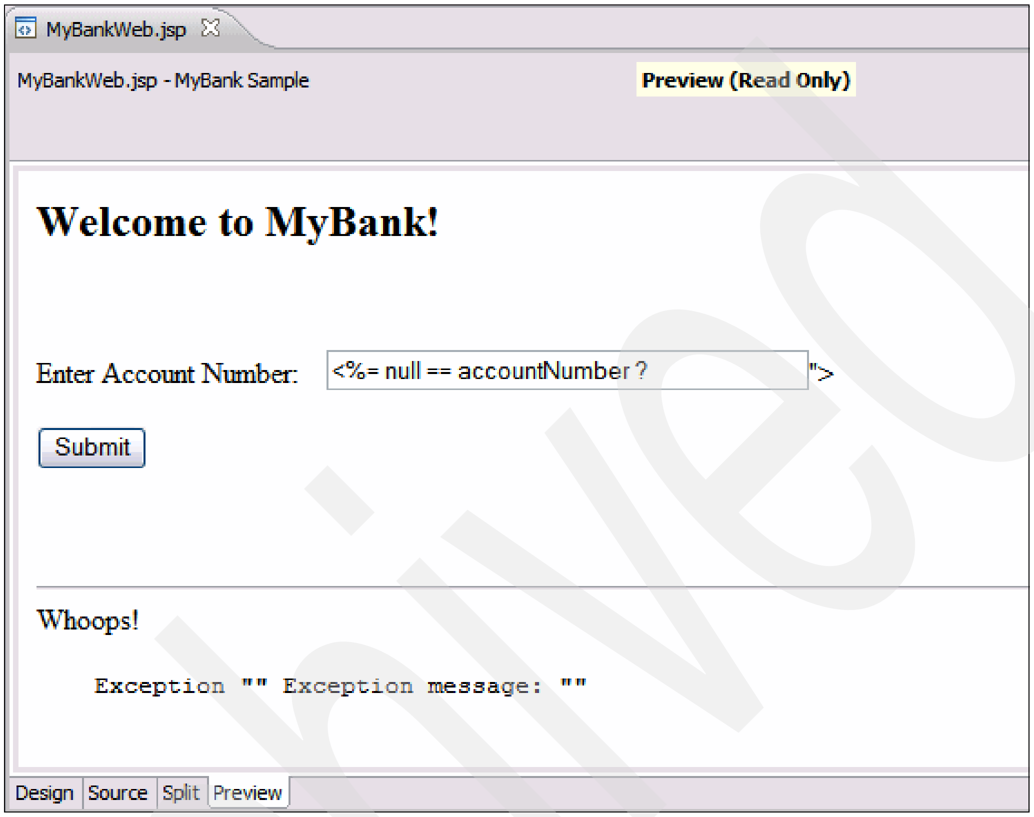

```
Java Resources
  src
    com.mybank.account
      AccountReport.java
      AccountService.java
      AccountSummary.java
      GetAccountReport.java
      GetAccountReportResponse.java
      ObjectFactory.java
      package-info.java
```

Figure 3-23 Generated classes

3.3.5 Code the JSP customer interface

Create a `MyBankWeb.jsp` JSP file in the Web content folder to provide the user interface. The new user interface look similar to that shown in Figure 3-24.



Welcome to MyBank!

Enter Account Number: <%= null == accountNumber ? ">

Whoops!

```
Exception "" Exception message: ""
```

Figure 3-24 JSP page

Example 3-2 shows the JSP code.

Example 3-2 MyBankWeb.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML lang="en">
<HEAD>
    <%@ page language="java" contentType="text/html; charset=ISO-8859-1"
             pageEncoding="ISO-8859-1" session="true" autoFlush="true"
             isThreadSafe="true" isErrorPage="false"
             import="samples.mybank.AccountSummaryService"
             import="java.util.List"
             import="java.util.Iterator"
             import="java.io.PrintWriter"
             import="java.io.StringWriter"
             import="com.ibm.websphere.sca.context.CurrentCompositeContext"
             import="com.ibm.websphere.sca.context.CompositeContext"
            %>
    <META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <META name="GENERATOR" content="IBM Software Development Platform">
    <TITLE>MyBank Sample </TITLE>
    <%
        String accountNumber = request.getParameter("accountNumber");
    %>
</HEAD>
<BODY>
<H2>Welcome to MyBank!</H2>
<form action="MyBankWeb.jsp" method="get"><label for="i1" accesskey="E">Enter Account
Number:&nbsp;&nbsp;&nbsp; </label>
    <INPUT type="text" id="i1" name="accountNumber" size="40" maxlength="220"
           value="<%= null == accountNumber ? "100" : accountNumber %>">
<BR>
     <INPUT type="submit" name="submit" value="Submit"></FORM>
<BR>
<%!
   private static AccountSummaryService accountService = null;
   private CompositeContext compositeContext = null;
   public void jspInit() {
      if (accountService == null) {
          compositeContext = CurrentCompositeContext.getContext();
          accountService =
(AccountSummaryService)compositeContext.getService(AccountSummaryService.class,
"AccountSummaryService");
        }
   }
%>
<%
    try {
        if (null != accountNumber) {
            List<String> values = accountService.getAccountSummary(accountNumber);
            for (String summary : values) {
                %> <P> <%=summary%> <BR>
<%
            }
%>
<HR>
    <%}
}catch(Exception e){
                  e.printStackTrace();
                  StringWriter sw= new StringWriter();
                  PrintWriter pw= new PrintWriter(sw);
                  e.printStackTrace(pw);
                  pw.flush();
%>
    Whoops!<BR clear="all">
<PRE>
    Exception &quot;<%=e.getClass().getName()%>&quot; Exception message:
&quot;<%=e.getMessage()%>&quot;<BR clear="all">
    <%=sw.toString() %>
</PRE>
<%
    }%>
</BODY>
</HTML>
```

3.3.6 Code the client implementation that will invoke the service

Create a new package in the `Java Resources/src` folder called `samples.mybank`, and add the `AccountSummaryService` class, shown in Example 3-3, and the `AccountSummaryServiceImpl` class, shown in Example 3-4, to the `samples.mybank` package.

Example 3-3 AccountSummaryService class

```
package samples.mybank;
import java.util.List;
public interface AccountSummaryService {
         public List<String> getAccountSummary(String name);
}
```

Example 3-4 AccountSummaryServiceImpl class

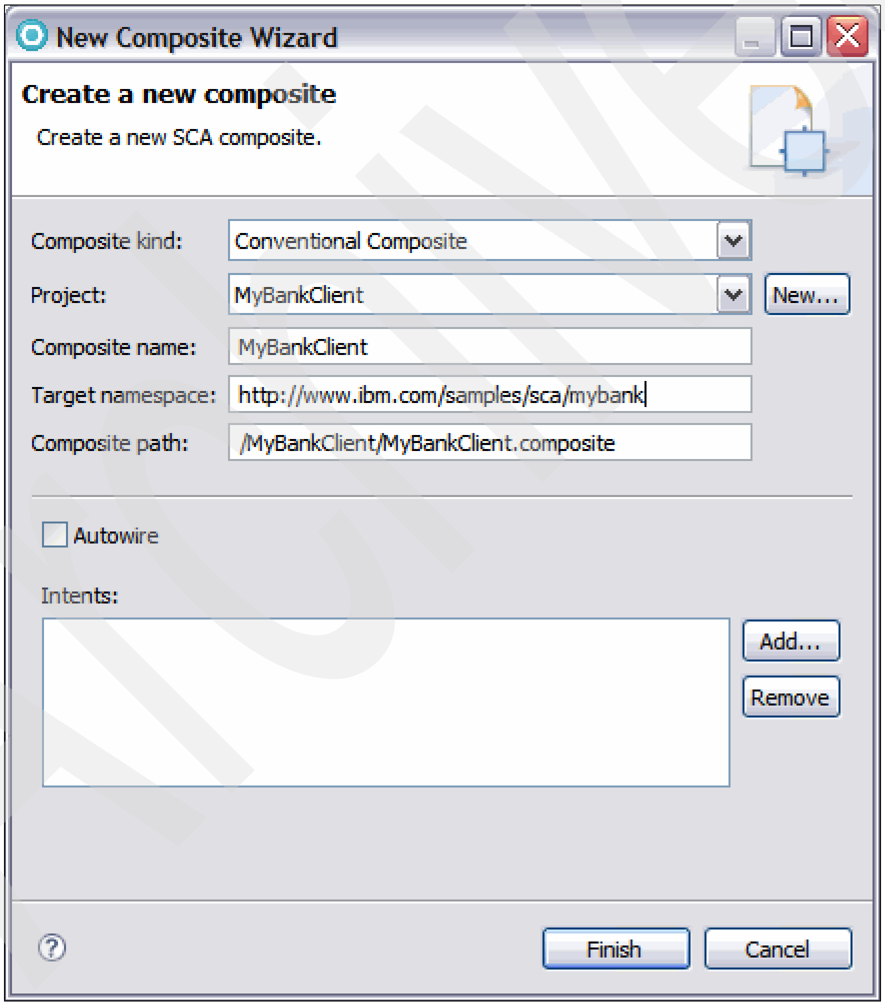
```
package samples.mybank;
import java.text.NumberFormat;
import java.util.Iterator;
import java.util.List;
import java.util.Locale;
import java.util.Vector;
import org.osoa.sca.annotations.Reference;
import org.osoa.sca.annotations.Service;
import com.mybank.account.AccountReport;
import com.mybank.account.AccountService;
import com.mybank.account.AccountSummary;
@Service(AccountSummaryService.class)
public class AccountSummaryServiceImpl implements AccountSummaryService {
     private AccountService accountService;
       @Reference
       public void setAccountService(AccountService accountService) {
                    this.accountService = accountService;
       }
       public List<String> getAccountSummary(String name) {
              String msg = "The account summary is: \n" ;
              List<String> summaryStrings = new Vector<String>();

              int accountNumber = Integer.parseInt(name);
              AccountReport report =
               accountService.getAccountReport(accountNumber);

              List<AccountSummary> summaryList = report.getAccountSummaries();
                   for(Iterator<AccountSummary> i =
                      summaryList.iterator();i.hasNext();) {
                               AccountSummary summary = i.next();
                               //Format balance
                               Float balance = summary.getBalance();
                               NumberFormat n =
                                 NumberFormat.getCurrencyInstance(Locale.US);
                               String balanceString = n.format(balance);
                               summaryStrings.add("ACCOUNT ID (" +
                                   summary.getAccountID() + ")
                                  ACCOUNT TYPE {" + summary.getAccountType() + ")
                                 BALANCE (" + balanceString + ")");
                     }
```

```
                        return summaryStrings;
                }
        }
```

3.3.7 Add the composite definition for the client

To add the composite definition for the client:

1. Create a new composite called `MyBankClient`, as shown in Figure 3-25.
   - Specify the target namespace as:
     `http://www.ibm.com/samples/sca/mybank`
   - The composite path must be:
     `/MyBankClient/WebContent/META-INF/sca-deployables/default.composite`
2. Click **Finish**.



New Composite Wizard

Create a new composite

Create a new SCA composite.

Composite kind: Conventional Composite

Project: MyBankClient

Composite name: MyBankClient

Target namespace: http://www.ibm.com/samples/sca/mybank

Composite path: /MyBankClient/MyBankClient.composite

Autowire

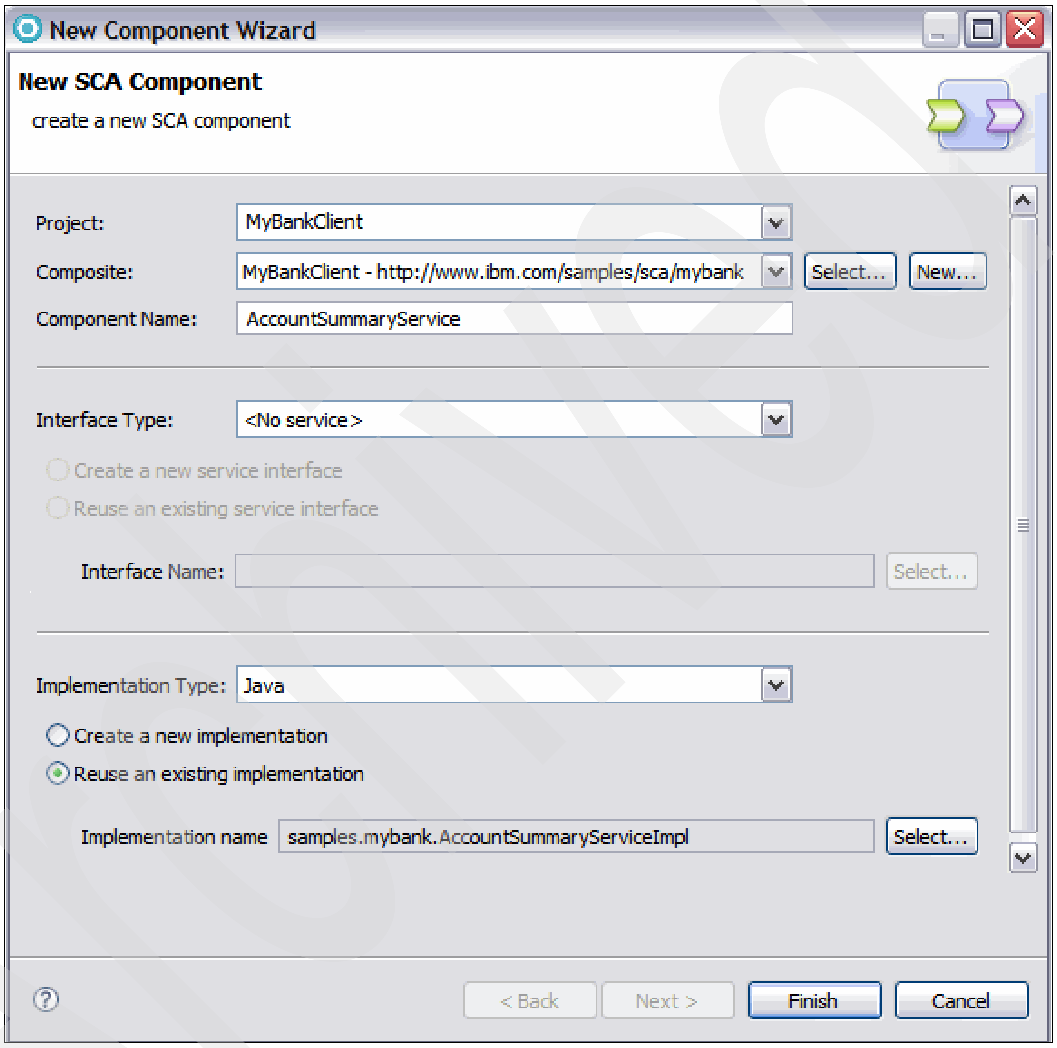
Intents:

Figure 3-25 Create the MyBankClient composite

Create a new component

To create a new component:

1. Create a new component called `AccountSummaryService`, as shown in Figure 3-26.
   - Use the existing `AccountSummaryServiceImpl` file to create the component.
   - Specify **<No Service>** as the interface type.
2. Click **Finish**.



Project: MyBankClient

Composite: MyBankClient - http://www.ibm.com/samples/sca/mybank

Component Name: AccountSummaryService

Interface Type: <No service>

Interface Name:

Implementation Type: Java

( ) Create a new implementation
[✓ Reuse an existing implementation]

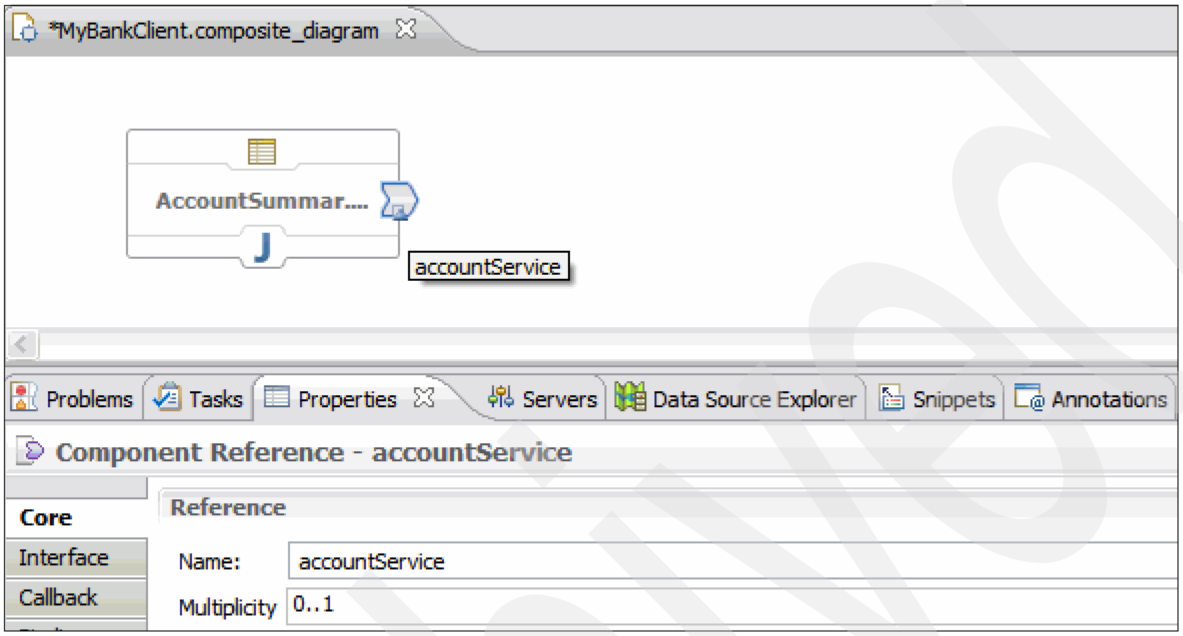
Implementation name: samples.mybank.AccountSummaryServiceImpl

Figure 3-26 Create the AccountSummaryService component

Add a reference to the composite

To add a reference to the composite:

1. Open the composite in the work area, and add a reference. In the Properties view, click the Core tab, and specify **accountService** as the reference name, as shown in Figure 3-27.



Component Reference - accountService

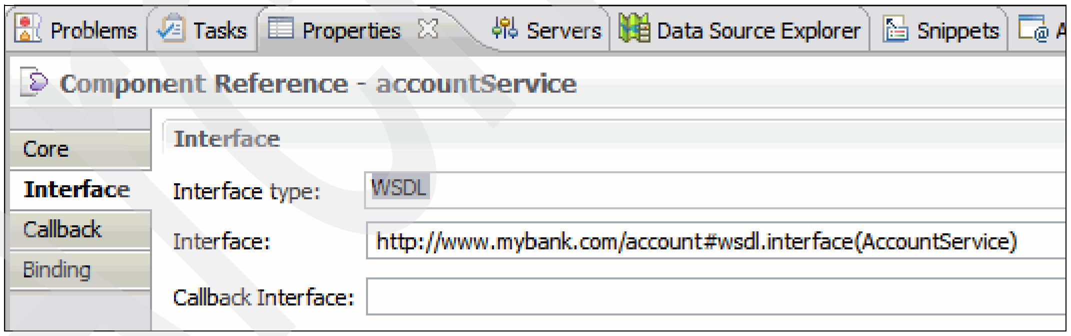
Reference

Name: accountService

Multiplicity 0..1

Figure 3-27 Define the reference name

2. Click the Interface tab, and define AccountService as the WSDL interface, as shown in Figure 3-28.



Component Reference - accountService

Interface

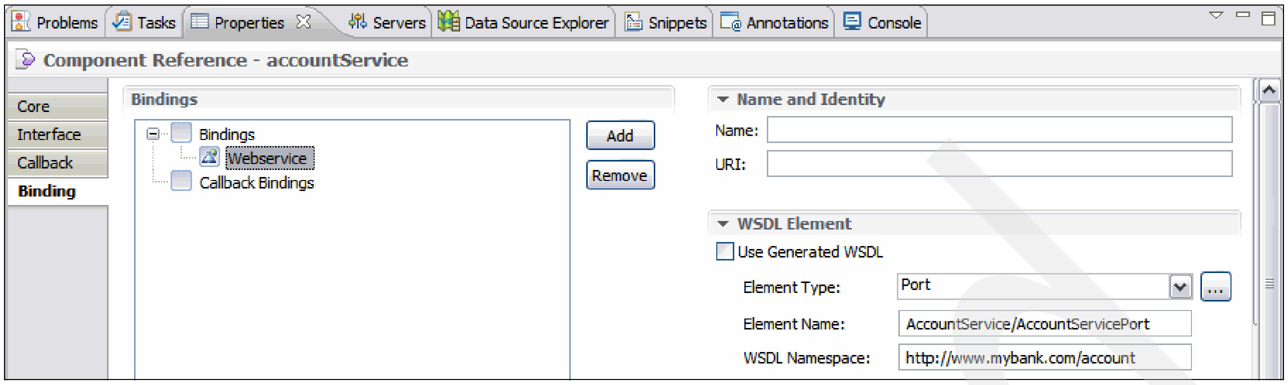
Interface type: WSDL

Interface: http://www.mybank.com/account#wsdl.interface(AccountService)

Callback Interface:

Figure 3-28 Define the WSDL location

3. Click the Binding tab, and define a Web service binding, as shown in Figure 3-29.



Bindings
- Bindings
  - Webservice
- Callback Bindings

Name:

URI:

Use Generated WSDL

Element Type: Port

Element Name: AccountService/AccountServicePort

WSDL Namespace: http://www.mybank.com/account

Figure 3-29 Create the Web service binding

4. Ensure that the new composite file with the content shown in Example 3-5 is added to the `META-INF/sca-deployables` folder.

Example 3-5 The default.composite file

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" name="MyBankClient"
targetNamespace="http://www.ibm.com/samples/sca/mybank">
   <component name="AccountSummaryService">
         <implementation.java class="samples.mybank.AccountSummaryServiceImpl"/>
         <reference name="accountService" target="AccountServiceComponent">
               <interface.wsdl
interface="http://www.mybank.com/account#wsdl.interface(AccountService)"/>
               <binding.ws
wsdlElement="http://www.mybank.com/account#wsdl.port(AccountService/AccountService
Port)"/>
         </reference>
   </component>
</composite>
```

Deploying an SCA application

This chapter describes how to deploy the sample application to the WebSphere Application Server run time. The Feature Pack for SCA supports the deployment of composites as assets that can be added to business-level application as composition units.

A typical SCA solution consists of a combination of Web, EJB, and SCA applications. The Web and EJB modules are deployed as enterprise applications, and the SCA composites are deployed as assets in a business-level application.

4.1 Deploy MyBankClient WAR file

This section provides instructions for deploying the MyBankClient WAR file. The WAR file is installed as a stand-alone WAR file (not as part of an EAR file).

4.1.1 Export the project as a WAR file

Right-click the MyBankClient project, and select **Export** → **WAR file**. Specify the location to store the file. This is the location from where you will import it to WebSphere Application Server.

4.1.2 Install the WAR file as a new enterprise application

Use the administrative console option to install a new enterprise application:

1. Browse to and select the exported WAR file.
2. Specify `mybankClient` as the context root.
3. Start the application.

4.2 Deploying the MyBank service SCA application

This section describes how to export and deploy an SCA application.

4.2.1 Exporting the application from the workspace

Packaging and exporting an SCA application for deployment is a two-part process in which you complete the following tasks:

- Create a contribution for the SCA project
- Export the project as an SCA archive file

Create a contribution for the SCA project

When you package an SCA application for deployment, you need to include an SCA contribution file. A contribution file contains a list of composites to be deployed.

You have to include the contribution file in the project in one of the following locations:

- The `META-INF/sca-contribution.xml` file in the root of the package
- The `META-INF/sca-deployables` directory with the composite file underneath

Tip: If you need to re-create the contribution file, you need to delete the existing contribution file first before using the SCA tools to create a new one. The file is located in the `<`*`SCA_project`*`>/META-INF/sca-contribution.xml` directory. A representation of the file is also in the `<`*`SCA_project`*`>/SCA Content/Contributions` folder.

Example 4-1 shows the contents of the contribution file for the MyBankService project.

Example 4-1 Contribution file

```
<?xml version="1.0" encoding="UTF-8"?>
<contribution xmlns="http://www.osoa.org/xmlns/sca/1.0">
   <deployable composite="ns1:MyBank"
xmlns:ns1="http://www.ibm.com/samples/sca/mybank"></deployable>
</contribution>
```

To create a contribution file in Rational Application Developer:

1. Be sure to save all files that you might have opened in the workspace.
2. In the Enterprise Explorer view, right-click the **SCA Content** → **Contributions** folder, and select **New** → **SCA Contribution**, as shown in Figure 4-1.

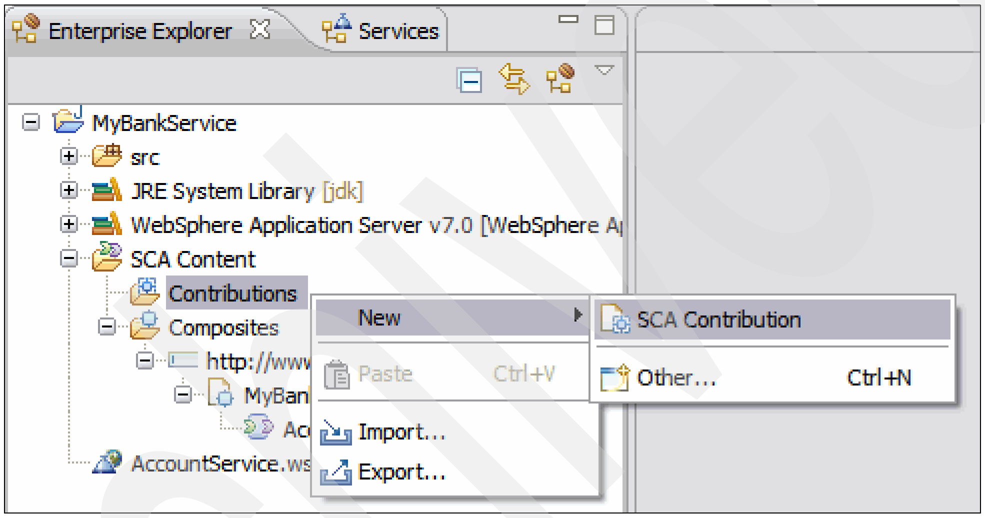
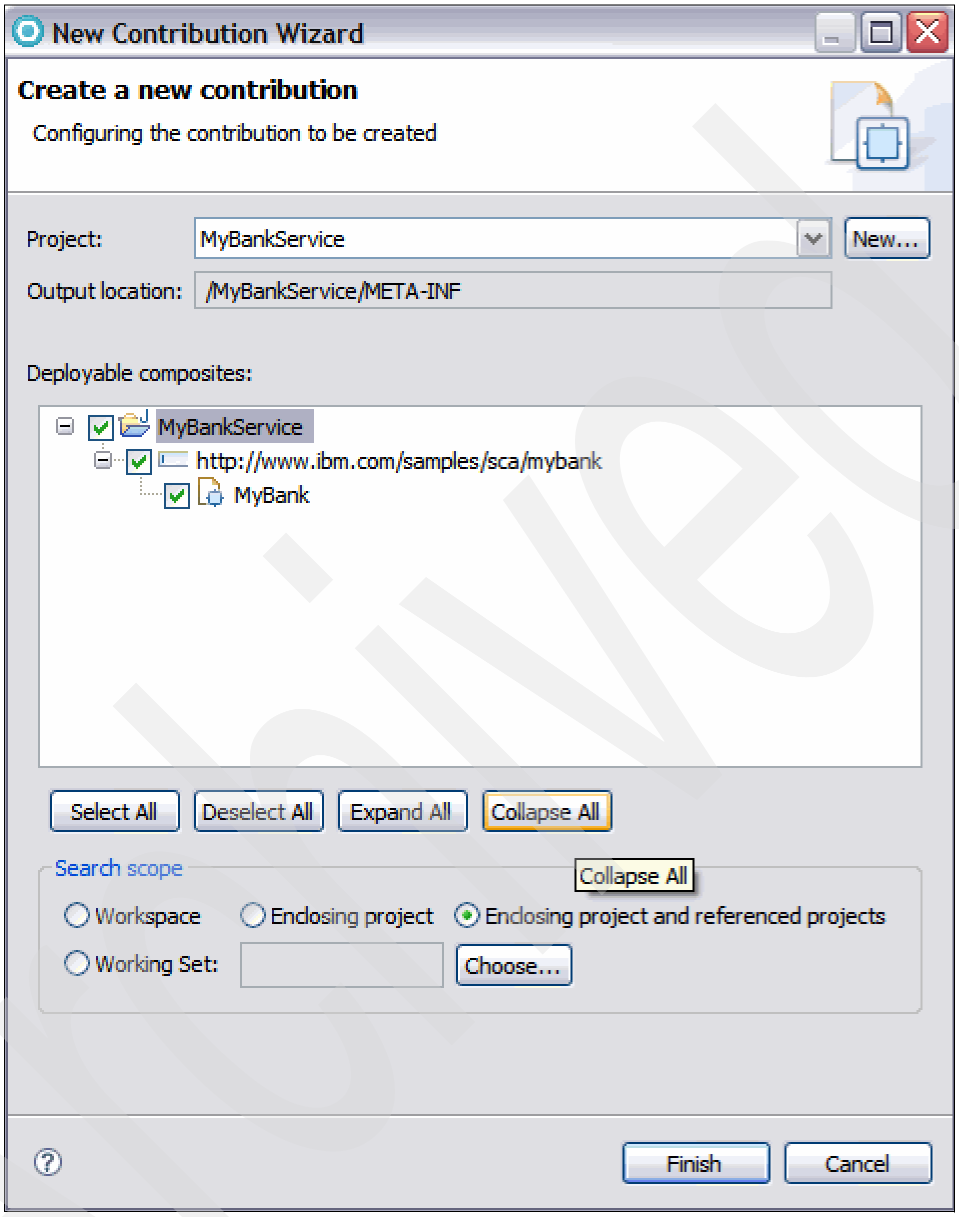


Figure 4-1 Create a contribution: Step 1

3. Select the composites to include in the contribution as shown in Figure 4-2, and click **Finish**.


New Contribution Wizard
Create a new contribution
Configuring the contribution to be created
Project: MyBankService

Output location: /MyBankService/META-INF

Deployable composites:

[✓] MyBankService
[✓] http://www.ibm.com/samples/sca/mybank
[✓] MyBank

Search scope

Workspace Enclosing project [✓ Enclosing project and referenced projects]

Working Set:

Figure 4-2   Create a contribution: Step 2

The new contribution is added to the **SCA Content** → **Contributions** folder and opens in the work area, as shown in Figure 4-3.

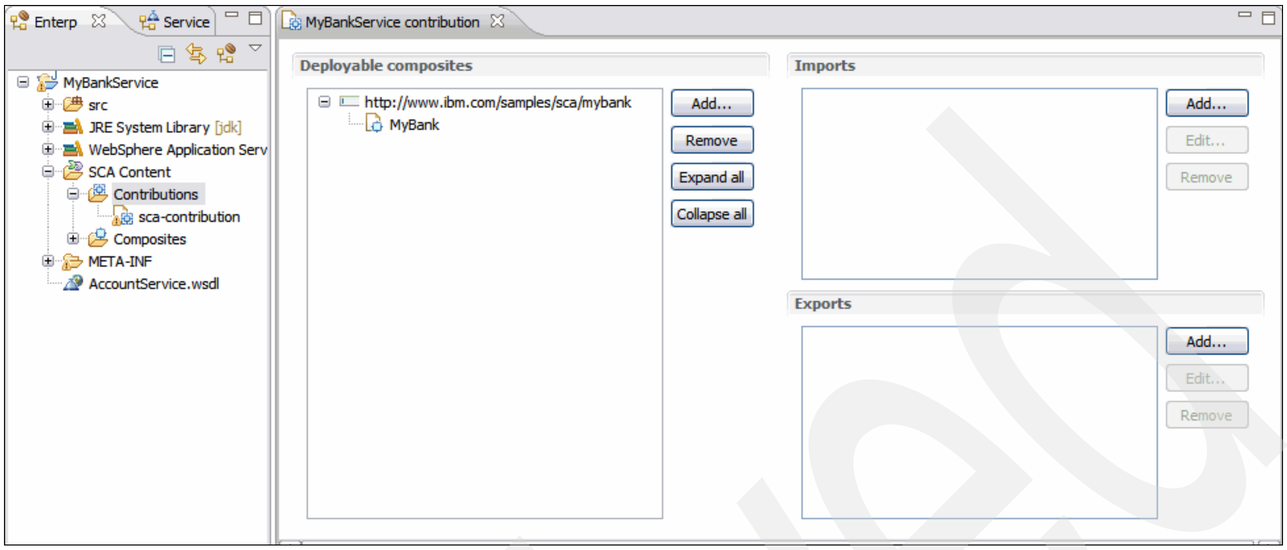


Figure 4-3 Create a contribution: Step 3

A warning might display in the Problems view:

```
One or more projects that make up this contribution has been configured to
use the Spring implementation type but the Spring runtime asset jar has not
been configured.
```

You can ignore this warning. It tells you to choose Spring as one of the possible implementations in a project, but we did not use it in our example.

4. Close and save the contribution.
5. If you are deploying to the WebSphere Application Server V7 unit test environment in Rational Application Developer, you can deploy the asset to the server. Select the server in the Servers view, right-click, and select **Add and Remove projects**.

   In our example, we export the project for deployment and install the application manually. The method used is the same method that you use for a separate installation of WebSphere.

Export the project as an SCA archive file

The final preparation for deployment is to export the contribution as an SCA archive file:

1. Right-click the project, and select **Export**, as shown in Figure 4-4.

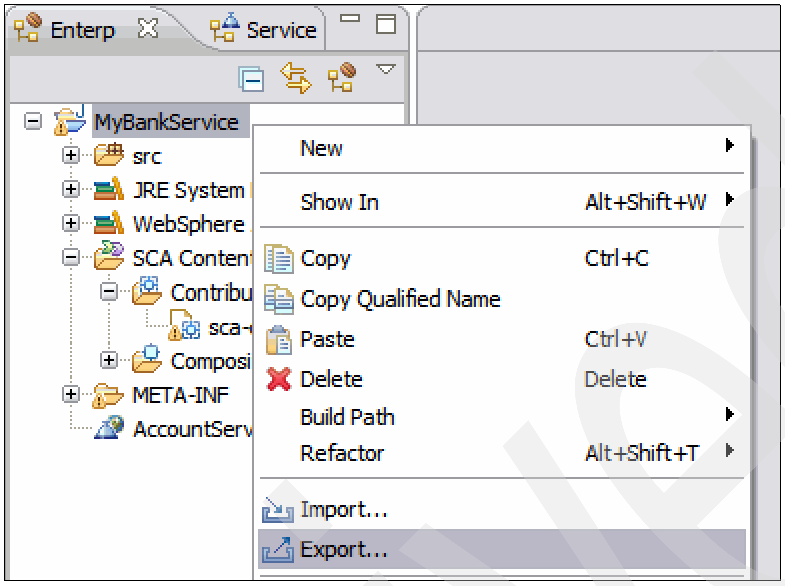
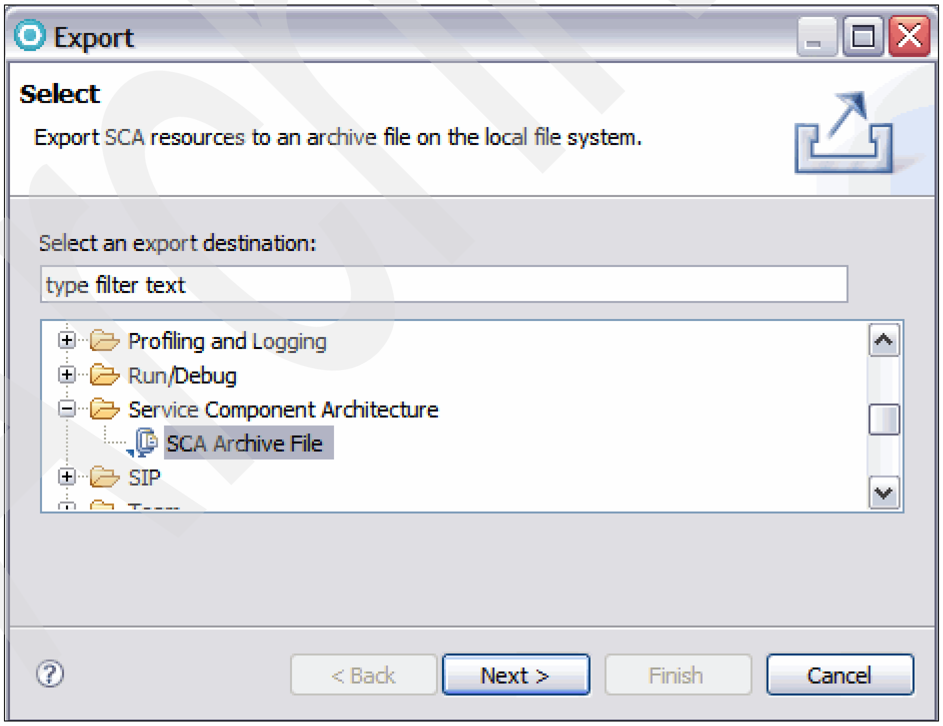


Figure 4-4 Export the SCA application: Step 1

2. Select **SCA Archive File** as the type of export file as shown in Figure 4-5.



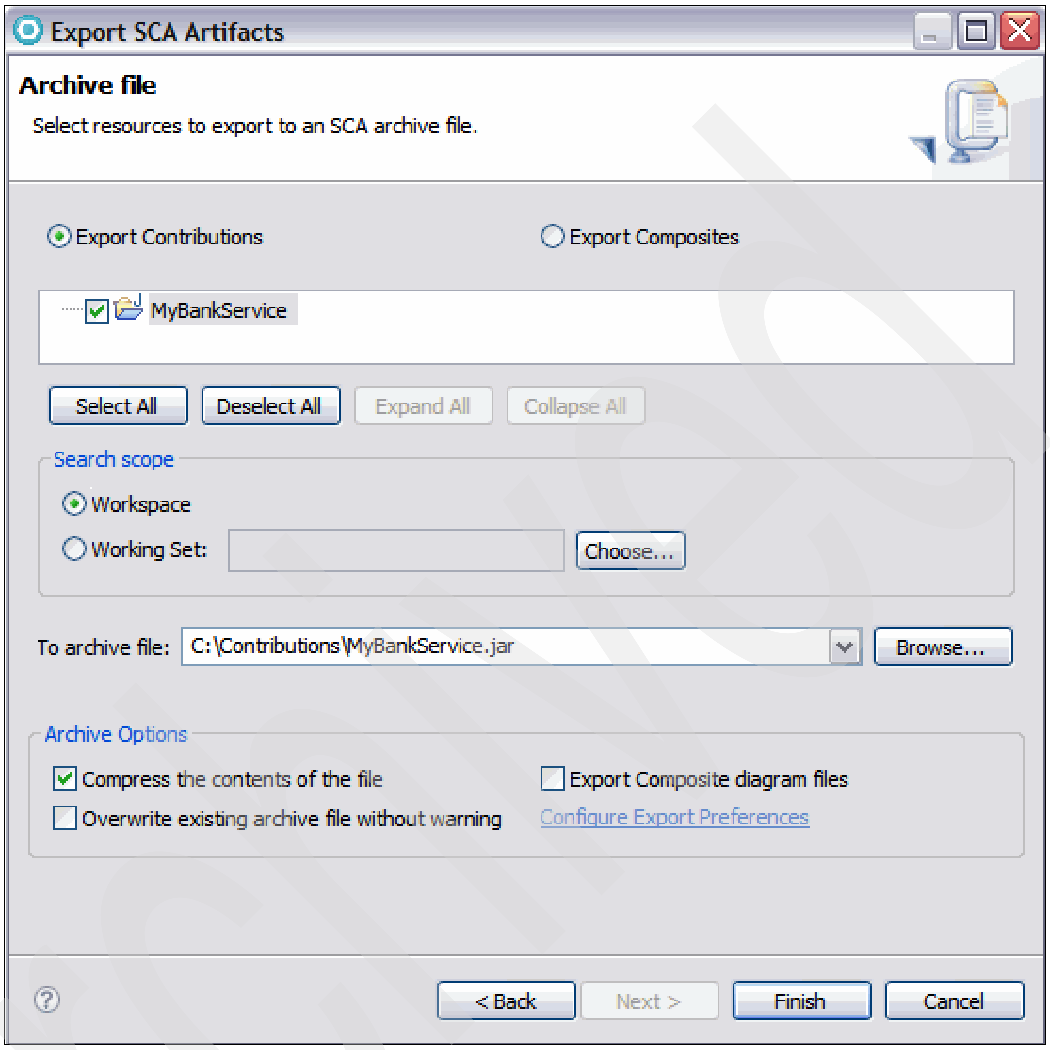
Export SCA resources to an archive file on the local file system.

Select an export destination:

type filter text

Figure 4-5 Export the SCA application: Step 2

3. Ensure that sure the “Export Contributions” option is selected, and select the contribution file, as shown in Figure 4-6. Then, click **Finish**.



[✓ Export Contributions] Export Composites

[✓] MyBankService

Search scope: [✓ Workspace] Working Set:

To archive file: C:\Contributions\MyBankService.jar

Archive Options:
[✓] Compress the contents of the file
[ ] Overwrite existing archive file without warning
[ ] Export Composite diagram files

Figure 4-6 Export the SCA application: Step 3

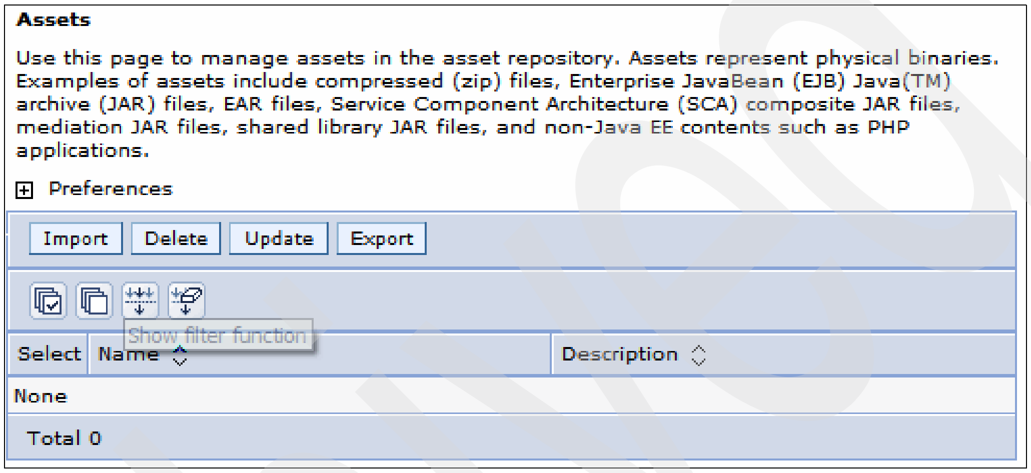
4.2.2 Deploying the composite to the application server

The next step is to deploy the composite. In this example, we use the administrative console for the application server.

Import the SCA archive file as an asset

The SCA archive file is imported into the application server environment as an asset.

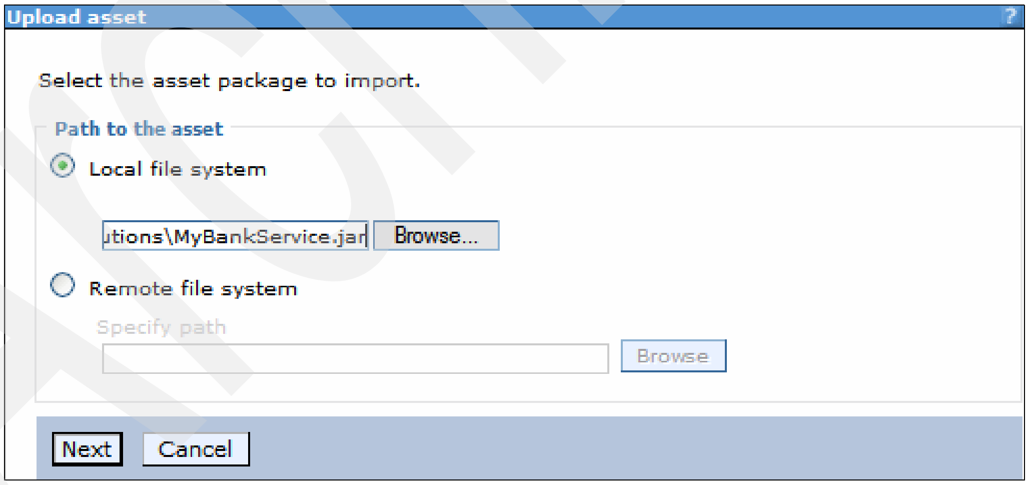
1. Select **Applications** → **Application types** → **Assets**. In the Assets panel, shown in Figure 4-7, click **Import**.



Use this page to manage assets in the asset repository. Assets represent physical binaries. Examples of assets include compressed (zip) files, Enterprise JavaBean (EJB) Java(TM) archive (JAR) files, EAR files, Service Component Architecture (SCA) composite JAR files, mediation JAR files, shared library JAR files, and non-Java EE contents such as PHP applications.

Figure 4-7 Import an asset

2. Select the SCA archive file as shown in Figure 4-8, and click **Next**.



Select the asset package to import.

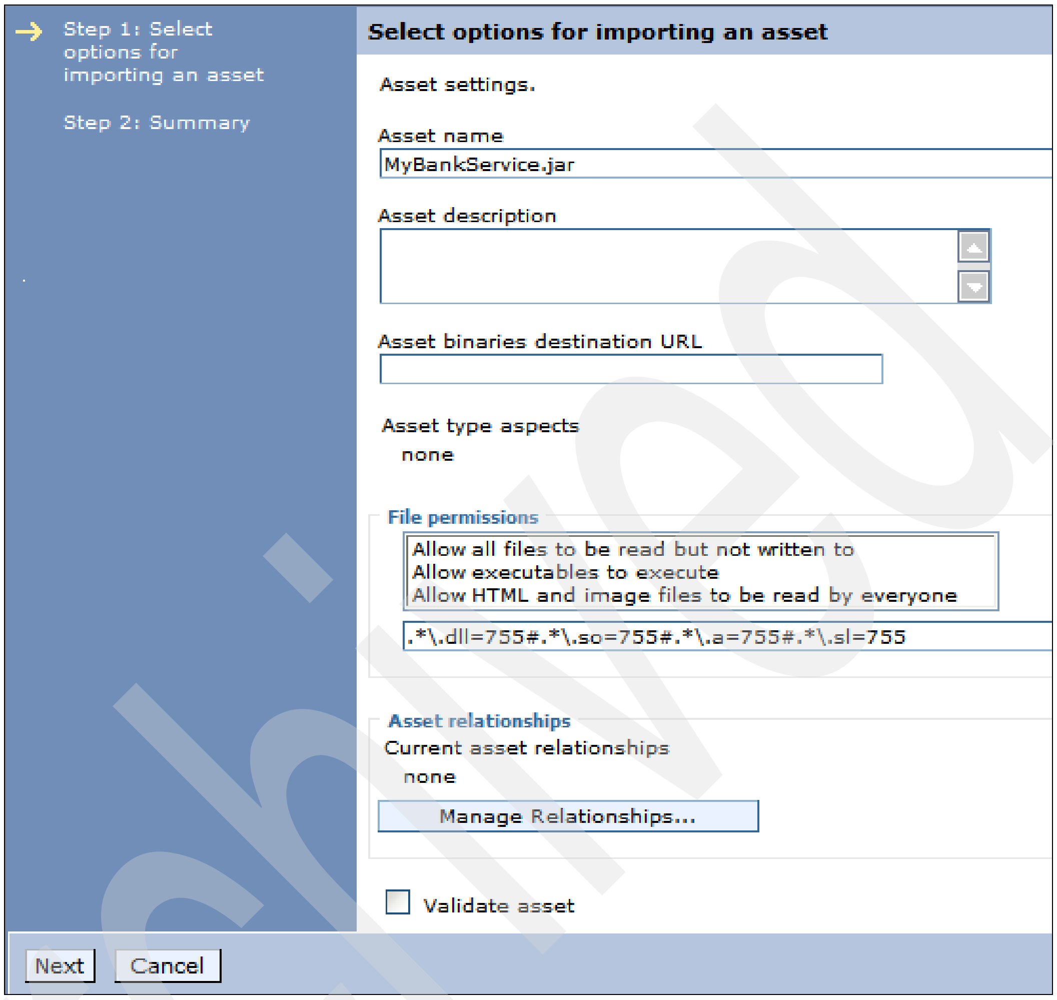
(●) Local file system

ıtions\MyBankService.jar

( ) Remote file system

Figure 4-8 Provide the location of the contribution file

3. Provide any options required. In our example, we take the defaults, as shown in Figure 4-9. Click **Next**.



Select options for importing an asset

Asset settings.

Asset name
MyBankService.jar

Asset description

Asset binaries destination URL

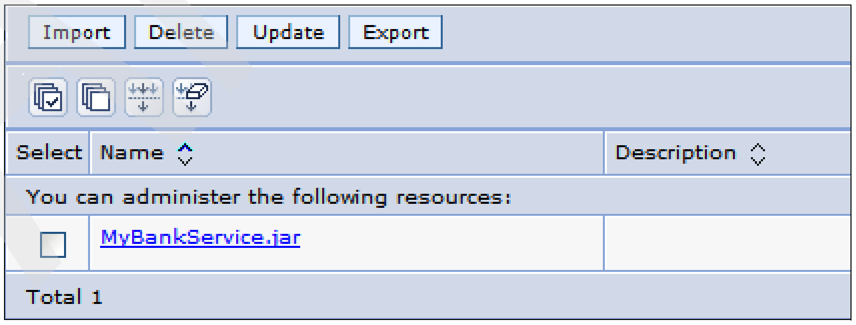
Asset type aspects
none

File permissions
Allow all files to be read but not written to
Allow executables to execute
Allow HTML and image files to be read by everyone
.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755

Asset relationships
Current asset relationships
none

Figure 4-9 Provide additional settings if necessary

4. On the summary page, click **Finish** to import the file. The new SCA archive file is now listed as an asset, as shown in Figure 4-10.



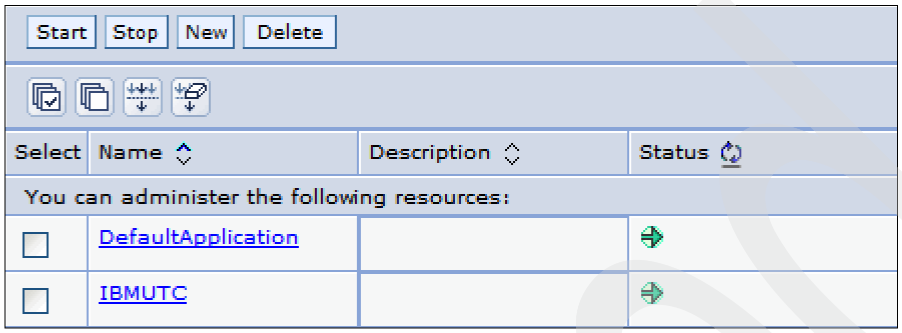
Select	Name	Description
You can administer the following resources:		
☐	MyBankService.jar	
Total 1		

Figure 4-10 New asset imported

Create a new business-level application

To create a new business-level application:

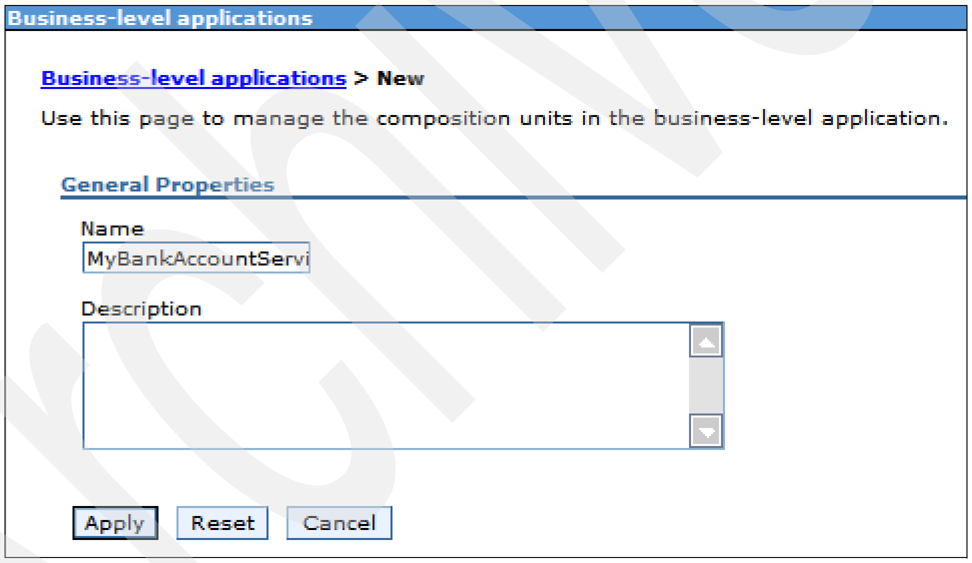
1. Select **Applications** → **Application types** → **Business-level applications**. To create a new business-level application, click **New** in the panel that opens, as shown in Figure 4-11.



Select	Name	Description	Status
You can administer the following resources:			
☐	DefaultApplication		➡
☐	IBMUTC		➡

Figure 4-11 List of business-level applications

2. Enter a new name for the business-level application, and click **Apply**. For our example, we name this new BLA `MyBankAccountService`, as shown in Figure 4-12.



Business-level applications > New

Use this page to manage the composition units in the business-level application.

General Properties

Name: MyBankAccountServi

Description:

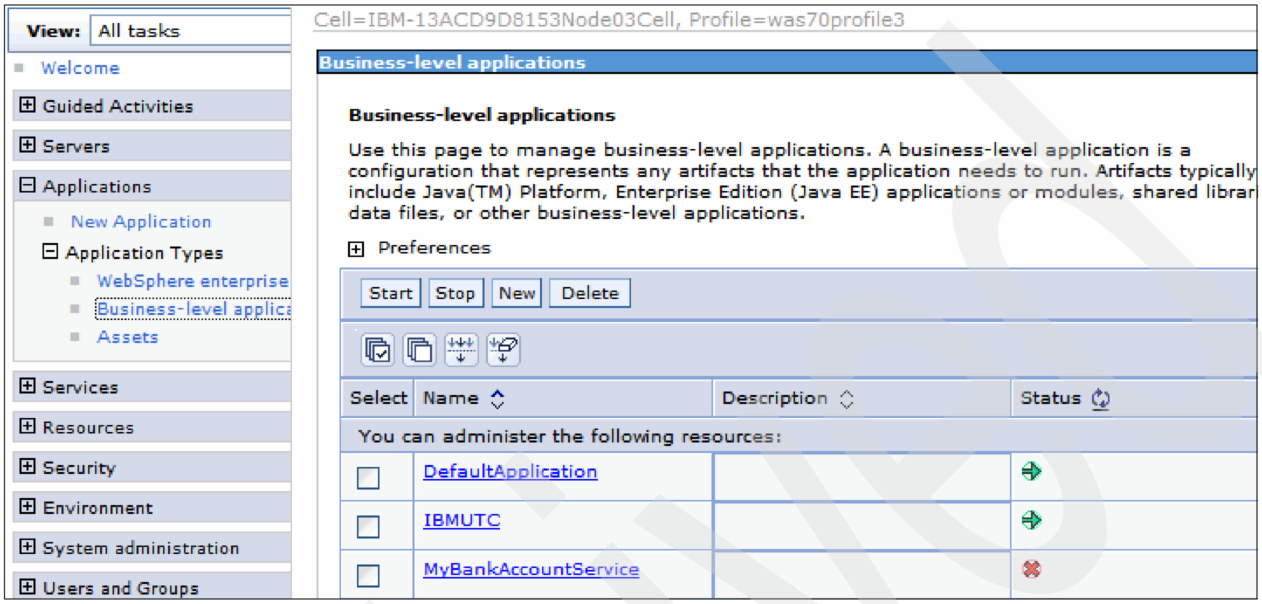
Figure 4-12 Define the new business-level application

3. Save the configuration.

Add the new asset to the business-level application

Next, you add the new asset to the MyBankAccountService business-level application:

1. Navigate to the list of business-level applications in the console, as shown in Figure 4-13. Click **MyBankAccountService**.



Cell=IBM-13ACD9D8153Node03Cell, Profile=was70profile3

Business-level applications

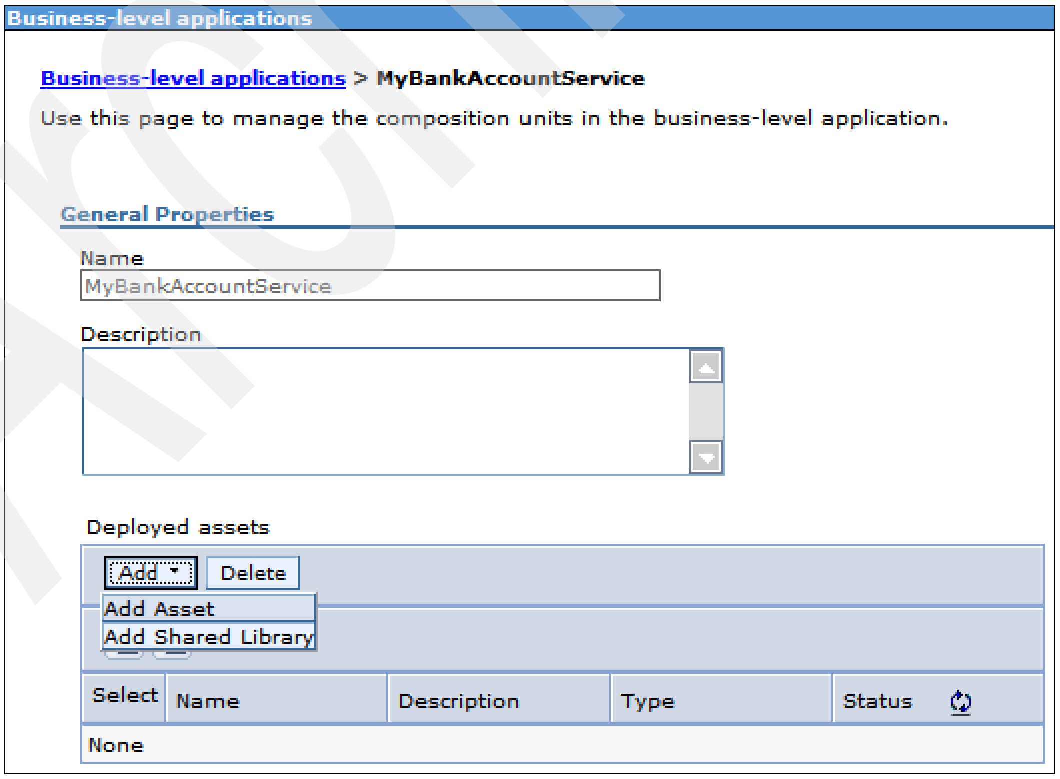
Business-level applications

Use this page to manage business-level applications. A business-level application is a configuration that represents any artifacts that the application needs to run. Artifacts typically include Java(TM) Platform, Enterprise Edition (Java EE) applications or modules, shared librar data files, or other business-level applications.

Select	Name	Description	Status
You can administer the following resources:			
☐	DefaultApplication		➔
☐	IBMUTC		➔
☐	MyBankAccountService		✖

Figure 4-13 List of business-level applications

2. In the properties page, click **Add** in the Deployed Assets section, as shown in Figure 4-14. Select **Add Asset**.



Business-level applications

Business-level applications > MyBankAccountService

Use this page to manage the composition units in the business-level application.

General Properties

Name
MyBankAccountService

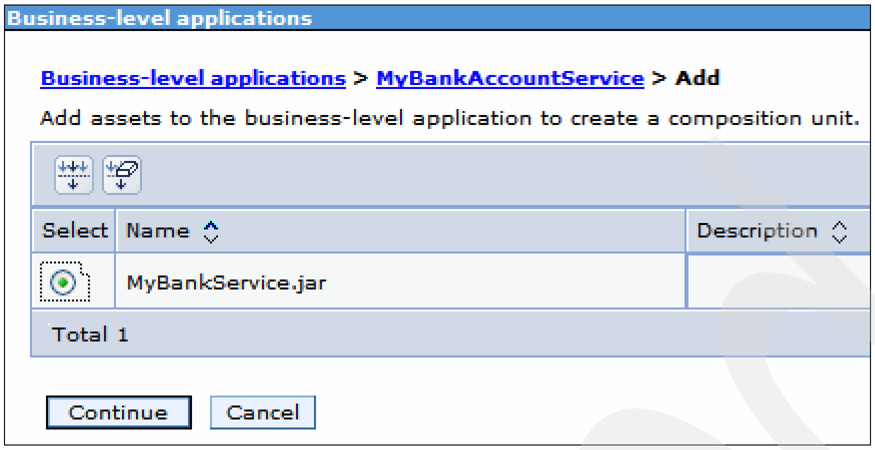
Description

Deployed assets

Select	Name	Description	Type	Status
None				

Figure 4-14 Add an asset to the business-level application

3. Select the SCA archive file as shown in Figure 4-15, and click **Continue**.


Business-level applications

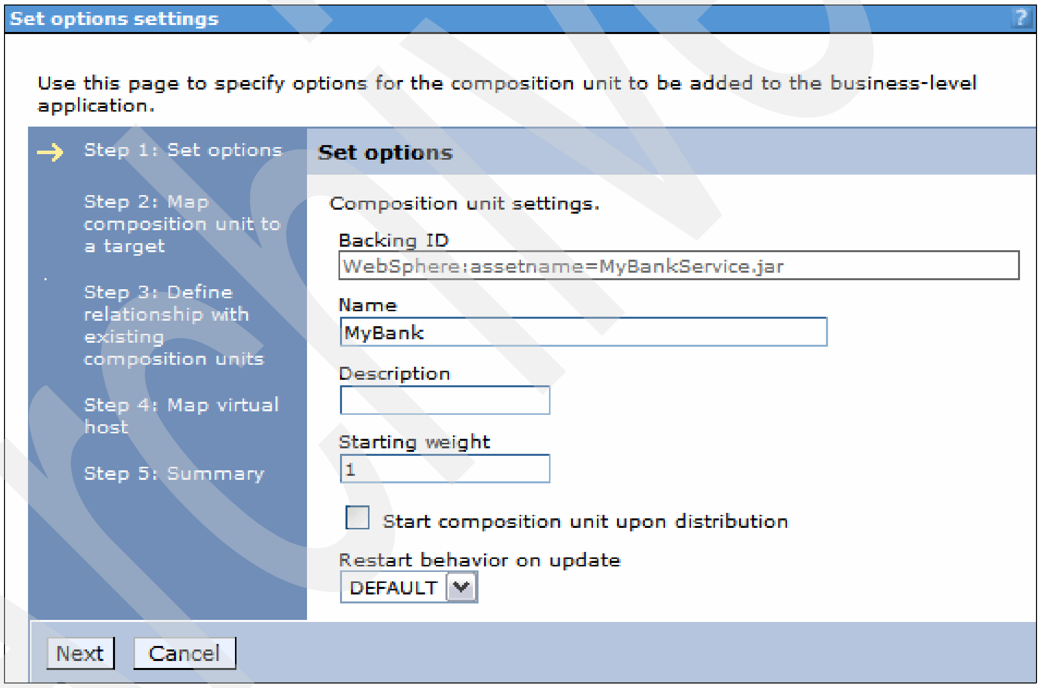
Business-level applications > MyBankAccountService > Add

Add assets to the business-level application to create a composition unit.

Select	Name	Description
●	MyBankService.jar	
Total 1		

Figure 4-15 Select the asset

4. Review the options settings as shown in Figure 4-16, and click **Next**.


Set options settings

Use this page to specify options for the composition unit to be added to the business-level application.

→ Step 1: Set options
Step 2: Map composition unit to a target
Step 3: Define relationship with existing composition units
Step 4: Map virtual host
Step 5: Summary

Set options

Composition unit settings.

Backing ID: WebSphere:assetname=MyBankService.jar

Name: MyBank

Description:

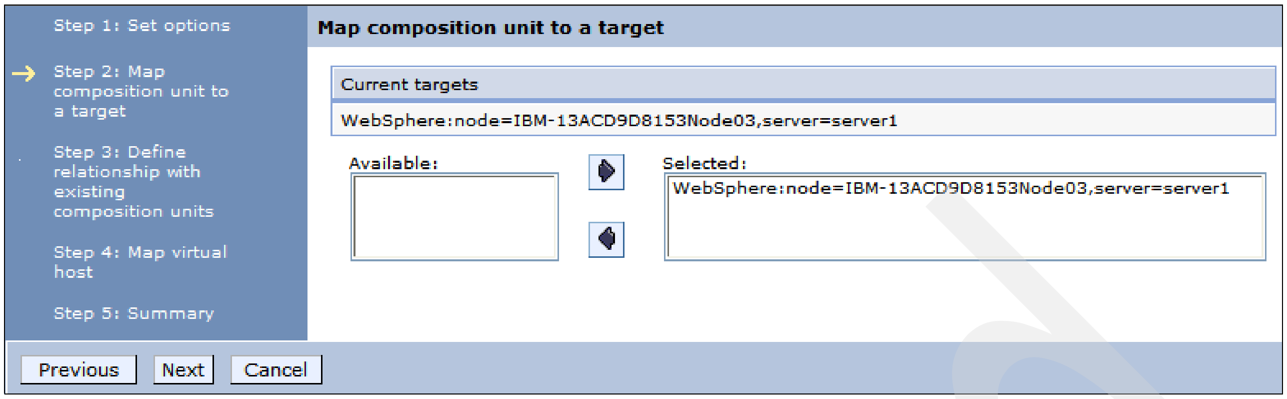
Starting weight: 1

☐ Start composition unit upon distribution

Restart behavior on update: DEFAULT

Figure 4-16 Review the options

5. Select the server where the asset will run as shown in Figure 4-17 on page 59, and click **Next**. When you add an asset, you must specify a target server or cluster that is enabled with the Feature Pack for SCA. Specify only a single server or cluster as the target. Do not map an SCA composition unit to multiple servers or clusters.


Step 1: Set options
Step 2: Map composition unit to a target
Step 3: Define relationship with existing composition units
Step 4: Map virtual host
Step 5: Summary

Map composition unit to a target

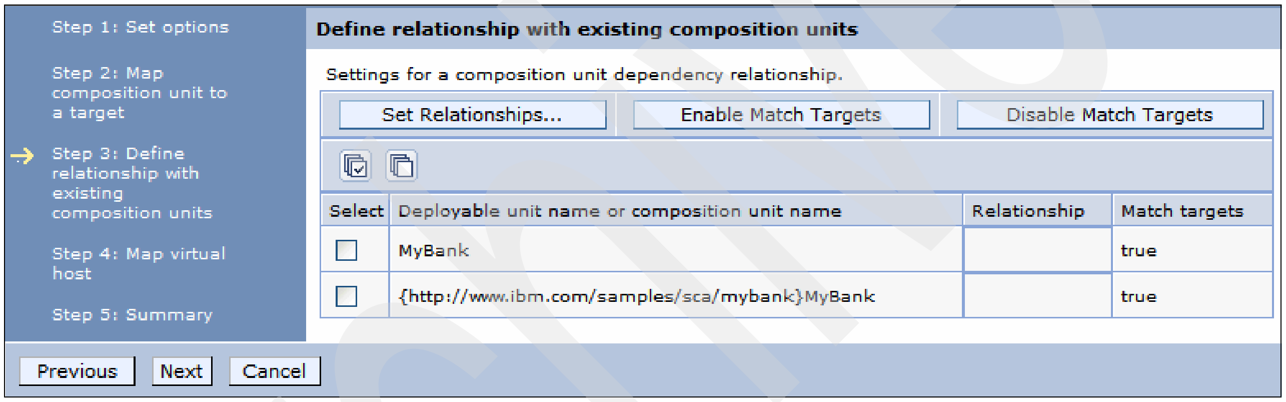
Current targets
WebSphere:node=IBM-13ACD9D8153Node03,server=server1

Available:

Selected: WebSphere:node=IBM-13ACD9D8153Node03,server=server1

Figure 4-17 Select the server where the asset will run

6. Review the relationship settings, as shown in Figure 4-18. If the composite you are installing requires a shared library asset, you need to define that relationship. You can find an example of this type of relationship in Chapter 9, “Using a Spring implementation as an SCA component” on page 123. Click **Next**.


Step 1: Set options
Step 2: Map composition unit to a target
Step 3: Define relationship with existing composition units
Step 4: Map virtual host
Step 5: Summary

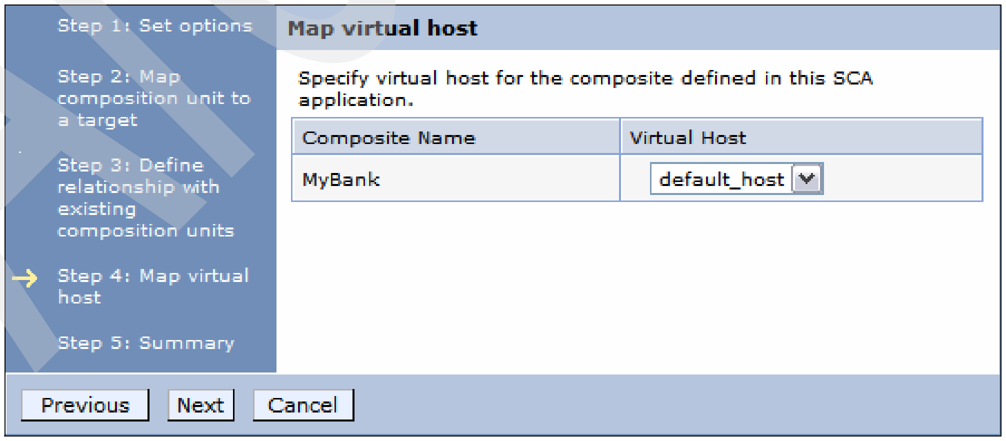
Define relationship with existing composition units

Settings for a composition unit dependency relationship.

Select	Deployable unit name or composition unit name	Relationship	Match targets
☐	MyBank		true
☐	{http://www.ibm.com/samples/sca/mybank}MyBank		true

Figure 4-18 Define relationships

7. Select the virtual host as shown in Figure 4-19, and click **Next**.


Step 1: Set options
Step 2: Map composition unit to a target
Step 3: Define relationship with existing composition units
Step 4: Map virtual host
Step 5: Summary

Map virtual host

Specify virtual host for the composite defined in this SCA application.

Composite Name	Virtual Host
MyBank	default_host

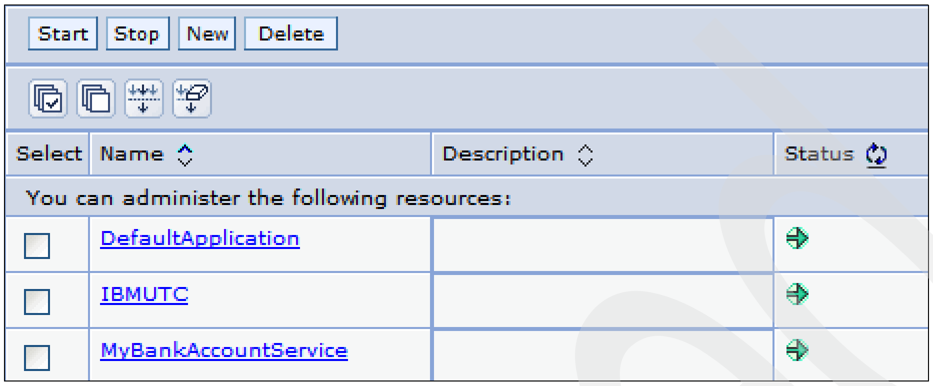
Figure 4-19 Select the virtual host

8. On the Summary page, review the selections, and click **Finish**.
9. Save the configuration.

Start and verify the business-level application

The last step is to start the application and to verify that it is available:

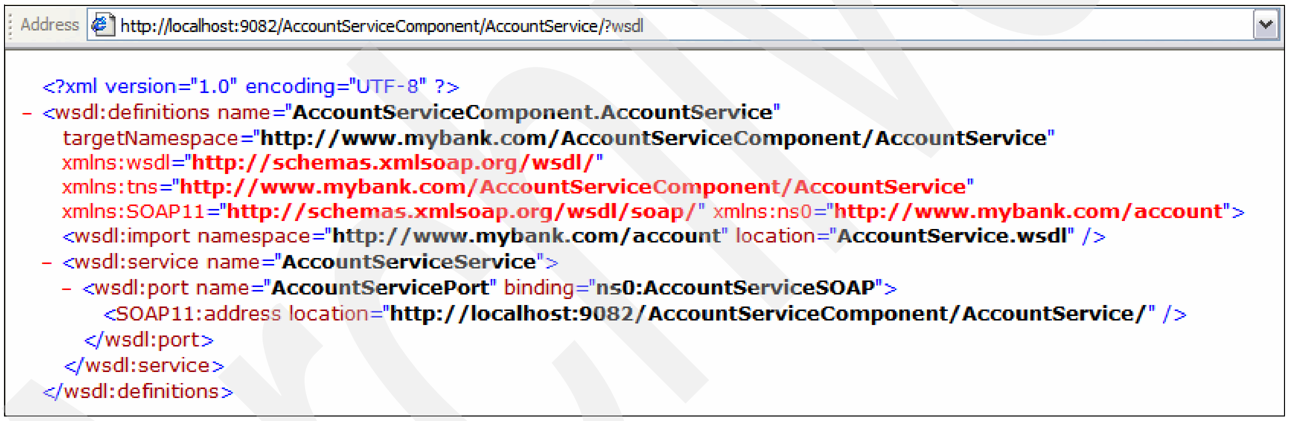
1. Navigate to the list of business-level applications, as shown in Figure 4-20. Select the box to the left of MyBankAccountService, and click **Start**.



Select	Name	Description	Status
You can administer the following resources:			
☐	DefaultApplication		➔
☐	IBMUTC		➔
☐	MyBankAccountService		➔

Figure 4-20 Start the application

2. Verify that the WSDL file is available from the server by entering the location URL in a Web browser, as shown in Figure 4-21.



```
http://localhost:9082/AccountServiceComponent/AccountService/?wsdl
```

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions name="AccountServiceComponent.AccountService"
    targetNamespace="http://www.mybank.com/AccountServiceComponent/AccountService"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:tns="http://www.mybank.com/AccountServiceComponent/AccountService"
    xmlns:SOAP11="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:ns0="http://www.mybank.com/account">
    <wsdl:import namespace="http://www.mybank.com/account" location="AccountService.wsdl" />
  - <wsdl:service name="AccountServiceService">
    - <wsdl:port name="AccountServicePort" binding="ns0:AccountServiceSOAP">
        <SOAP11:address location="http://localhost:9082/AccountServiceComponent/AccountService/" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

Figure 4-21 Display the service WSDL file

4.3 Test the MyBank service and client

Now that you have deployed the MyBank client and service, you can test the application. The MyBank application uses a Derby database shipped with the sample. This section describes how to define the database to the run time.

4.3.1 Create the database and the data source definition

The MyBank application uses a Derby database shipped with the sample. This section describes how to define the database to the run time.

Create the JDBC provider

To imitate the instructions from the sample, use a Derby database for this application. You can create a Derby JDBC provider if there is not one already defined as follows:

1. From the left navigation panel of the Administrative console, expand **Resources** → **JDBC**, and click JDBC Providers.
2. On the JDBC Provider page:
   a. Select **Node=*node_name*, Server=server1** from the scope drop-down list before creating a new JDBC provider.
   b. Click **New** to create a new JDBC Provider.
   c. Select **Derby** from the Database type drop-down list.
   d. Select **Derby JDBC Provider** from the Provider type drop-down list.
   e. Select **Connection pool data source** from the Implementation type drop-down list
   f. Accept the default name of **Derby JDBC Provider**.
   g. Click **Next**.
3. Click **Finish** on the Summary page.
4. Click **Save** at the top of the JDBC Provider page to save your changes.

Create the data source

You can create a new data source for use in the MyBank application as follows:

1. From the left navigation panel of the Administrative console, expand **Resources** → **JDBC**, and click **Data sources**.
2. On the Data Sources page:
   a. Select **Node=*node_name*, Server=*server_name*** from the scope drop-down list.
   b. Click **New** to create a new data source.
   c. Enter `MyBankDB` for the Data source name.
   d. Enter `mybank` for the JNDI name.
   e. Click **Next**.
3. For the JDBC Provider, select the **Derby JDBC Provider** from the drop-down list.
4. Click **Next**.
5. On the database specific properties page:
   a. Enter the path to the MyBankDB database in the database name field. The database is located in the following location:

      `*app_server_root*\samples\SCA\MyBank\MyBankDB`

      For example,

      `C:/WebSphere/AppServer/samples/SCA/MyBank/MyBankDB`

   b. Clear the "Use this data source in container managed persistence (CMP)" option, and click **Next.**
6. On the Setup Security Aliases page, take the default value, and click **Next**.

7. Click **Finish** on the summary panel.
8. Click **Save** at the top of the Data sources page to save your changes.
9. After you save the changes, select **MyBankDB** in the list of Data sources, and click **Test Connection**. Verify that you see a message that the test was successful.

4.3.2 Invoke the client

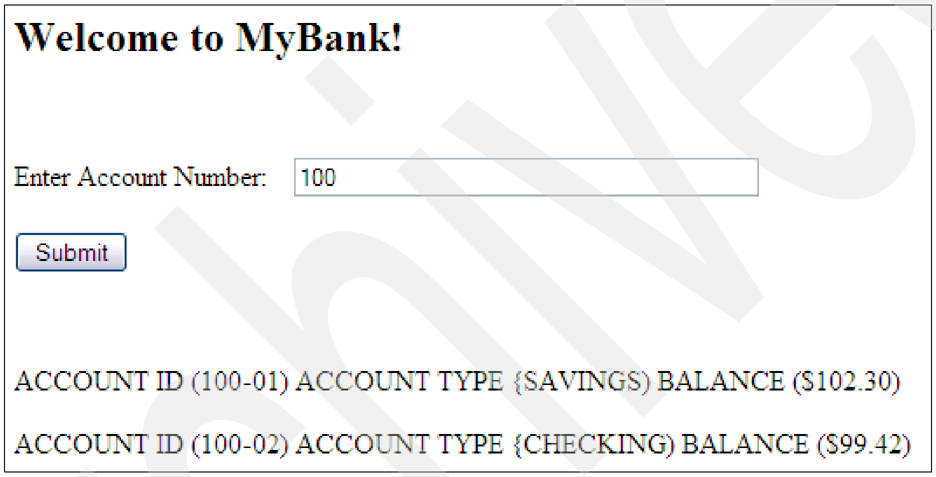
Invoke the JSP using the following address:

```
http://localhost:WC_defaulthost_port/MyBankClient/MyBankWeb.jsp
```

For example:

```
http://localhost:9082/MyBankClient/MyBankWeb.jsp
```

The JSP contains a field for the account number, in which the default value of `100` is entered. Click **Submit**. The application returns results as shown in Figure 4-22.



Welcome to MyBank!

Enter Account Number: 100

Submit

ACCOUNT ID (100-01) ACCOUNT TYPE {SAVINGS) BALANCE ($102.30)

ACCOUNT ID (100-02) ACCOUNT TYPE {CHECKING) BALANCE ($99.42)

Figure 4-22 MyBank test

5

Administering SCA applications with the administrative console

In 4.2.2, “Deploying the composite to the application server” on page 54, we explained how to deploy an SCA composite to WebSphere Application Server V7.

Now that you have deployed composites, the next logical step is to learn to administer the application and its resources. The process of managing WebSphere applications after they are deployed is essentially the same regardless of whether you are a developer working in a unit test environment or an administrator who has been handed an SCA archive to deploy and manage. Developers and administrators dealing with a single system typically use the administrative console for these tasks. Thus, we use the administrative console in this chapter.

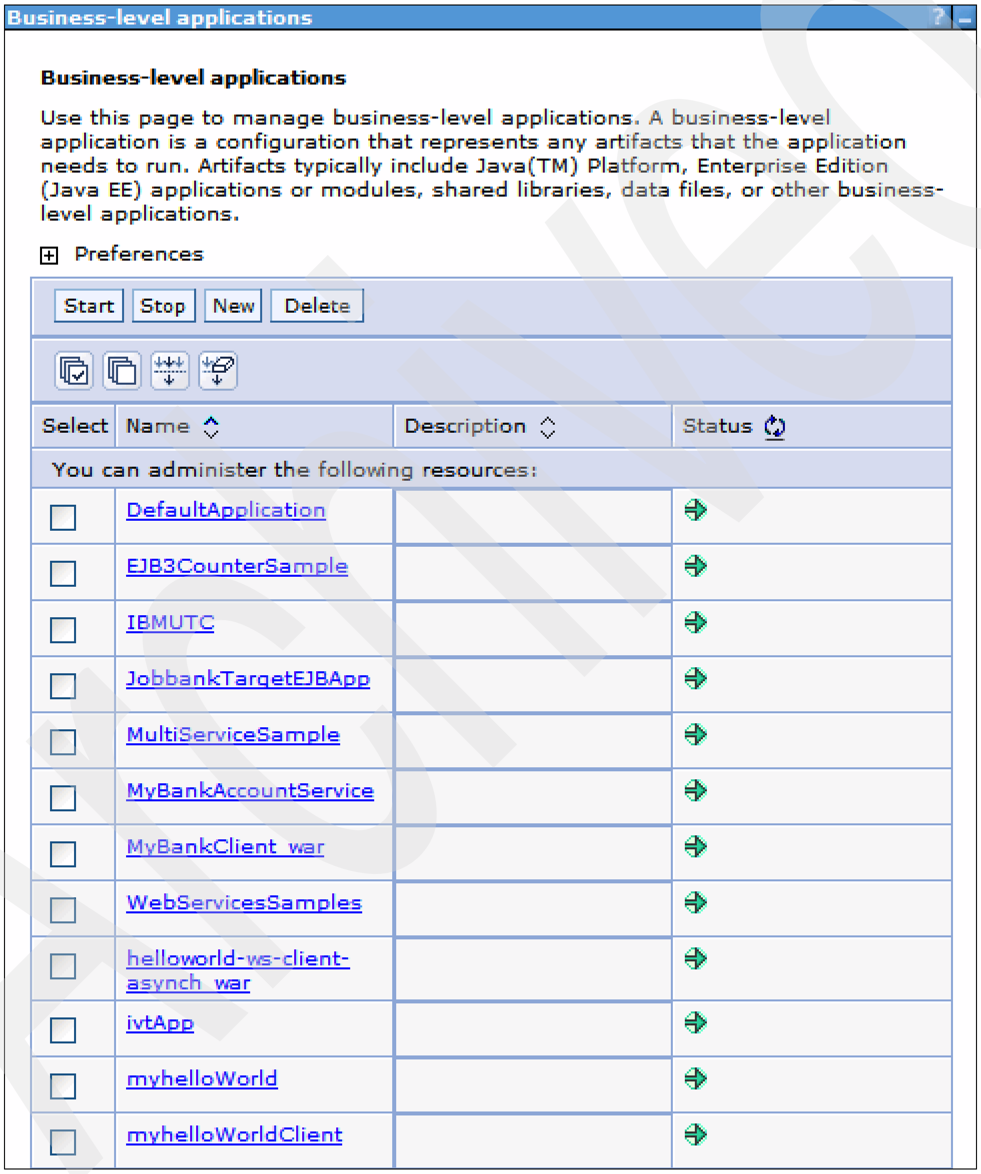
This chapter describes the tasks that are typically completed by the administrator for SCA applications.

Note: Some administrative options for SCA applications in the WebSphere Application Server administrative console display only if the composite has certain configuration contents. Thus, we use a variety of deployed SCA examples to illustrate these concepts.

5.1 Business-level applications

As discussed in 1.5, “From an administrator’s perspective” on page 10, you install SCA archives into WebSphere Application Server as assets and then add them to business-level applications.

You can display the list of business-level applications in the administrative console by selecting **Applications** → **Application Types** → **Business-level applications**. A list of the applications and their status is displayed. You can use this page to start and stop business-level applications.



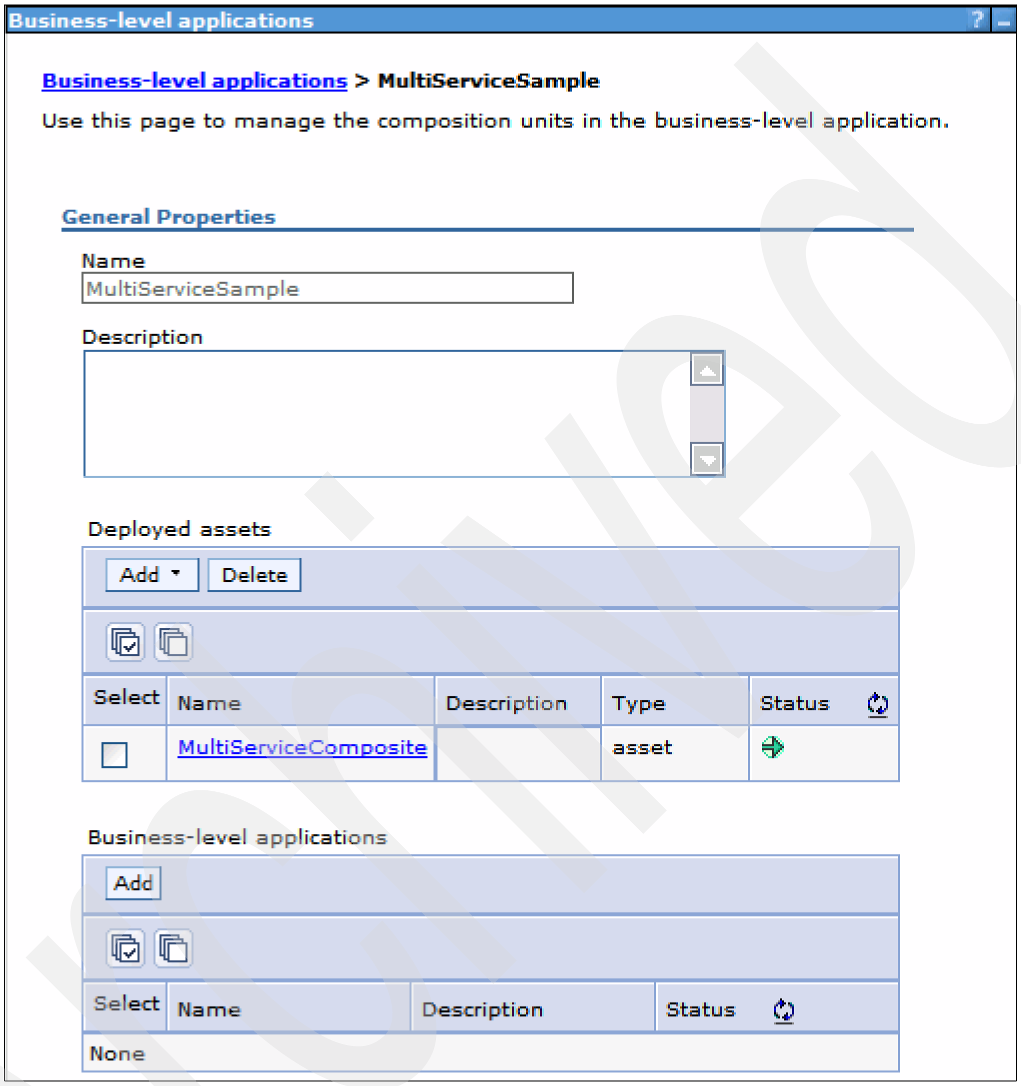
Business-level applications

Use this page to manage business-level applications. A business-level application is a configuration that represents any artifacts that the application needs to run. Artifacts typically include Java(TM) Platform, Enterprise Edition (Java EE) applications or modules, shared libraries, data files, or other business-level applications.

Select	Name	Description	Status
You can administer the following resources:			
☐	DefaultApplication		➜
☐	EJB3CounterSample		➜
☐	IBMUTC		➜
☐	JobbankTargetEJBApp		➜
☐	MultiServiceSample		➜
☐	MyBankAccountService		➜
☐	MyBankClient_war		➜
☐	WebServicesSamples		➜
☐	helloworld-ws-client-asynch_war		➜
☐	ivtApp		➜
☐	myhelloWorld		➜
☐	myhelloWorldClient		➜

Figure 5-1 List of business-level applications

Clicking a business-level application opens the configuration page. Figure 5-2 shows the configuration page for the MultiServiceSample business-level application. The page contains a list of the composition units, both deployed assets and business-level applications, that belong to the application. You can add or remove composition units from this page.


Business-level applications

Business-level applications > **MultiServiceSample**

Use this page to manage the composition units in the business-level application.

General Properties

Name: MultiServiceSample

Description:

Deployed assets

Select	Name	Description	Type	Status
☐	MultiServiceComposite		asset	➔

Business-level applications

Select	Name	Description	Status
None			

Figure 5-2 MultiServiceSample business-level application

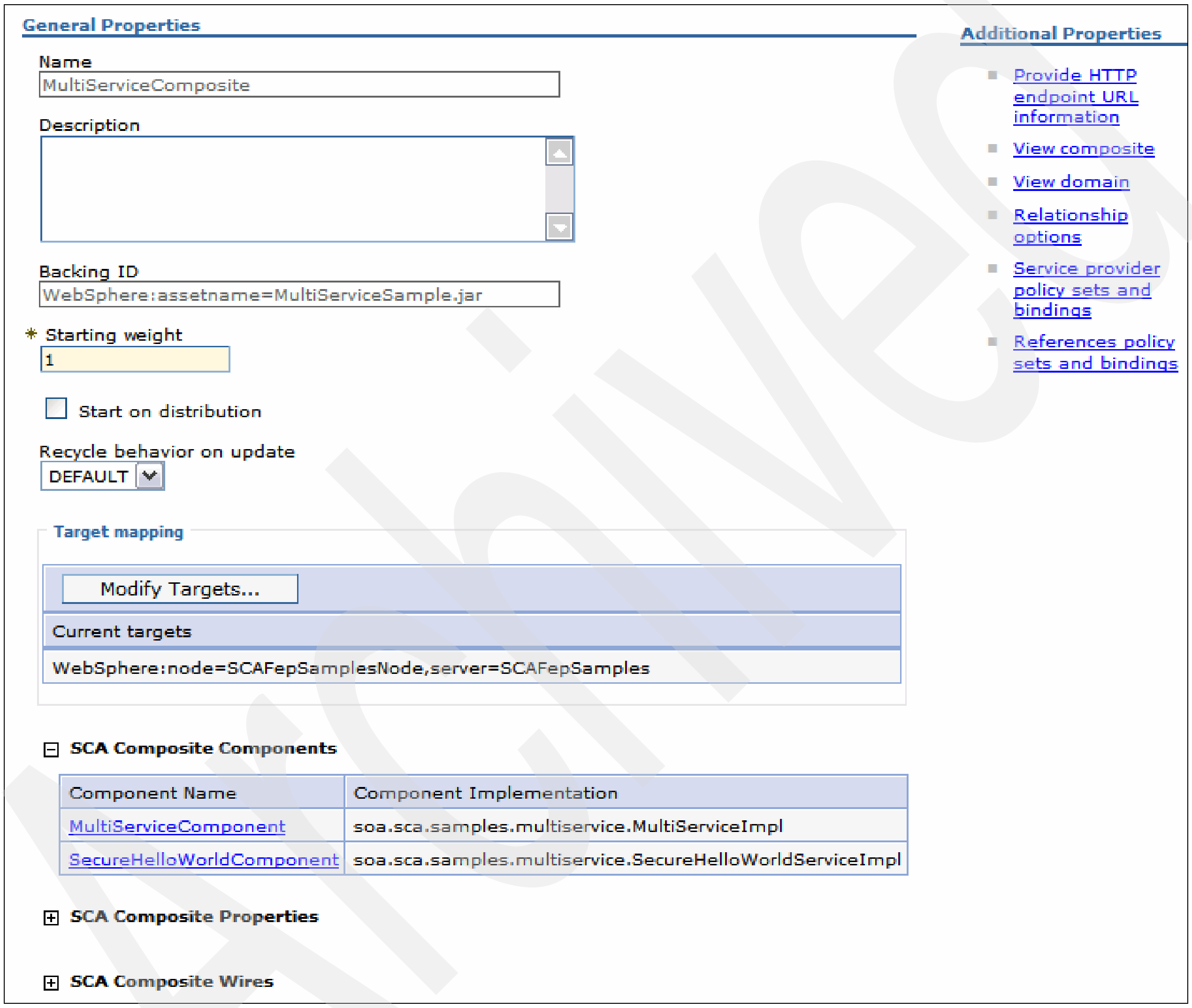
For the purposes of a discussion on SCA business-level applications, this page includes the list of deployed assets. These assets are SCA composites that have been imported to the system and then added to the business-level application as a composition unit.

Clicking the asset name (MultiServiceComposite) opens the composition unit settings.

5.2 Working with composites

The composition unit settings for an SCA composite artifact are similar to those for any artifact, with a few exceptions, which we describe in this section.

To display the SCA composite, select **Applications** → **Application Types** → **Business-level applications** → ***BLA_name*** → **[Deployed assets]** ***composite_name***. The General Properties window opens, as shown in Figure 5-3.



Name: MultiServiceComposite

Description:

Backing ID: WebSphere:assetname=MultiServiceSample.jar

* Starting weight: 1

☐ Start on distribution

Recycle behavior on update: DEFAULT

Target mapping

Current targets

WebSphere:node=SCAFepSamplesNode,server=SCAFepSamples

SCA Composite Components

Component Name	Component Implementation
MultiServiceComponent	soa.sca.samples.multiservice.MultiServiceImpl
SecureHelloWorldComponent	soa.sca.samples.multiservice.SecureHelloWorldServiceImpl

Figure 5-3 MyBank composition unit

The “Target mapping” in this window section allows you to modify the target server (the server on which the composite will run).

At the bottom of the configuration page, you find the following options that are specific to SCA:

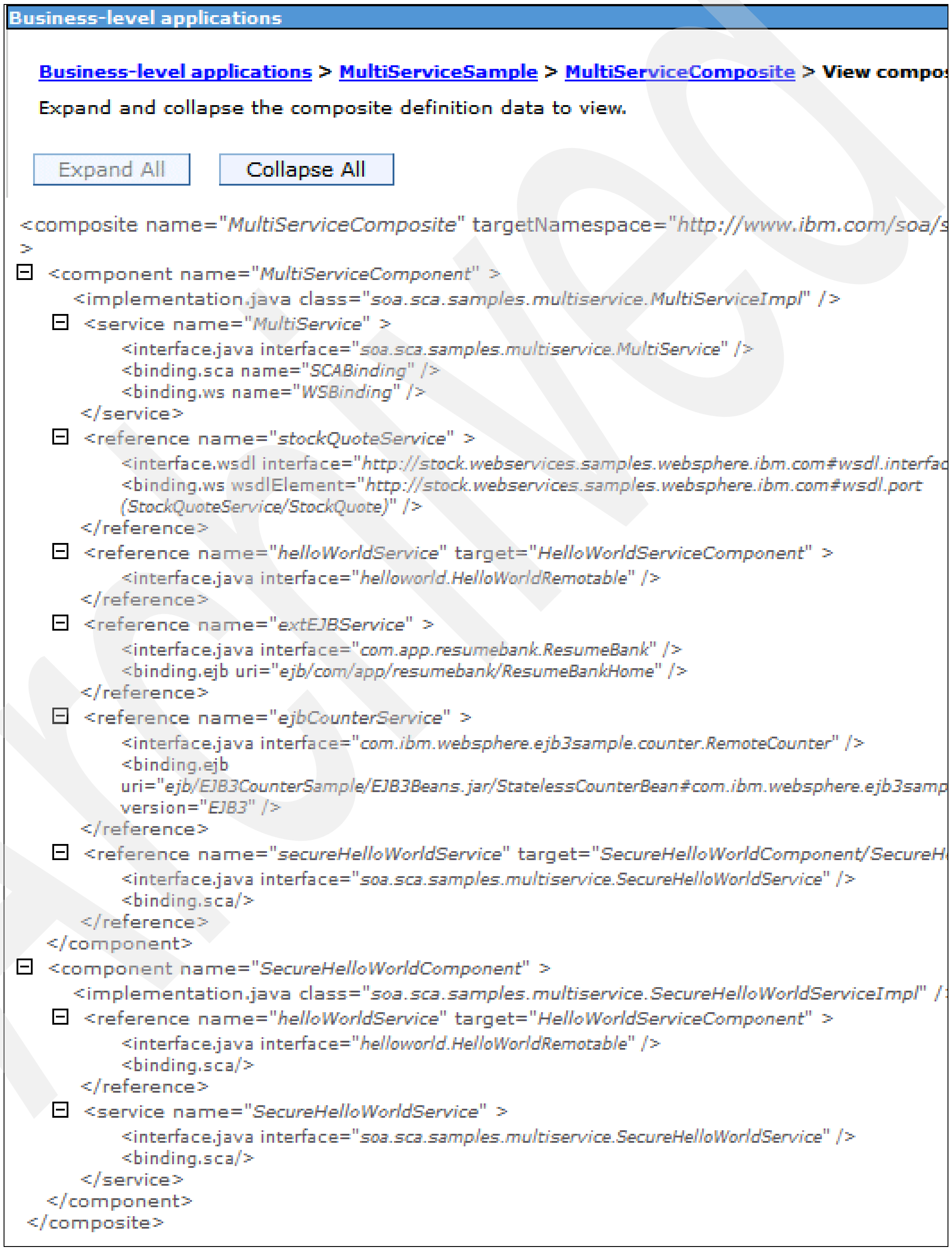
- SCA composite components
- SCA composite properties
- SCA composite wires

Figure 5-3 shows that the MultiServiceComposite composition unit represents the `MultiServiceSample.jar` SCA archive. That SCA archive contains two components:

- MultiServiceComponent
- SecureHelloWorldComponent

On the right of the page, several links are displayed in the “Additional Properties” section. Three of these links are standard links that are specific to SCA:

- The **View composite** link displays the composite XML file (Figure 5-4).


Business-level applications
Business-level applications > MultiServiceSample > MultiServiceComposite > View compos
Expand and collapse the composite definition data to view.
Expand All
Collapse All

```
<composite name="MultiServiceComposite" targetNamespace="http://www.ibm.com/soa/s
>
 ⊟ <component name="MultiServiceComponent" >
     <implementation.java class="soa.sca.samples.multiservice.MultiServiceImpl" />
   ⊟ <service name="MultiService" >
         <interface.java interface="soa.sca.samples.multiservice.MultiService" />
         <binding.sca name="SCABinding" />
         <binding.ws name="WSBinding" />
     </service>
   ⊟ <reference name="stockQuoteService" >
         <interface.wsdl interface="http://stock.webservices.samples.websphere.ibm.com#wsdl.interfac
         <binding.ws wsdlElement="http://stock.webservices.samples.websphere.ibm.com#wsdl.port
         (StockQuoteService/StockQuote)" />
     </reference>
   ⊟ <reference name="helloWorldService" target="HelloWorldServiceComponent" >
         <interface.java interface="helloworld.HelloWorldRemotable" />
     </reference>
   ⊟ <reference name="extEJBService" >
         <interface.java interface="com.app.resumebank.ResumeBank" />
         <binding.ejb uri="ejb/com/app/resumebank/ResumeBankHome" />
     </reference>
   ⊟ <reference name="ejbCounterService" >
         <interface.java interface="com.ibm.websphere.ejb3sample.counter.RemoteCounter" />
         <binding.ejb
         uri="ejb/EJB3CounterSample/EJB3Beans.jar/StatelessCounterBean#com.ibm.websphere.ejb3samp
         version="EJB3" />
     </reference>
   ⊟ <reference name="secureHelloWorldService" target="SecureHelloWorldComponent/SecureH
         <interface.java interface="soa.sca.samples.multiservice.SecureHelloWorldService" />
         <binding.sca/>
     </reference>
   </component>
 ⊟ <component name="SecureHelloWorldComponent" >
     <implementation.java class="soa.sca.samples.multiservice.SecureHelloWorldServiceImpl" /
   ⊟ <reference name="helloWorldService" target="HelloWorldServiceComponent" >
         <interface.java interface="helloworld.HelloWorldRemotable" />
         <binding.sca/>
     </reference>
   ⊟ <service name="SecureHelloWorldService" >
         <interface.java interface="soa.sca.samples.multiservice.SecureHelloWorldService" />
         <binding.sca/>
     </service>
   </component>
</composite>
```

Figure 5-4 View composite

- The **View domain** link displays all the SCA components, services, and references that are deployed within the cell.
- The **Relationship options** link specifies a relationship that declares a dependency relationship to another asset that is deployed as a shared library in the same business-level application. We discuss an example of this concept later in Chapter 9, “Using a Spring implementation as an SCA component” on page 123.

5.2.1 Working with policy sets

If policy sets are attached to the composite at deployment time, you can attach and detach the policy sets and assign policy set bindings from the administrative console. You can find the following links on the General Properties window for the composite (as shown in Figure 5-3 on page 66):

- The **Service provider policy sets and bindings** link allows you to attach policy sets and policy set bindings to a composition unit, service, endpoints, or operations.
- The **References policy sets and bindings** link

You can find more information about working with policy sets in Chapter 7, “Attaching Quality of Service properties” on page 83.

5.2.2 Modifying the host and port prefixes for Web services bindings

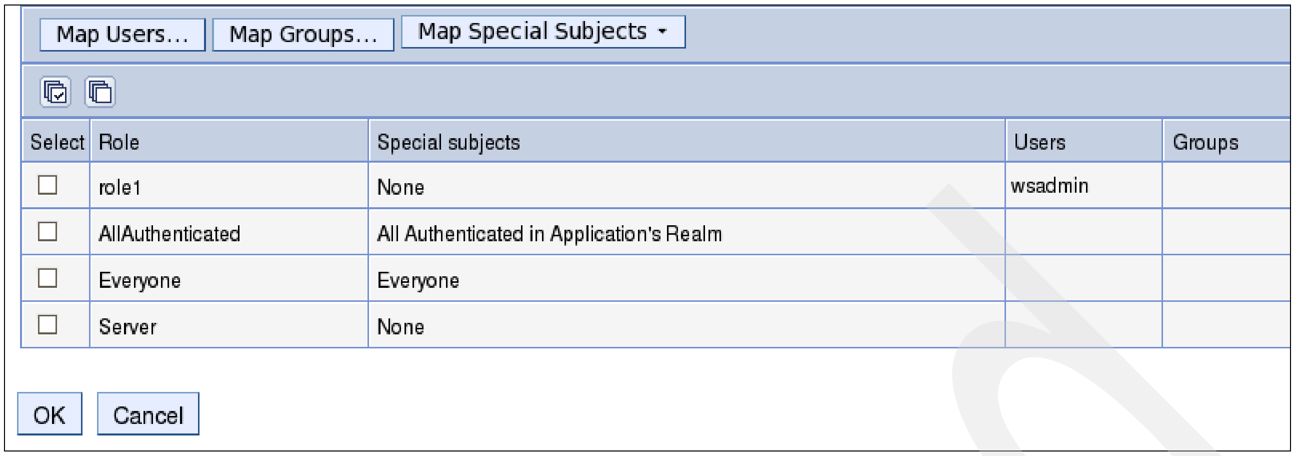
If you have Web services bindings in your composite, you have the **Provide HTTP endpoint URL information** link on the composite General Properties page (as shown in Figure 5-3 on page 66). This link allows you to override the URL host and port information in the binding. This setting is important when you have Web service binding endpoints in a cluster environment where a single proxy server will route the calls to appropriate application server.

5.2.3 Working with security role mappings

If the SCA composite contains a `META-INF/definitions.xml` file in the assets (for declaratively protecting SCA components and operations), a link is available to a configuration page for mapping users and groups to roles.

Select **Applications** → **Application Types** → **Business-level applications** → ***BLA_name*** → **[Deployed assets]** ***composite_name*** → **Map security roles to users or groups**.

Every security role in the `definitions.xml` file is listed in the table, as shown in Figure 5-5 on page 69, allowing you to assign user, group, or special subjects to the role.



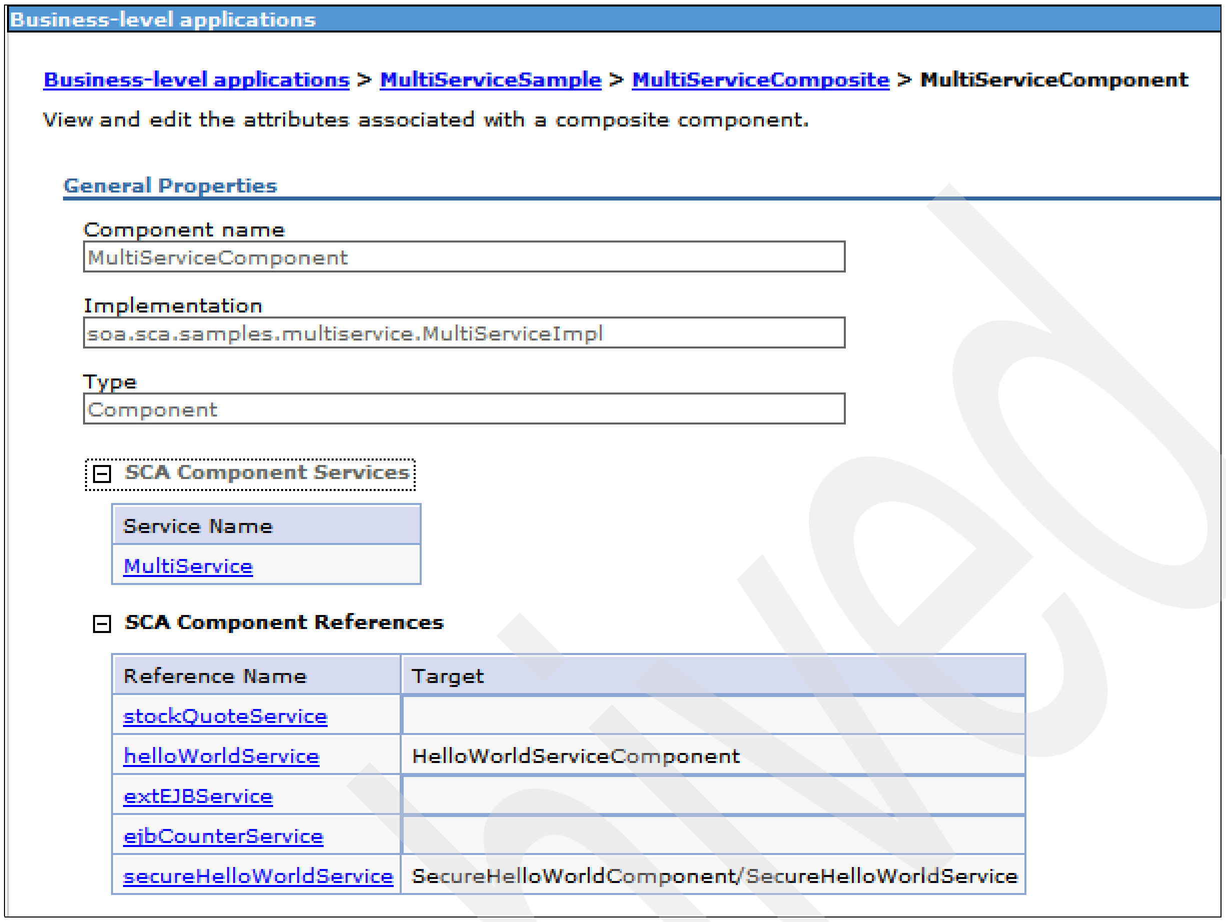
Select	Role	Special subjects	Users	Groups
☐	role1	None	wsadmin	
☐	AllAuthenticated	All Authenticated in Application's Realm		
☐	Everyone	Everyone		
☐	Server	None		

Figure 5-5 Map security roles to users and groups

5.3 Working with components

You can work with individual components in a composite. Links to each component are found in the General Properties window for the composite (as shown in Figure 5-3 on page 66).

First navigate to the composite by selecting **Applications** → **Application Types** → **Business-level applications** → ***BLA_name*** → **[Deployed assets]** ***composite_name***. Then, select the link for the component to open the General Properties window for the component (as shown in Figure 5-6 on page 70). This page contains links for the services and references for the components.


Business-level applications

Business-level applications > MultiServiceSample > MultiServiceComposite > MultiServiceComponent

View and edit the attributes associated with a composite component.

General Properties

Component name: MultiServiceComponent

Implementation: soa.sca.samples.multiservice.MultiServiceImpl

Type: Component

SCA Component Services

Service Name
MultiService

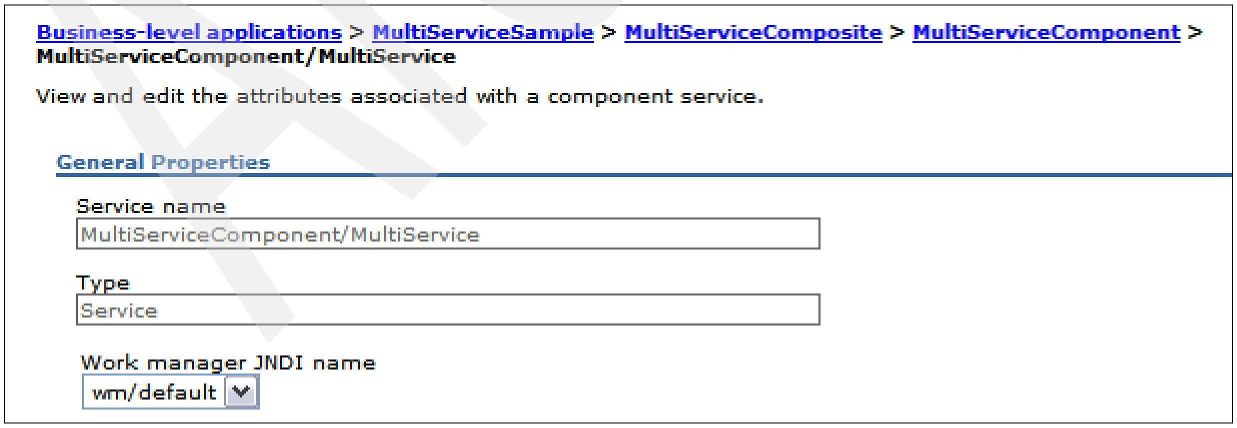
SCA Component References

Reference Name	Target
stockQuoteService	
helloWorldService	HelloWorldServiceComponent
extEJBService	
ejbCounterService	
secureHelloWorldService	SecureHelloWorldComponent/SecureHelloWorldService

Figure 5-6 Component General Properties

5.3.1 Working with component services

You can view the service properties for the component by clicking the link for the service in the SCA component services section (shown in Figure 5-6 on page 70). In this example, we clicked the **MultiService** link, as shown in Figure 5-7.


Business-level applications > MultiServiceSample > MultiServiceComposite > MultiServiceComponent > MultiServiceComponent/MultiService

View and edit the attributes associated with a component service.

General Properties

Service name: MultiServiceComponent/MultiService

Type: Service

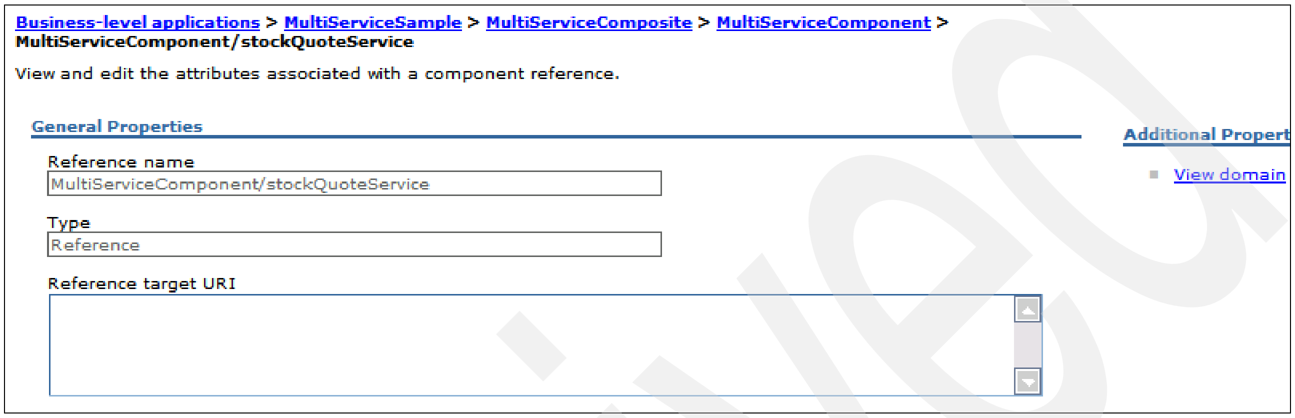
Work manager JNDI name: wm/default

Figure 5-7 Component service settings

5.3.2 Working with component references

When a component declares references in its composite descriptor, the reference can be viewed and potentially modified. You can find links to each reference in the General Properties window for the component (as shown in Figure 5-6 on page 70). The configuration options that you have for a reference depend on the binding type.

Figure 5-8 shows the stockQuoteService reference. We know the reference has a Web services binding (see Figure 5-4 on page 67).


Business-level applications > MultiServiceSample > MultiServiceComposite > MultiServiceComponent > MultiServiceComponent/stockQuoteService

View and edit the attributes associated with a component reference.

General Properties

Reference name: MultiServiceComponent/stockQuoteService

Type: Reference

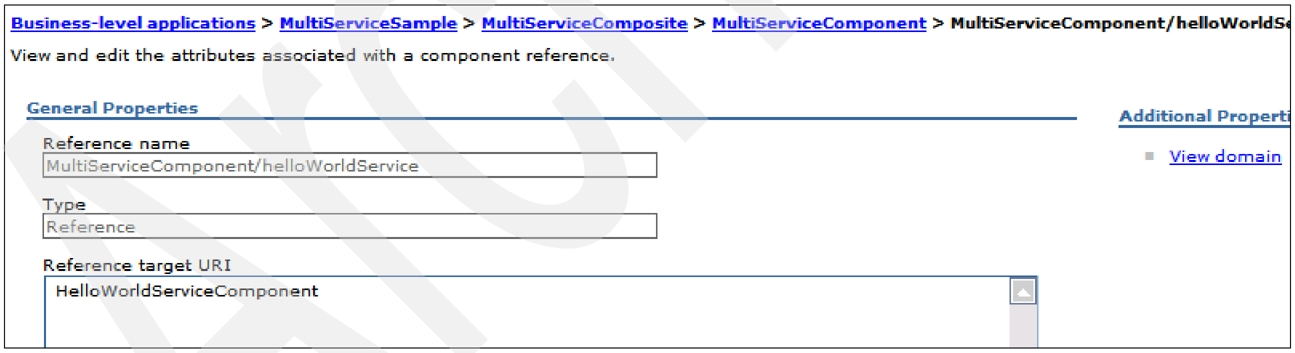
Reference target URI:

Additional Propert

View domain

Figure 5-8 SCA component references

You can use the Reference target URI field to define the target for the reference if the target exists in the same SCA domain. If you are not sure what is available in the SCA domain, you can use the **View Domain** link on the right to view the options. In Figure 5-9, the HelloWorldService reference contains a target URI that points to a component in another composite in the SCA domain.


Business-level applications > MultiServiceSample > MultiServiceComposite > MultiServiceComponent > MultiServiceComponent/helloWorldSe

View and edit the attributes associated with a component reference.

General Properties

Reference name: MultiServiceComponent/helloWorldService

Type: Reference

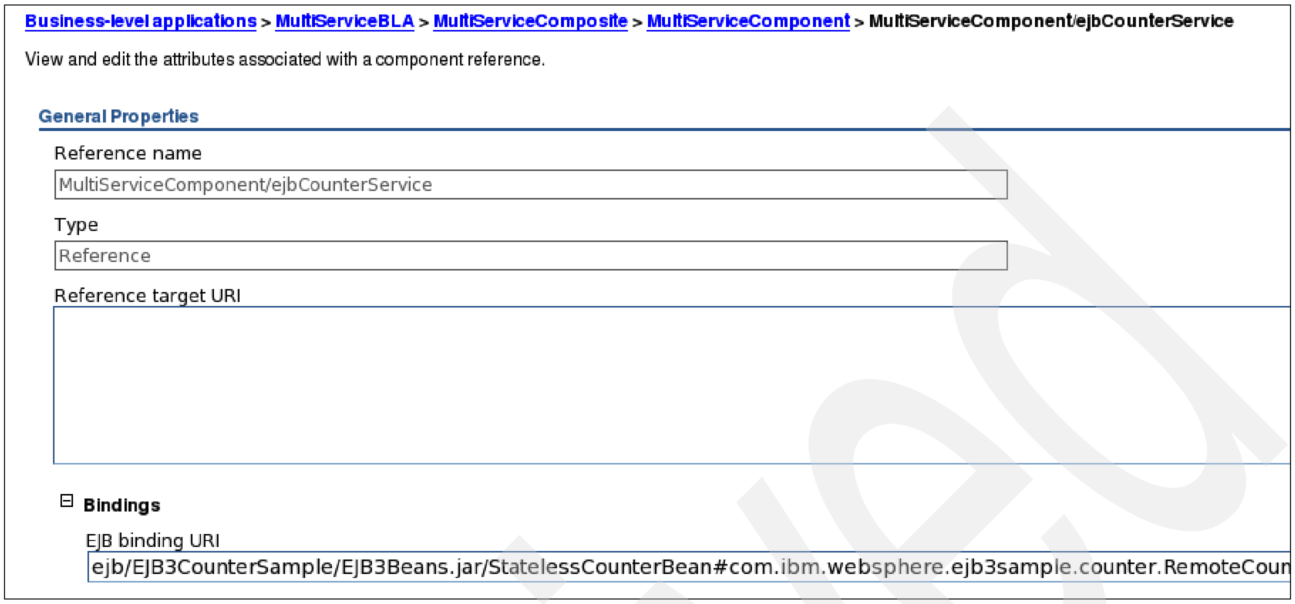
Reference target URI: HelloWorldServiceComponent

Additional Properti

View domain

Figure 5-9 Setting the reference attributes

When the reference has an EJB binding, you have an additional field that allows you to modify the EJB binding URI, as shown in Figure 5-10, for the ejbCounterService reference.


Business-level applications > MultiServiceBLA > MultiServiceComposite > MultiServiceComponent > MultiServiceComponent/ejbCounterService

View and edit the attributes associated with a component reference.

General Properties

Reference name
MultiServiceComponent/ejbCounterService

Type
Reference

Reference target URI

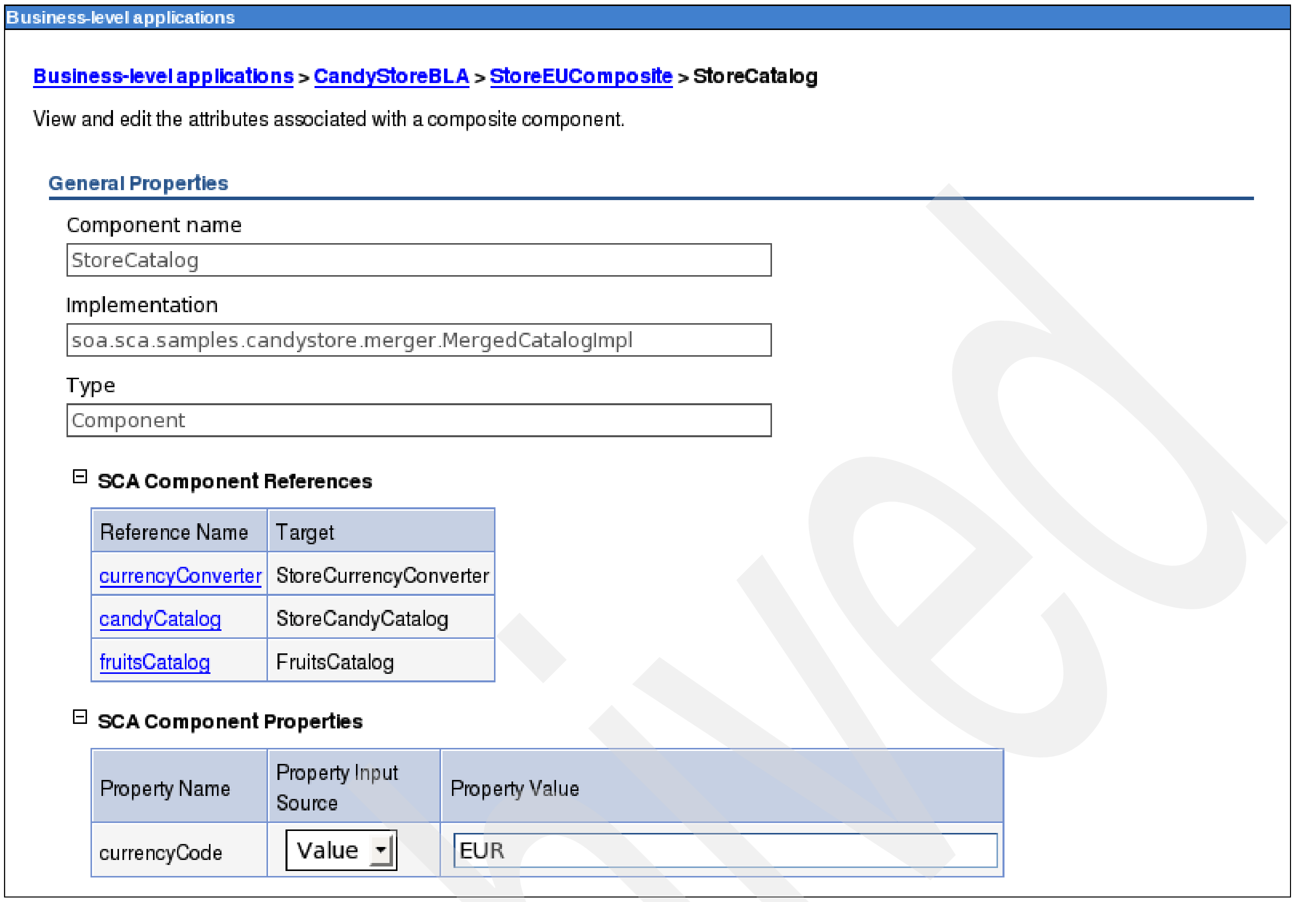
Bindings

EJB binding URI
ejb/EJB3CounterSample/EJB3Beans.jar/StatelessCounterBean#com.ibm.websphere.ejb3sample.counter.RemoteCoun

Figure 5-10 SCA component reference bindings

5.3.3 Working with component properties

For a component with properties that are declared in its composite descriptor, the administrative console allows you to change the value and the value’s input source. Select **Applications** → **Application Types** → **Business-level applications** → ***BLA_name*** → **[Deployed assets]** ***composite_name*** → [SCA Composite Components] ***component_name*** → **SCA Component Properties** (Figure 5-11). If the composite does not have properties declared, the SCA Component Properties section does not display.


Business-level applications

Business-level applications > CandyStoreBLA > StoreEUComposite > StoreCatalog

View and edit the attributes associated with a composite component.

General Properties

Component name
StoreCatalog

Implementation
soa.sca.samples.candystore.merger.MergedCatalogImpl

Type
Component

SCA Component References

Reference Name	Target
currencyConverter	StoreCurrencyConverter
candyCatalog	StoreCandyCatalog
fruitsCatalog	FruitsCatalog

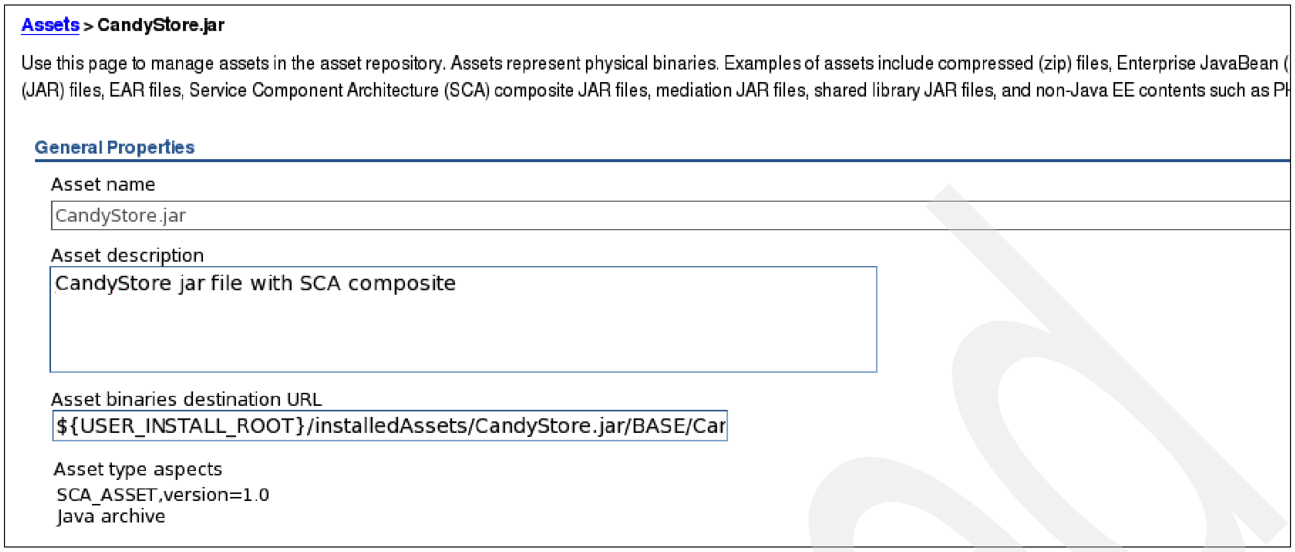
SCA Component Properties

Property Name	Property Input Source	Property Value
currencyCode	Value	EUR

Figure 5-11 SCA component properties

5.4 Viewing assets

You can view the SCA assets that are imported into the system. Select **Applications** → **Application Types** → **Assets**. You can identify SCA assets by looking at the **Asset type aspects** settings. You should see **SCAASSET,version=1.0** for SCA composites, as shown in Figure 5-12 on page 74.


Assets > CandyStore.jar

Use this page to manage assets in the asset repository. Assets represent physical binaries. Examples of assets include compressed (zip) files, Enterprise JavaBean (
(JAR) files, EAR files, Service Component Architecture (SCA) composite JAR files, mediation JAR files, shared library JAR files, and non-Java EE contents such as PH

General Properties

Asset name
CandyStore.jar

Asset description
CandyStore jar file with SCA composite

Asset binaries destination URL
${USER_INSTALL_ROOT}/installedAssets/CandyStore.jar/BASE/Car

Asset type aspects
SCA_ASSET,version=1.0
Java archive

Figure 5-12 Asset properties

6

Administering SCA applications with wsadmin

The administrative console makes administering applications and servers easy. However, there are many times when a scripting method is preferred (for example, when you need to take the same administrative actions across several WebSphere Application Server installations or when the actions will be executed by non-administrators).

In this chapter, we describe the features that are provided by wsadmin for managing SCA applications. The wsadmin scripting tool supports two scripting languages:

- Jython
- JACL

We use Jython in this chapter.

The wsadmin scripting commands are located in the Reference section of the information center:

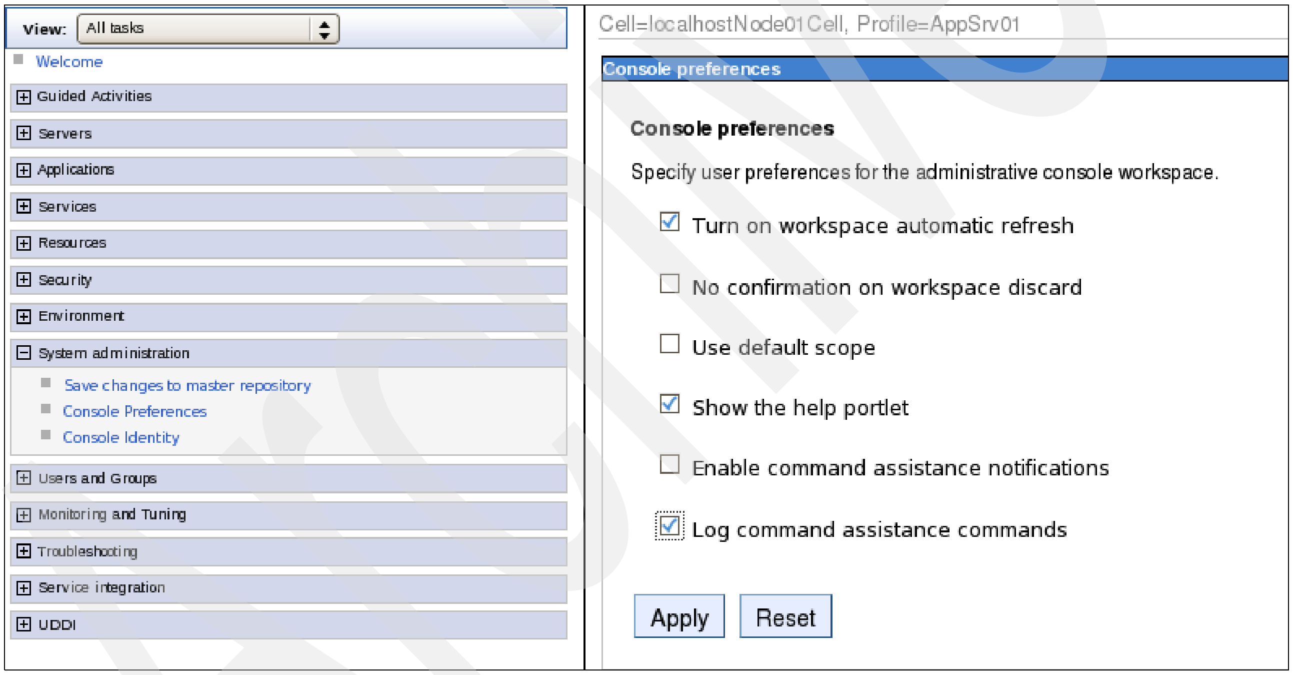
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/welc_ref_adm_script.html

6.1 Using command assistance

Writing wsadmin scripts is not a simple or intuitive task. For this reason, WebSphere Application Server includes a feature called *command assistance* in the administrative console. This feature can help you build wsadmin scripts. The command assistance feature records the commands that are executed as a result of actions that you take in the administrative console. (Note that not all actions have a corresponding wsadmin command recorded.) These commands can be displayed in the console and optionally recorded in a file in the server's log directory. You can copy and paste these commands into your scripts where appropriate.

To use the command assistance feature, you need to make sure the console preferences are set appropriately. Log in to the administrative console and go to **System administration** → **Console Preferences**, as shown in Figure 6-1.

- The command for the current console action is shown in the Help portlet of the console. To see these commands, you need to select **Show help portlet**.
- To also have the commands logged to a file, select **Log command assistance commands**.



Cell=localhostNode01Cell, Profile=AppSrv01

Console preferences

Console preferences

Specify user preferences for the administrative console workspace.

[✓] Turn on workspace automatic refresh
[ ] No confirmation on workspace discard
[ ] Use default scope
[✓] Show the help portlet
[ ] Enable command assistance notifications
[✓] Log command assistance commands

Figure 6-1 Enabling command assistance in the console

With the logging feature enabled, the scripting commands are written to the following log file:

```
app_server_root/profiles/profile_name/logs/server_name/commandAssistanceJython
Commandswasadmin.log
```

Example 6-1 shows an example of viewing the commands written to the log.

Example 6-1 Viewing the scripting commands recorded in a log

```
[wasadmin@localhost AppSrv01]$ pwd
/opt/IBM/WebSphere/AppServer/profiles/AppSrv01
[wasadmin@localhost AppSrv01]$ tail -f
logs/server1/commandAssistanceJythonCommands_wasadmin.log
```

```
# [2/24/10 16:06:14:304 CET] BLAManagement
AdminTask.listBLAs('[-includeDescription true]')
```

This example shows the output of the command assistance file that resulted when **Applications** → **Application Types** → **Business-level applications** was selected in the administrative console. The lines beginning with a hash (#) are comments. The line beginning with AdminTask is the beginning of the commands.

Using the command assist feature is a quick and easy way to work with wsadmin commands,

6.2 Starting and ending a wsadmin session

You can enter the wsadmin commands in command-line mode or record them in a script for execution. You can execute the **wsadmin** command from:

app_server_root/profiles/*profile_name*/bin

Example 6-2 shows how to start wsadmin from a command line on a Linux® system. In this example, note the **-lang** option and its value, **jython**.

Example 6-2 Starting wsadmin

```
[wasadmin@localhost AppSrv01]$ ./bin/wsadmin.sh -conntype SOAP -user wasadmin
-password passw0rd -lang jython
WASX7209I: Connected to process "server1" on node localhostNode01 using SOAP
connector;  The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>
```

You normally use wsadmin to execute commands against an active server. The wsadmin tool also provides the option to work with configuration files directly using the local (disconnected) mode. You invoke the local mode of operation by starting the scripting client using the command-line option **-conntype NONE** or by setting the `com.ibm.ws.scripting.connectiontype=NONE` property in the `wsadmin.properties` file.

To end a session, enter `exit` at the wsadmin input line.

6.3 Using wsadmin for SCA artifacts

This section explains the wsadmin commands that are useful when working with the Feature Pack for SCA. These commands look complex, but remember that we used the command assistance feature to build the commands.

6.3.1 Listing and viewing business-level applications

You can use the **AdminTask.listBLAs** command to list the business-level applications. The following command records when you display the business-level applications in the console:

```
AdminTask.listBLAs('[-includeDescription true ]')
```

Example 6-3 shows the output of this command.

Example 6-3 Listing the business-level applications (unformatted)

```
wsadmin>AdminTask.listBLAs('[-includeDescription true ]')
'WebSphere:blaname=MultiServiceSample\r\n\r\nWebSphere:blaname=query\r\n\r\nWebS
phere:blaname=myhelloWorld\r\n\r\nWebSphere:blaname=JobbankTargetEJBApp\r\n\r\nW
ebSphere:blaname=DefaultApplication\r\n\r\nWebSphere:blaname=EJB3CounterSample\r
\n\r\nWebSphere:blaname=IBMUTC\r\n\r\nWebSphere:blaname=MyBankAccountService\r\n
\r\nWebSphere:blaname=helloworld-ws-client-asynch_war\r\n\r\nWebSphere:blaname=W
ebServicesSamples\r\n\r\nWebSphere:blaname=myhelloWorldClient\r\n\r\nWebSphere:b
laname=ivtApp\r\n\r\nWebSphere:blaname=MyBankClient_war\r\n'
```

The output of this command is unformatted and can be difficult to read. It is a PyString with the elements separated with "`\n\n`". Example 6-4 shows how to use the wsadmin **`print`** command to make the output more readable.

Example 6-4 List business-level applications using the wsadmin print command

```
wsadmin>print AdminTask.listBLAs('[-includeDescription true ]')
WebSphere:blaname=MultiServiceSample
WebSphere:blaname=query
WebSphere:blaname=myhelloWorld
WebSphere:blaname=JobbankTargetEJBApp
WebSphere:blaname=DefaultApplication
WebSphere:blaname=EJB3CounterSample
WebSphere:blaname=IBMUTC
WebSphere:blaname=MyBankAccountService
WebSphere:blaname=helloworld-ws-client-asynch_war
WebSphere:blaname=WebServicesSamples
WebSphere:blaname=myhelloWorldClient
WebSphere:blaname=ivtApp
WebSphere:blaname=MyBankClient_war
wsadmin>
```

To view the properties of a specific business-level application, use the **`viewBLA`** command as shown in Example 6-5.

Example 6-5 Viewing a business-level application

```
wsadmin>print AdminTask.viewBLA('-blaID CandyStoreBLA')
Business-level application options (BLAOptions)
Business-level applications settings.
Business-level application name (name):
[CandyStoreBLA]
Business-level application description (description):
[CandyStore from /opt/IBM/WebSphere/AppServer/samples/SCA/CandyStore/]
Operation done!
```

Managing business-level applications using the scripting library

WebSphere Application Server V7 provides a scripting library that you can use to simplify the use of scripting commands. The Feature Pack for SCA includes the AdminBLA script library to work with business-level applications. You can use the scripts in this library to create, query, and manage your business-level applications. You can run each script individually or combine procedures to create custom automation scripts.

The AdminBLA script procedures are located in the following directory:

```
app_server_root/scriptLibraries/application/V70
```

For more information about using the AdminBLA script procedures, see:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/rxml_7libbla.html

6.3.2 Listing and viewing composition units

SCA composites are imported and deployed to business-level applications as composition units.

List the composition units in a business-level application

You can use the **`AdminTask.listCompUnits`** command to list the composition units in a business-level application, as shown in Example 6-6.

Example 6-6 Viewing composition units

```
wsadmin>print AdminTask.listCompUnits('-blaID CandyStoreBLA')
WebSphere:cuname=StoreEUComposite
```

Display the properties of a composition unit

You can use the **`AdminTask.viewCompUnit`** command to display the properties of a specific composition units. Example 6-7 shows the help information for the command.

Example 6-7 AdminTask.viewCompUnit

```
wsadmin>print AdminTask.help('viewCompUnit')
WASX8006I: Detailed help for command: viewCompUnit
Description: View options for specified a composition unit of a business-level
application.
Target object:   None

Arguments:

*blaID - ID for the business-level application in any of the following forms:

<bla name>; blaname=<bla name>; WebSphere:blaname=<bla name>; or
WebSphere:blaname=<bla name>,blaedition=<bla edition>. The edition does not need
to be specified unless there is more than one edition.

*cuID - ID for the composition unit in any of the following forms:
<cu name>; cuname=<cu name>; WebSphere:cuname=<cu name>; or WebSphere:cuname=<cu
name>,cuedition=<cu edition>. The edition does not need to be specified unless
there is more than one edition.
```

Example 6-8 shows the use of the **`viewCompUnit`** command.

Example 6-8 Viewing composition units

```
wsadmin>print AdminTask.viewCompUnit('-blaID CandyStoreBLA -cuID
StoreEUComposite')
WS Binding For Service (ServiceWSBinding)
... // removed for brevity
```

```
Component Service (ComponentService)
Service at component level
Component Name (componentName):[FruitsCatalog]
Service Name (serviceName):
[Catalog]
Work Manager (serviceWorkManager):
[]
Component Name (componentName):
[StoreCandyCatalog]
Service Name (serviceName):
[Catalog]
Work Manager (serviceWorkManager):
[]
Component Name (componentName):
[StoreCurrencyConverter]
Service Name (serviceName):
[CurrencyConverter]
Work Manager (serviceWorkManager):
[]
Operation done!
wsadmin>
```

List the active SCA composites

You can list the active SCA composites using the **AdminControl.queryNames** command, as shown in Example 6-9.

Example 6-9 Listing the active SCA composites

```
wsadmin>var = AdminControl.queryNames('type=SCAComposite,')
wsadmin>print var
WebSphere:name=HelloJeeScaServiceComposite,process=server1,platform=dynamicproxy,n
ode=localhostNode01,version=7.0.0.7,type=SCAComposite,mbeanIdentifier=HelloJeeScaS
erviceCompositeApp,cell=localhostNode01Cell,spec=1.0
WebSphere:name=HelloServiceTwoWayComposite,process=server1,platform=dynamicproxy,n
ode=localhostNode01,version=7.0.0.7,type=SCAComposite,mbeanIdentifier=HelloService
TwoWayCompositeApp,cell=localhostNode01Cell,spec=1.0
WebSphere:name=JobBank,process=server1,platform=dynamicproxy,node=localhostNode01,
version=7.0.0.7,type=SCAComposite,mbeanIdentifier=JobBankApp,cell=localhostNode01C
ell,spec=1.0
...
```

A single active SCA composite can be queried as shown in Example 6-10.

Example 6-10 Query a single SCA composite

```
wsadmin>storeeucomp=AdminControl.queryNames('name=StoreEUComposite,type=SCAComposi
te,')
wsadmin>storeeucomp
'WebSphere:name=StoreEUComposite,process=server1,platform=dynamicproxy,node=localh
ostNode01,version=7.0.0.7,type=SCAComposite,mbeanIdentifier=StoreEUCompositeApp,ce
ll=localhostNode01Cell,spec=1.0'
```

6.3.3 Viewing SCA domain information

You can view information about components in an SCA domain by using the **AdminTask.exportCompositeToDomain** command to export the domain information to a file, as shown in Example 6-11.

Example 6-11 AdminTask.exportCompositeToDomain

```
wsadmin>AdminTask.exportCompositeToDomain('[-domainName IBM-hostNode04Cel
l -fileName C:/domain_info]')
'IBM-hostNode04Cell exported to C:/domain_info.\n'
wsadmin>
```

Example 6-12 shows an excerpt of the file with the SCA domain information.

Example 6-12 Excerpt of the SCA domain information

```
<?xml version="1.0" encoding="UTF-8"?>
<domain name="IBM-hostNode04Cell">
<component name = "MultiServiceComponent" mapTarget =
"WebSphere:cell=IBM-hostNode04Cell,node=SCAFepSamplesNode,server=SCAFepSamples">
<service name = "MultiService">
<interface.java interface = "soa.sca.samples.multiservice.MultiService"/>
</service>
<reference name = "stockQuoteService" target = ""/>
<reference name = "helloWorldService" target = "HelloWorldServiceComponent"/>
<reference name = "extEJBService" target = ""/>
<reference name = "ejbCounterService" target = ""/>
<reference name = "secureHelloWorldService" target =
"SecureHelloWorldComponent/SecureHelloWorldService"/>
<httpurlendpoints name = "endpoints" uri = ""/>
</component>
...
</domain>
```

6.3.4 Installing SCA business-level applications

A simple set of example wsadmin commands that you can use to manage business-level application encompasses creating an empty business-level application with the **AdminTask.createEmptyBLA** command, importing an asset to the business-level application with the **AdminTask.importAsset** command, and starting it with the **AdminTask.startBLA** command.

For example, to deploy the MyBankAccountService composite to WebSphere Application Server, follow these steps:

1. Create the MyBankAccountService business-level application:

   ```
   AdminTask.createEmptyBLA('[-name MyBankAccountService]')
   ```

2. Import the `MyBankService.jar` SCA archive file as an asset:

   ```
   AdminTask.importAsset('[-storageType FULL -source
   /mnt/hgfs/Desktop/MyBankService.jar ]')
   ```

3. Add the asset to the business-level application as a composition unit:

   ```
   AdminTask.addCompUnit('[-blaID WebSphere:blaname=MyBankAccountService
   -cuSourceID WebSphere:assetname=MyBankService.jar ]')
   ```

4. Save the configuration:

   ```
   AdminConfig.save()
   ```

5. Start the application:

   ```
   AdminTask.startBLA('[-blaID WebSphere:blaname=MyBankAccountService ]')
   ```

Example 6-13 shows a wsadmin session that executes these commands.

Example 6-13 Installing MyBankAccountService composite

```
[wasadmin@localhost AppSrv01]$ ./bin/wsadmin.sh -conntype SOAP -user wasadmin
-password passw0rd -lang jython
WASX7209I: Connected to process "server1" on node localhostNode01 using SOAP
connector;  The type of process is: UnManagedProcess
WASX7031I: For help, enter: "print Help.help()"
wsadmin>AdminTask.createEmptyBLA('[-name MyBankAccountService]')
'WebSphere:blaname=MyBankAccountService'
wsadmin>AdminTask.importAsset('[-storageType FULL -source
/mnt/hgfs/jacek/Desktop/MyBankService.jar ]')
'WebSphere:assetname=MyBankService.jar'
wsadmin>AdminTask.addCompUnit('[-blaID WebSphere:blaname=MyBankAccountService
-cuSourceID WebSphere:assetname=MyBankService.jar ]')
'WebSphere:cuname=MyBank'
wsadmin>AdminConfig.save()
''
wsadmin>AdminTask.startBLA('[-blaID WebSphere:blaname=MyBankAccountService ]')
'CWWMH0196I: Business-level application "WebSphere:blaname=MyBankAccountService"
was started successfully.'
wsadmin>quit
```

To uninstall the same application, use the following commands:

1. Delete the composition unit:

   ```
   AdminTask.deleteCompUnit('[-blaID WebSphere:blaname=MyBankAccountService -cuID
   WebSphere:cuname=MyBank ]')
   ```

2. Delete the business-level application:

   ```
   AdminTask.deleteBLA('[-blaID WebSphere:blaname=MyBankAccountService ]')
   ```

3. Delete the asset:

   ```
   AdminTask.deleteAsset('[-assetID WebSphere:assetname=MyBankService.jar ]')
   ```

4. Save the configuration:

   ```
   AdminConfig.save()
   ```

7

Attaching Quality of Service properties

The term *Quality of Service* (or QoS) describes capabilities or constraints that can be applied to service components or to the interactions between service components that are represented by services and references. QoS is not a new concept and has been used to describe the configuration of Web services in WebSphere Application Server.

The Web services Feature Pack for v6.1 introduced proprietary configuration of Web services through policy sets. The policy sets are static configuration settings that are applied to your Web services when you deploy them. They are proprietary but provide a simple way to attach a common configuration to a number of services. WebSphere Application Server comes packaged with a number of default policy sets, for example, WSI-RSP is the Reliable Secure Profile and consists of WS-ReliableMessaging, WS-Addressing, WS-SecureConversation and WS-Security. When attached to a Web service, those settings apply to all accesses to that service. For more information, see:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.express.iseries.doc/info/iseriesexp/ae/cwbs_wsspsramp.html

This chapter discusses the use of intents and policy sets to apply qualities of service to SCA applications.

7.1 SCA Policy Framework

The SCA Policy Framework specification is comprised of intents and policy sets. *Intents* represent abstract assertions about a QoS or, in other words, *intentions*. *Policy sets* contain concrete policies that can be applied to SCA bindings and implementations. In SCA, the term *interaction polices* is used to define the QoS applied to the run time interactions between services and references, for example by requiring the encryption of messages between service and reference. The term *implementation policies* is used to define how components behave within their runtime container, for example by requiring that the component run within a global transaction. You can find a good list of these policies at:

http://www.ibm.com/developerworks/websphere/library/techarticles/0902_beck/0902_beck.html

7.1.1 Intents

Typically, a developer will add SCA intents to one or more SCA elements in the composite file. These intents are then bundled with the SCA application. At deployment, the deployer chooses the concrete implementations of those intentions. This model aids and promotes the re-use of components because the actual decisions regarding QoS can be made based on the context of the deployment.

The Feature Pack for SCA v1.0.1 supports the following intents:

- managedTransaction
- managedTransaction.global
- managedTransaction.local
- noManagedTransaction
- propagatesTransaction
- suspendsTransaction
- transactedOneWay
- immediateOneWay
- authentication
- authentication.message
- authentication.transport
- confidentiality
- confidentiality.message
- confidentiality.transport
- integrity
- integrity.message
- integrity.transport
- atLeastOnce
- atMostOnce
- ordered
- exactlyOnce

An SCA intent is represented by the `requires` attribute, as shown in Example 7-1.

Example 7-1 Adding an intent to a component

```
<component name="SampleComponent">
      <implementation.java class="example.SampleServiceImpl"
             requires="managedTransaction.global"/>
</component>
```

When using the Web services binding, the intent will look like Example 7-2.

Example 7-2 Adding an intent to a Web services binding

```
<reference name="sampleReference">
        <binding.ws requires="confidentiality.message"
          ...
/>
</reference>
```

7.1.2 Policy sets

In contrast to the abstract nature of Intents, in the SCA Policy Framework a policySet element is used to define a set of concrete policies that apply to a binding or implementation type.

Note: The concept of policy sets in SCA is separate from the concept of WebSphere Application Server policy sets.

In the Feature Pack for SCA v1.0.1, the SCA policy sets are used to control access to the SCA component according to the authorization policy. To understand more about SCA policy sets, refer to the *Using SCA authorization and security identity policies* article in the information center at:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/tsec_authsoa_policy.html

WebSphere Application Server policy sets refers to the pre-configured policy sets that are provided with WebSphere Application Server V7. The Feature Pack for SCA supports configuring these policy sets with SCA services using Web services bindings.

7.1.3 WebSphere Application Server policy sets

When using the Web services binding, the service or reference must also have a WebSphere Application Server policy set attached to ensure that the WebSphere run time provides the QoS that is indicated by the intent, which can be accomplished in several ways:

- Using the administrative console panels when deploying the application.
- Using wsadmin commands when deploying the application.
- Using the proprietary WebSphere Application Server policy set attributes in the composite file.
- Using the proprietary WebSphere Application Server policy set attributes using Rational Application Developer 7.5 SCA tools.

You can add the following WebSphere Application Server proprietary policy sets attributes to the composite SCDL file:

- was:wsPolicySet
- was:wsServicePolicySetBinding
- was:wsReferencePolicySetBinding

The `was` portion of these attributes is a namespace prefix that is mapped to the following location:

```
http://www.ibm.com/xmlns/prod/websphere/sca/1.0/2007/06
```

If you attach these attributes to the composite file, the named policy set is attached to the application at deployment (providing that it exists in the deployment environment).

A large list of policy types and policy sets are provided with WebSphere Application Server that you can extend. See the following item in the information center for an up-to-date list.

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.base.doc/info/aes/ae/cwbs_wsspsps.html

In the composite file, a policy set attribute is added as shown in Example 7-3.

Example 7-3 Adding a policy set to a reference binding

```
<reference name="AccountServiceReference">
         <binding.ws was:wsPolicySet="WSHTTPS default"
               ...
         />
</reference>
```

Policy set bindings

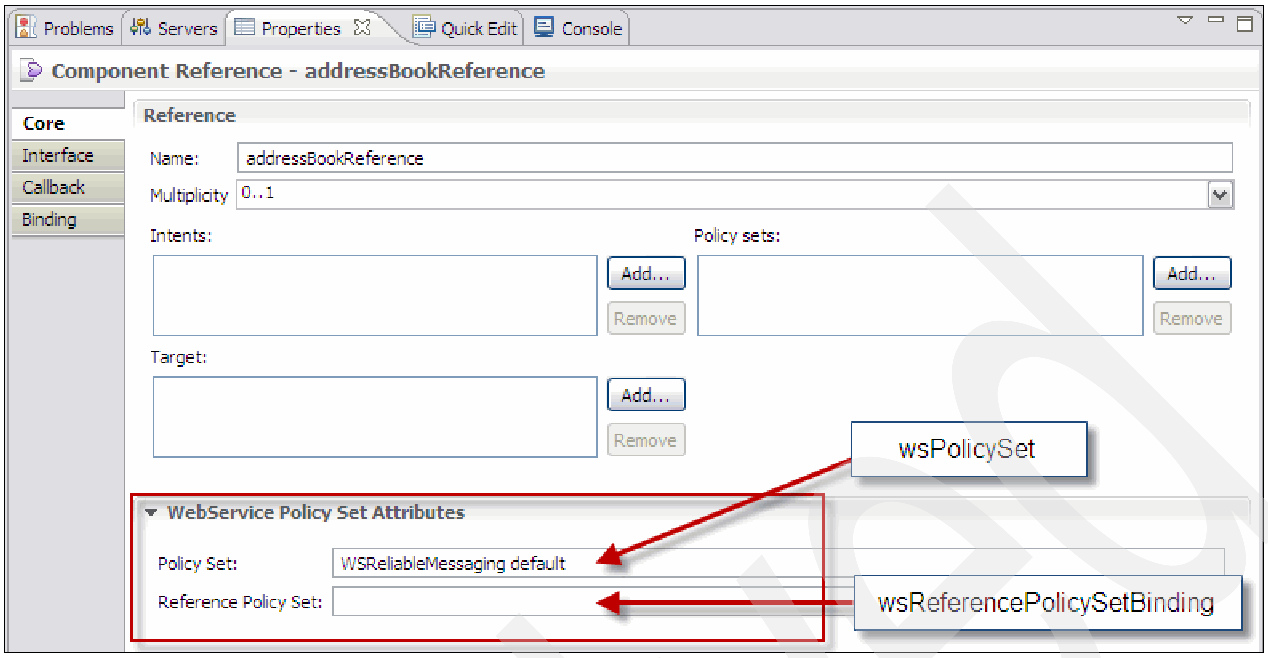
If you need WebSphere Application Server policy set bindings, you can use wsServicePolicySetBinding or wsReferencePolicySetBinding to refer to a general provider policy set binding or general client policy set binding. Alternatively, you can create application level bindings post-deployment using the same administrative capabilities that you use for a JAX-WS application.

The following information center article contains information about WebSphere Application Server Policy set bindings:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/ae/cwbs_wsspsbind.html

Attaching policy sets in the composite file

In Rational Application Developer 7.5 SCA tools, you can use the Properties tab when working with your composite file to indicate policy sets to attach. In Figure 7-1, we have attached the WSReliableMessaging default policy set to a reference. This particular policy set does not require a `bindings.xml` file, so there is no need to enter a value in the Reference Policy Set field.



Component Reference - addressBookReference

Reference

Name: addressBookReference

Multiplicity 0..1

Intents:

Policy sets:

Target:

WebService Policy Set Attributes

Policy Set: WSReliableMessaging default

Reference Policy Set:

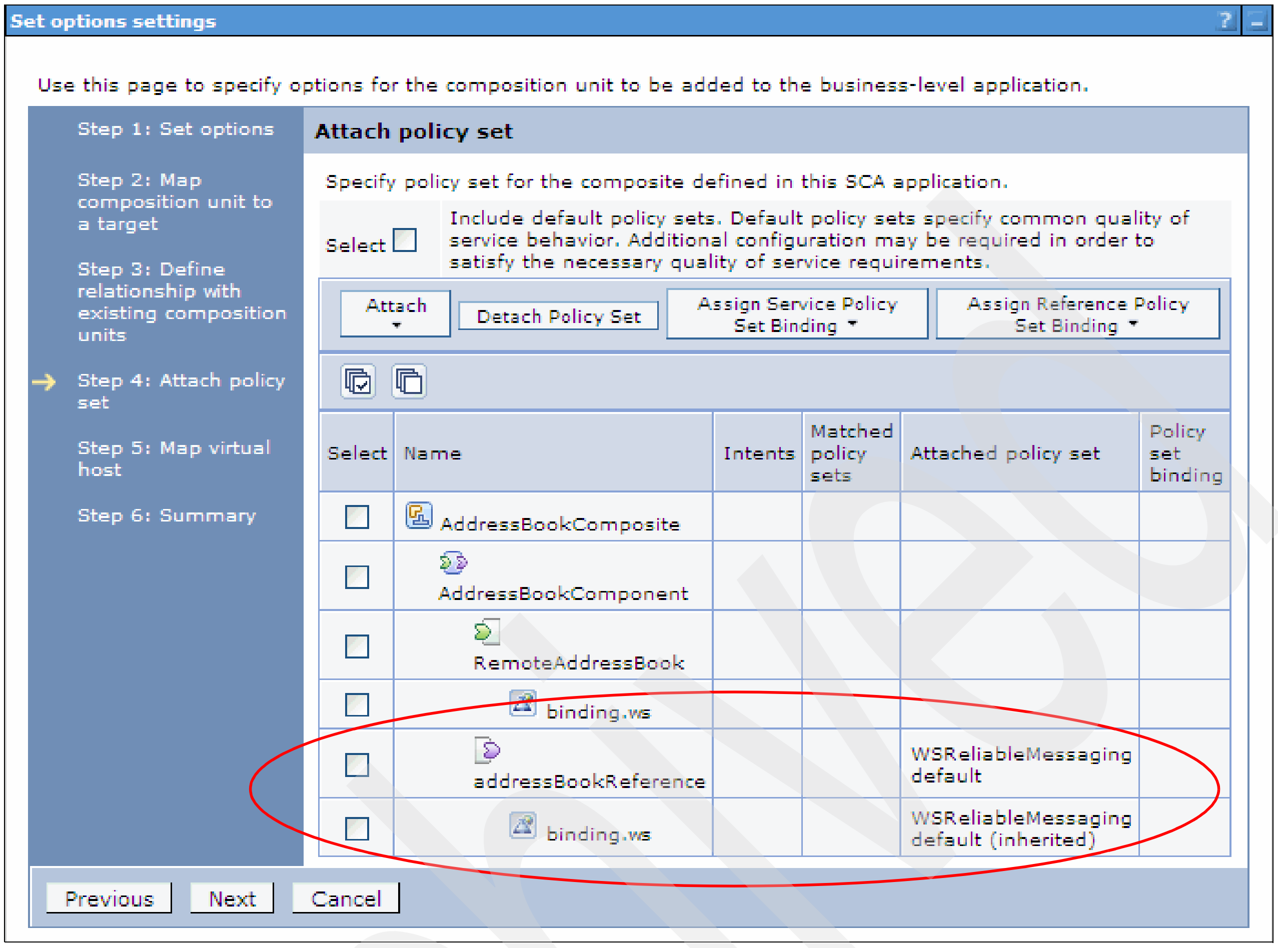
Figure 7-1   Setting the policy set attributes

Attaching the policy set, as shown in Figure 7-1, results in the addition of the `was:wsPolicySet` attribute to the reference in the composite file (Example 7-4).

Example 7-4   Policy set attributes on a reference

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
xmlns:was="http://www.ibm.com/xmlns/prod/websphere/sca/1.0/2007/06"
autowire="false" name="AddressBookComposite" targetNamespace="http://addressbook">
  <component name="AddressBookComponent">
      <implementation.java class="com.remoteutils.RemoteAddressBookImpl"/>
      <service name="RemoteAddressBook">
          <interface.java interface="com.remoteutils.RemoteAddressBook"/>
          <binding.ws/>
      </service>
      <reference name="addressBookReference" was:wsPolicySet="WSReliableMessaging
default">
          <interface.wsdl
interface="http://addressbook.com/#wsdl.interface(AddressBook)"/>
          <binding.ws
wsdlElement="http://addressbook.com/#wsdl.service(AddressBookService)"/>
      </reference>
  </component>
</composite>
```

When you deploy this composite, WebSphere Application Server detects the attached policy set automatically. You see this in the administrative console when you add the composite asset to the business-level application, as shown in Figure 7-2 on page 88.


Set options settings

Use this page to specify options for the composition unit to be added to the business-level application.

Step 1: Set options
Step 2: Map composition unit to a target
Step 3: Define relationship with existing composition units
Step 4: Attach policy set
Step 5: Map virtual host
Step 6: Summary

Attach policy set

Specify policy set for the composite defined in this SCA application.

Select ☐ Include default policy sets. Default policy sets specify common quality of service behavior. Additional configuration may be required in order to satisfy the necessary quality of service requirements.

Attach
Detach Policy Set
Assign Service Policy Set Binding
Assign Reference Policy Set Binding

Select	Name	Intents	Matched policy sets	Attached policy set	Policy set binding
☐	AddressBookComposite				
☐	AddressBookComponent				
☐	RemoteAddressBook				
☐	binding.ws				
☐	addressBookReference			WSReliableMessaging default	
☐	binding.ws			WSReliableMessaging default (inherited)	

Figure 7-2 Displaying the policy sets for the composite in the administrative console

7.1.4 Deployment and runtime enforcement

Intents that are added to the composite file by the developer require runtime configuration by the deployer who attaches the policy sets. In the majority of cases, the intent attribute behaves as a hint to the deployer. It allows the deployer to chose the correct policy set according to the context in which the application is being deployed. It is permissible to *not* attach a policy set and to ignore the intent. There is no runtime validation or enforcement of the intents.

The exception to this rule is when using the transaction intents on an implementation or interaction (but not using them within the binding.ws). In this case, you do not need to attach a concrete policy set because this policy set will get picked up by WebSphere Application Server automatically.

However, if you are using the binding.ws and require WS-AtomicTransaction (WS-AT) protocol over the Web service interaction, then you must either set the `was:wsPolicySet` attribute to `WSTransaction` or attach the policy set to the binding through the administrative console.

The example reference shown in Example 7-5 uses WS-AT.

Example 7-5 Adding the WSTransaction policy set to a reference

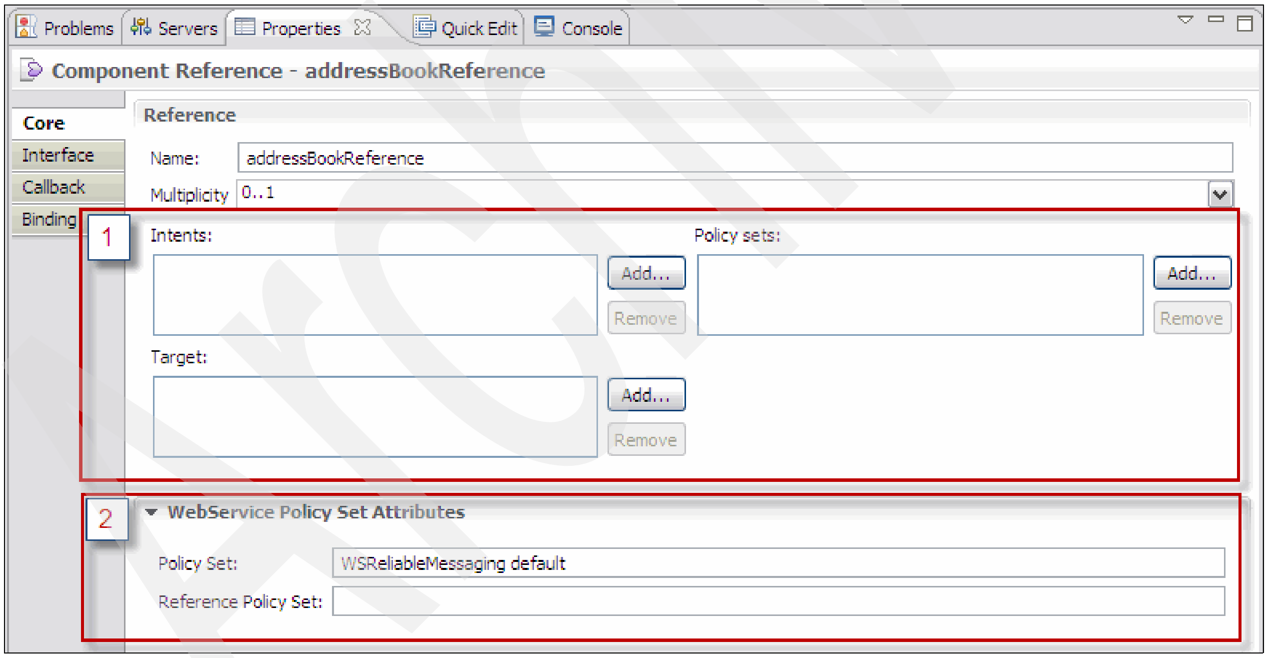
```
<reference name="MyReference"
xmlns:was="http://www.ibm.com/xmlns/prod/websphere/sca/1.0/2007/06"
was:wsPolicySet="WSTransaction">
```

To learn more about transaction intents, see the following article in the information center:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/csca_global_trans.html

7.2 Rational Application Developer support for intents and policy sets

If you are using the SCA tools in Rational Application Developer 7.5, then you will see the SCA intents, SCA policy sets, and WebSphere Web service policy sets from within the Properties tab of the composite file as shown in Figure 7-3. The area marked “1” provides the fields that are used to apply SCA intents and SCA policy sets, and the area marked “2” provides the fields for the WebSphere Application Server proprietary policy set information. We will discuss these areas further as we work though an example.



Name: addressBookReference

Multiplicity 0..1

Policy Set: WSReliableMessaging default

Reference Policy Set:

Figure 7-3 Policy set and intent properties for a composite as seen in the SCA tools

7.3 Intent and policy set examples

In this section, we look at some examples. The first example shows how to configure an SCA component to access an existing JAX-WS Web service that is configured to use the WS-ReliableMessaging default policy set. We also show how to test this configuration using Rational Application Developer Web Services Explorer.

The second example looks at attaching transaction intents and mapping them to policy sets.

7.3.1 Example 1: Configuring components to use Web services policy sets

This example re-uses a previously configured Web service that supports the WS-ReliableMessaging protocol. The Web service was configured to use the WS-ReliableMessaging default policy set. This policy set delivers in-memory support for WS-ReliableMessaging, providing automatic retry in the case of wire failures.

For a detailed explanation of Web services policy sets, see *IBM WebSphere Application Server V7.0 Web Services Guide*, SG24-7758.

The Web service that we use is based on a JAX-WS tutorial application found in the samples and tutorials that ship with Rational Application Developer v7.5. It is called *jwsAddressBook*.

SCA component description

The AddressBookComponent SCA component, shown in Figure 7-4, is built to delegate requests to the jwsAddressBook Web service.

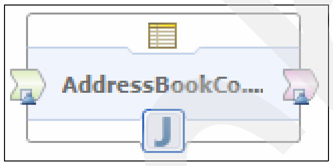


Figure 7-4 RemoteAddressBook component

The component is implemented with the RemoteAddressBook class and contains the RemoteAddressBook interface (shown in Example 7-6) that matches the Web service and imports classes generated from the jwsAddressBook's WSDL file. In this example, notice the `@Remotable` annotation that is required by SCA.

Example 7-6 RemoteAddressBook interface

```
package com.remoteutils;
import org.osoa.sca.annotations.Remotable;
import com.addressbook.AddressType;
import com.addressbook.FindAddressFault;
import com.addressbook.PersonType;
@Remotable
public interface RemoteAddressBook {
      public boolean saveAddress(PersonType person);
      public AddressType findAddress(String name) throws FindAddressFault ;
}
```

The component implementation (Example 7-7) contains an important `@Reference` annotation that provides an instance of the AddressBook type from the Web service. It is important that the name matches the name of the reference in the composite file.

Example 7-7 RemoteAddressBookImpl implementation

```
package com.remoteutils;
import com.addressbook.AddressBook;
import com.addressbook.AddressType;
import com.addressbook.FindAddressFault;
import com.addressbook.PersonType;
import org.osoa.sca.annotations.Reference;
import org.osoa.sca.annotations.Service;
@Service (RemoteAddressBook.class)
public class RemoteAddressBookImpl implements RemoteAddressBook {
      @Reference
      public AddressBook addressBookReference;

      public boolean saveAddress(PersonType person) {
              return addressBookReference.saveAddress(person);
      }
      public AddressType findAddress(String name) throws FindAddressFault {
              return addressBookReference.findAddress(name);
      }
}
```

Example 7-8 shows the composite file.

Example 7-8 AddressBookComposite

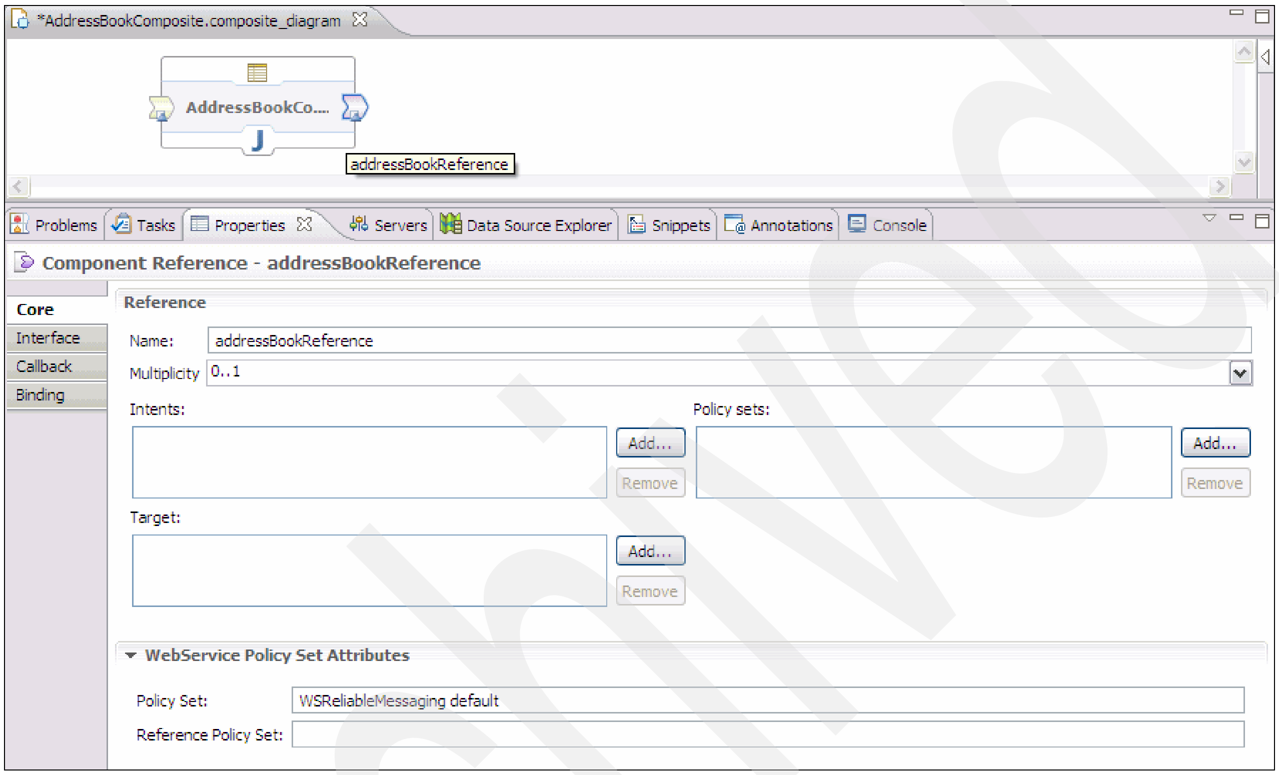
```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
xmlns:was="http://www.ibm.com/xmlns/prod/websphere/sca/1.0/2007/06"
autowire="false" name="AddressBookComposite" targetNamespace="http://addressbook">
  <component name="AddressBookComponent">
      <implementation.java class="com.remoteutils.RemoteAddressBookImpl"/>
      <service name="RemoteAddressBook">
          <interface.java interface="com.remoteutils.RemoteAddressBook"/>
          <binding.ws/>
      </service>
      <reference name="addressBookReference"
         was:wsPolicySet="WSReliableMessaging default">
         <interface.wsdl
            interface="http://addressbook.com/#wsdl.interface(AddressBook)"/>
         <binding.ws
          wsdlElement="http://addressbook.com/#wsdl.service(AddressBookService)"/>
      </reference>
  </component>
</composite>
```

The component has a service with a binding.ws. You can test it using the Rational Application Developer 7.5 Web Services Explorer.

Working with the reference properties using the SCA tools

Looking at this component in Rational Application Developer 7.5, you can see view and modify these settings. Open the composite file in the workspace, and then open the Properties view. You can individually select the service, reference, and component in the work area to view or change the properties on each.

Figure 7-5 shows the Core tab of the Properties for the reference.



Component Reference - addressBookReference

Reference

Name: addressBookReference

Multiplicity 0..1

Intents:

Policy sets:

Target:

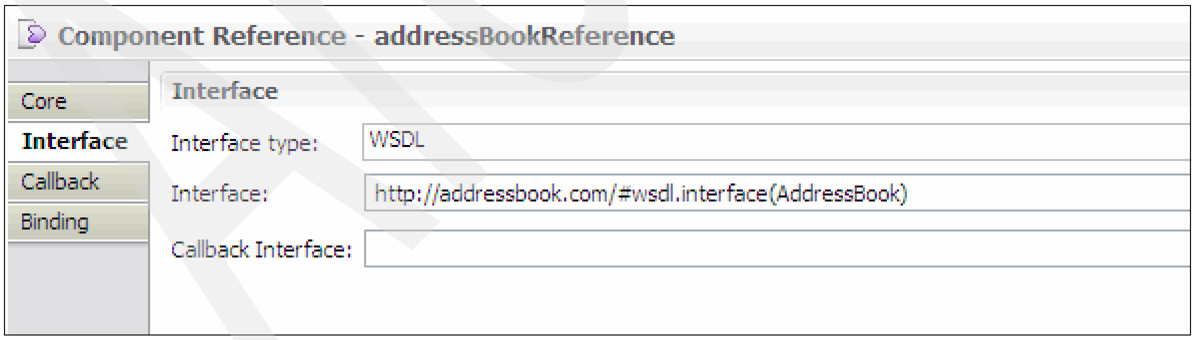
WebService Policy Set Attributes

Policy Set: WSReliableMessaging default

Reference Policy Set:

Figure 7-5 addressBookReference Core property settings

Figure 7-6 shows the properties for the reference in the Interface tab.



Component Reference - addressBookReference

Interface

Interface type: WSDL

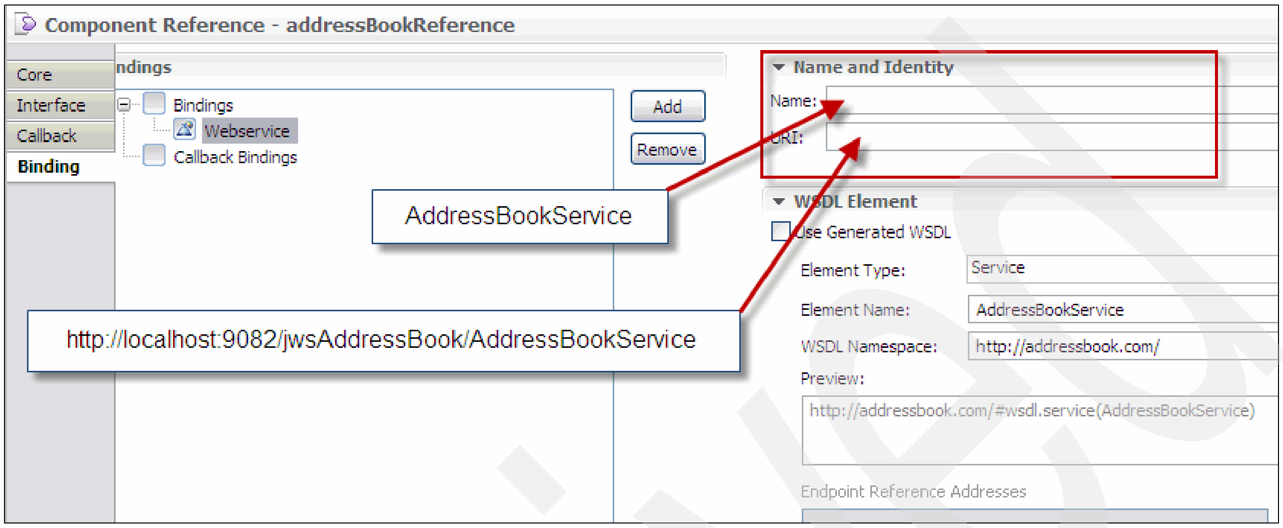
Interface: http://addressbook.com/#wsdl.interface(AddressBook)

Callback Interface:

Figure 7-6 addressBookReference Interface property settings

Figure 7-7 shows the properties for the reference in the Core tab.

We have omitted the optional Name and Identity values here. These values act as a help to the run time to resolve the correct Web service but are not necessary. You need to update the values shown in this example to reflect your `host:port` values.


Component Reference - addressBookReference

Core | Interface | Callback | Binding

ndings

Bindings
Webservice
Callback Bindings

Add
Remove

AddressBookService

http://localhost:9082/jwsAddressBook/AddressBookService

Name and Identity
Name:
URI:

WSDL Element
Use Generated WSDL
Element Type: Service
Element Name: AddressBookService
WSDL Namespace: http://addressbook.com/
Preview:
http://addressbook.com/#wsdl.service(AddressBookService)

Endpoint Reference Addresses

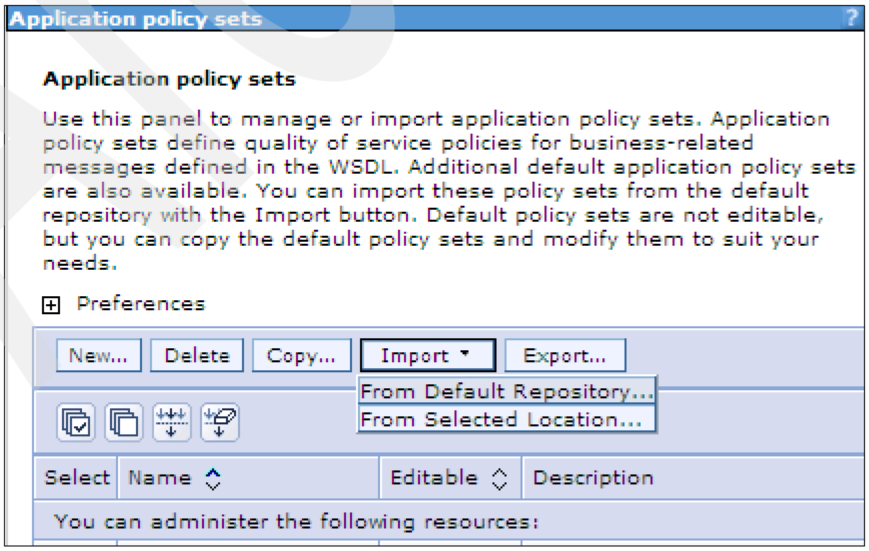
Figure 7-7 addressBookReference Binding property settings

Make the policy set available to the WebSphere run time

If you attach a policy set to a composite, you need to make sure that the policy is available to the WebSphere run time before adding that asset to a business-level application. In this example, we make the WSReliableMessaging default policy available.

To make the policy set available to the WebSphere run time, follow these steps in the administrative console:

1. Select **Services** → **Policy Sets** → **Application policy sets**.
2. Check the list of policy sets to see if the one you want is there. If not, click **Import**, and select **From Default Repository**, as shown in Figure 7-8.


Application policy sets

Application policy sets

Use this panel to manage or import application policy sets. Application policy sets define quality of service policies for business-related messages defined in the WSDL. Additional default application policy sets are also available. You can import these policy sets from the default repository with the Import button. Default policy sets are not editable, but you can copy the default policy sets and modify them to suit your needs.

Preferences

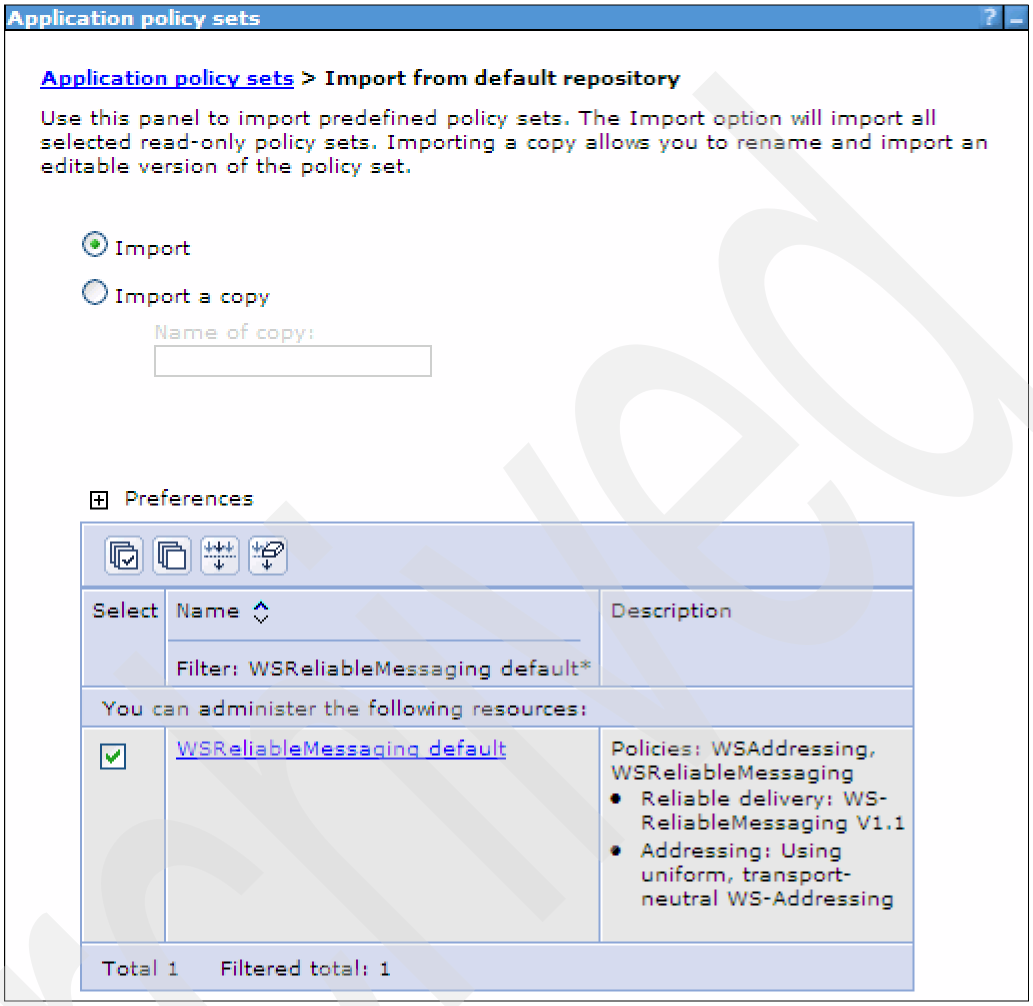
New... | Delete | Copy... | Import | Export...

From Default Repository...
From Selected Location...

Select	Name	Editable	Description
You can administer the following resources:			

Figure 7-8 Import the policy set

3. Find the policy set. (The resulting lists includes multiple entries, so be sure to go to the next page if you do not find the policy set for which you are looking.) Select the box to the left of the policy, as shown in Figure 7-9, and click **OK**.



Application policy sets > **Import from default repository**

Use this panel to import predefined policy sets. The Import option will import all selected read-only policy sets. Importing a copy allows you to rename and import an editable version of the policy set.

[✓ Import]
Import a copy
Name of copy:

Select	Name	Description
	Filter: WSReliableMessaging default*	
You can administer the following resources:		
✓	WSReliableMessaging default	Policies: WSAddressing, WSReliableMessaging • Reliable delivery: WS-ReliableMessaging V1.1 • Addressing: Using uniform, transport-neutral WS-Addressing
Total 1	Filtered total: 1	

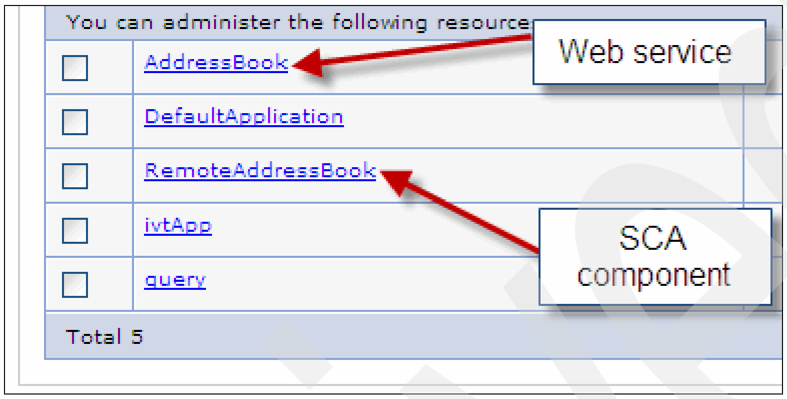
Figure 7-9 Import the policy set

4. Save the configuration.
5. You can now add the asset to a business-level application.

Attaching the policy sets from the WebSphere administrative console

You can also attach the WebSphere Application Server policy sets to references using the administrative console or the wsadmin commands after the application is deployed as follows:

1. In the administrative console, select **Applications** → **Application Types** → **Business-level Applications**. The list of business-level applications, shown in Figure 7-10, shows both the AddressBook business-level application that includes the AddressBook Web service application and the RemoteAddressBook business-level application that includes AddressBookComposite.



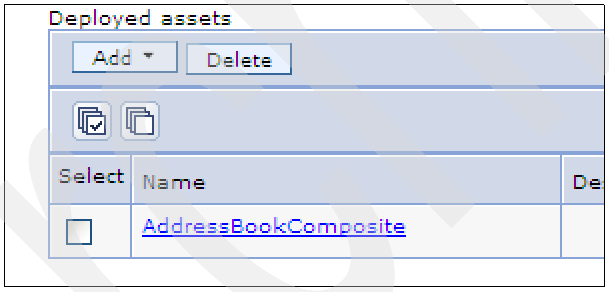
You can administer the following resource

☐	AddressBook
☐	DefaultApplication
☐	RemoteAddressBook
☐	ivtApp
☐	query

Total 5

Figure 7-10 List of business-level applications

2. Click RemoteAddressBook business-level application to open its configuration page. The "Deployed assets" section, shown in Figure 7-11, shows the SCA composite listed as an asset.



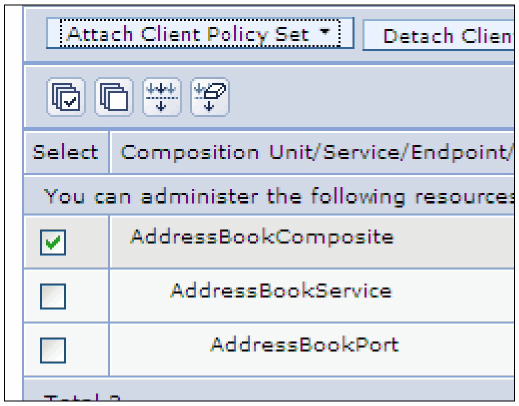
Deployed assets

Select	Name	De
☐	AddressBookComposite	

Figure 7-11 Select the composite

3. Click the link for the composite to open its configuration page. Select the **References policy sets and bindings** link from the Additional Properties tab. Remember, in this example, we attach the policy set only to the reference, not the whole service.

4. Select the AddressBookComposite as shown in Figure 7-12.


Attach Client Policy Set ▾ | Detach Clien

Select	Composition Unit/Service/Endpoint/
You can administer the following resources	
✓	AddressBookComposite
☐	AddressBookService
☐	AddressBookPort

Figure 7-12   Select the composite

5. Then attach the policy set as shown in Figure 7-13. Note that when attaching the WSReliableMessaging policy set to the reference, you also need to ensure that the Web service that is being referenced has the WSReliableMessaging policy set applied.

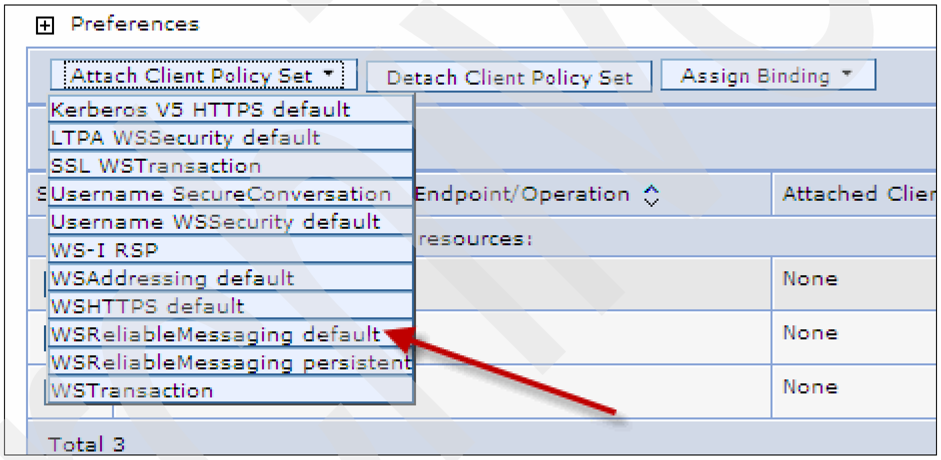
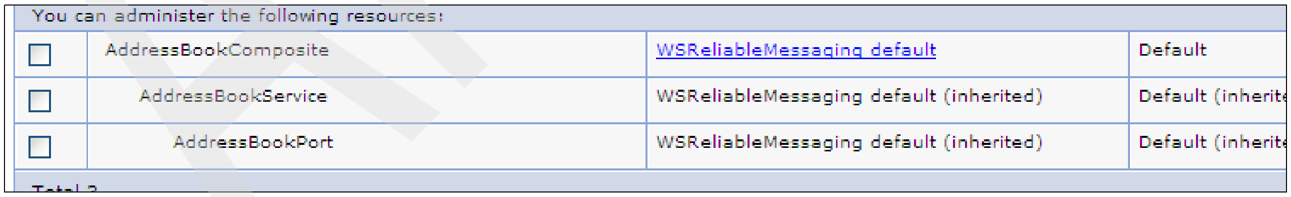


Figure 7-13   Attach the policy set

Figure 7-14 shows the results.



You can administer the following resources:			
☐	AddressBookComposite	WSReliableMessaging default	Default
☐	AddressBookService	WSReliableMessaging default (inherited)	Default (inherite
☐	AddressBookPort	WSReliableMessaging default (inherited)	Default (inherite

Figure 7-14   Verify the policy set is attached

6. Restart your application to ensure that the policy set is applied to the run time.

Testing the component and policy sets

Providing that you have deployed your service and SCA component successfully, you can now test it. You can use Rational Application Developer 7.5 with the Web Services Explorer as follows:

1. Start the Web Service Explorer. Then, follow these steps:
   a. Select the Launch the Web Services Explorer icon in the toolbar, as shown in Figure 7-15.

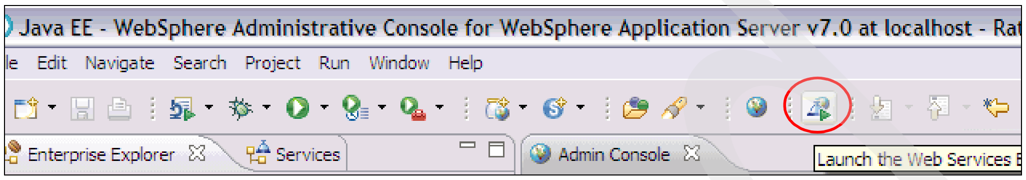
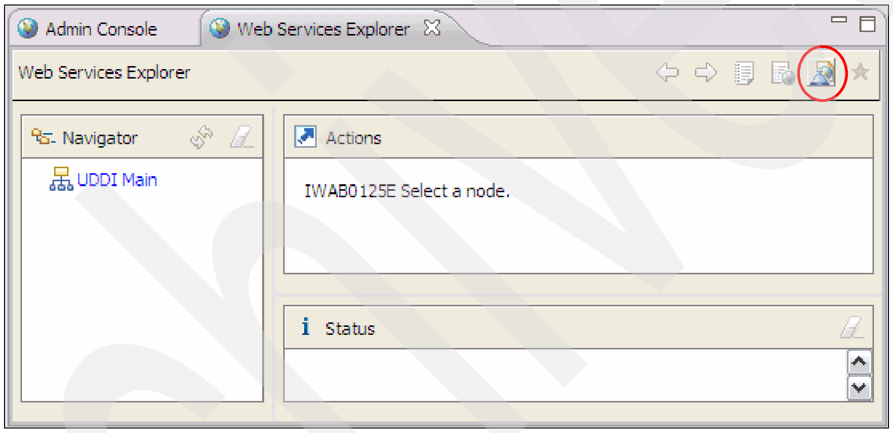

Java EE - WebSphere Administrative Console for WebSphere Application Server v7.0 at localhost - Rat
le Edit Navigate Search Project Run Window Help
Enterprise Explorer
Services
Admin Console
Launch the Web Services E

Figure 7-15 Launch the Web Services Explorer

   b. Select the WSDL page icon in top right-hand corner, as shown in Figure 7-16.

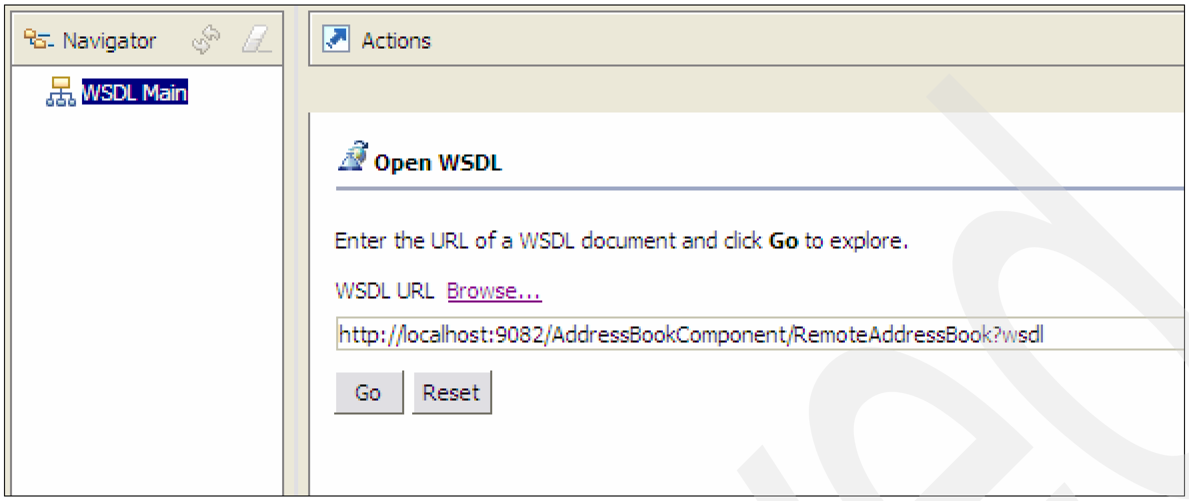


IWAB0125E Select a node.

Figure 7-16 WSDL page icon

   c. Select the WSDL Main in navigator pane.

2. Enter the WSDL URL as shown in Figure 7-17. Remember, in this example, we used the binding.ws for the service and selected the “Use generated WSDL” option. (The `host:port` value will differ depending on the application server). Click **Go**.



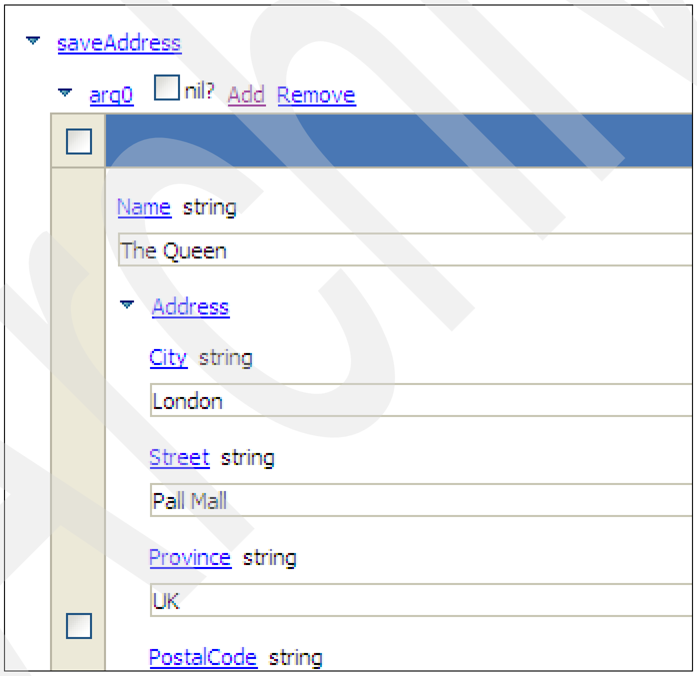
Enter the URL of a WSDL document and click **Go** to explore.

WSDL URL Browse...

http://localhost:9082/AddressBookComponent/RemoteAddressBook?wsdl

Figure 7-17 Web Services Explorer

3. Expand the Navigator until you can chose a method to invoke. Select **saveAddress**.
4. Enter test address information to be saved as shown in Figure 7-18.



Name string

The Queen

Address

City string

London

Street string

Pall Mall

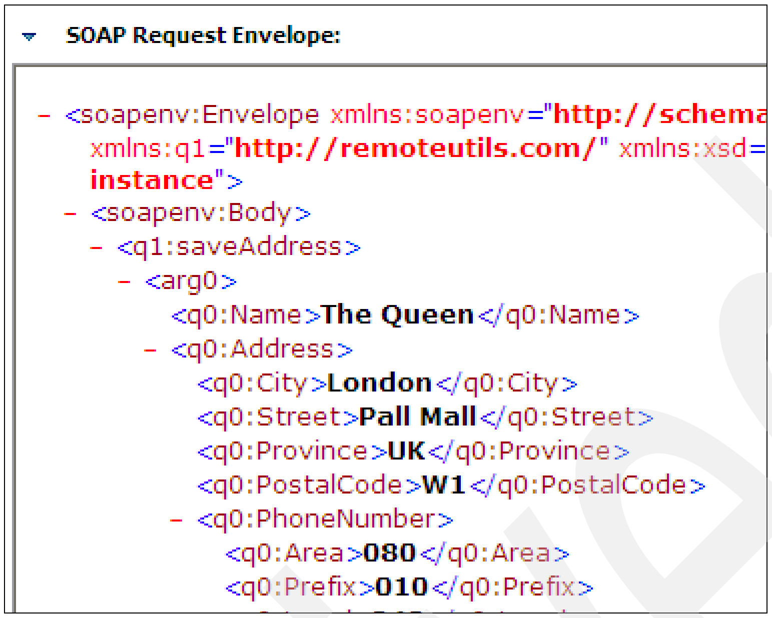
Province string

UK

PostalCode string

Figure 7-18 Test data

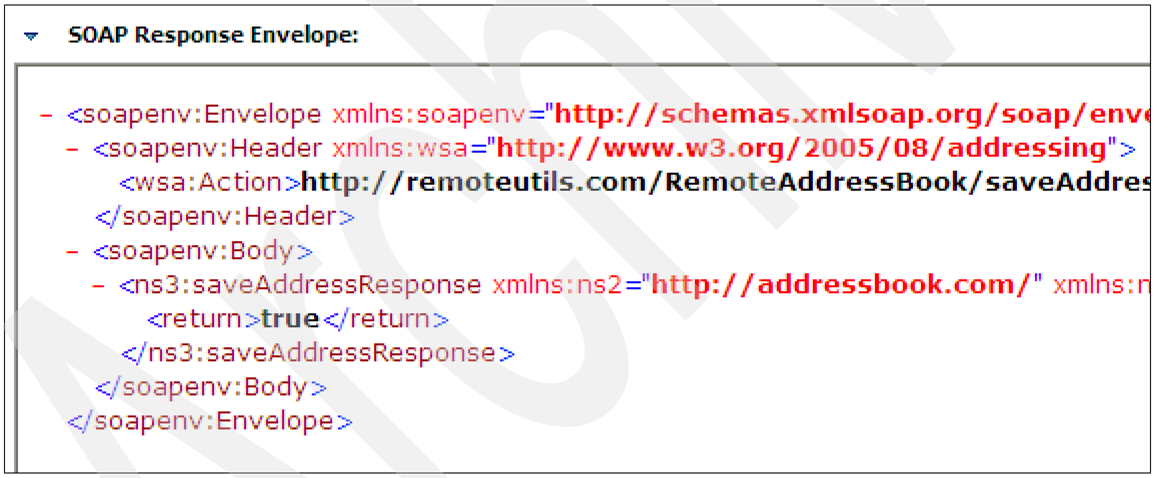
5. Then, click **Go**. The request message displays, as shown in Figure 7-19.



```
SOAP Request Envelope:

- <soapenv:Envelope xmlns:soapenv="http://schema
    xmlns:q1="http://remoteutils.com/" xmlns:xsd=
    instance">
  - <soapenv:Body>
    - <q1:saveAddress>
      - <arg0>
          <q0:Name>The Queen</q0:Name>
        - <q0:Address>
            <q0:City>London</q0:City>
            <q0:Street>Pall Mall</q0:Street>
            <q0:Province>UK</q0:Province>
            <q0:PostalCode>W1</q0:PostalCode>
          - <q0:PhoneNumber>
              <q0:Area>080</q0:Area>
              <q0:Prefix>010</q0:Prefix>
```

Figure 7-19 SOAP request message

Figure 7-20 shows the response message.



```
SOAP Response Envelope:

- <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/enve
  - <soapenv:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
      <wsa:Action>http://remoteutils.com/RemoteAddressBook/saveAddres
    </soapenv:Header>
  - <soapenv:Body>
    - <ns3:saveAddressResponse xmlns:ns2="http://addressbook.com/" xmlns:n
        <return>true</return>
      </ns3:saveAddressResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Figure 7-20 SOAP response message

Using the Trace Analyzer to inspect the messages

You can configure the Trace Analyzer for WebSphere Application Server to collect trace data and to ensure that the Reliable Messaging protocol messages have flowed as you expected.

You can find more information about Trace Analyzer on Alphaworks at:

`http://www.alphaworks.ibm.com/tech/ta4was`

Follow these steps:

1. First turn on trace for the server using the administrative console. Use `com.ibm.ws.websvcs.*=All` as the trace string.
2. Then in Trace Analyzer, select **Refine** → **Search** → **Open Search Dialogue.** Enter `MessageTrace` as a String.
3. You can follow the flow of SOAP messages into the SCA component, then forwarded onto the Web Service, as well as the responses back.

   Look at the messages to verify that WS-RM protocol messages are sent and received successfully. For example, see the CreateSequence message in Example 7-9.

Example 7-9 CreateSequence message

```
Content-Type: text/xml; charset=UTF-8
Message contents:
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header>
      <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">
         http://localhost:9082/jwsAddressBook/AddressBookService</wsa:To>
      <wsa:ReplyTo xmlns:wsa="http://www.w3.org/2005/08/addressing">
         <wsa:Address>

http://docs.oasis-open.org/ws-rx/wsmc/200702/anonymous?id=urn:uuid:FB6923CCD81191A
AB51265390476674
         </wsa:Address>
      </wsa:ReplyTo>
      <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
         urn:uuid:FB6923CCD81191AAB51265390476675</wsa:MessageID>
      <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
         http://docs.oasis-open.org/ws-rx/wsrm/200702/CreateSequence
      </wsa:Action>
   </soapenv:Header>
   <soapenv:Body>
      <wsrm:CreateSequence
xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702">
         <wsrm:AcksTo>
            <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">

http://docs.oasis-open.org/ws-rx/wsmc/200702/anonymous?id=urn:uuid:FB6923CCD81191A
AB51265390476674
            </wsa:Address>
         </wsrm:AcksTo>
         <wsrm:Offer>
            <wsrm:Identifier>urn:uuid:FB6923CCD81191AAB51265390476676
            </wsrm:Identifier>
            <wsrm:Endpoint>
               <wsa:Address xmlns:wsa="http://www.w3.org/2005/08/addressing">

http://docs.oasis-open.org/ws-rx/wsmc/200702/anonymous?id=urn:uuid:FB6923CCD81191A
AB51265390476674
               </wsa:Address>
            </wsrm:Endpoint>
         </wsrm:Offer>
      </wsrm:CreateSequence>
   </soapenv:Body>
</soapenv:Envelope>
```

Example 7-10 shows the response message.

Example 7-10 Response message

```
WSWS3572I: Outbound HTTP SOAP response:
Content-Type: text/xml
Message
contents:
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Header>
      <wsa:To xmlns:wsa="http://www.w3.org/2005/08/addressing">

http://docs.oasis-open.org/ws-rx/wsmc/200702/anonymous?id=urn:uuid:567881CBBD36187
D411265390745311
      </wsa:To>
      <wsa:MessageID xmlns:wsa="http://www.w3.org/2005/08/addressing">
         urn:uuid:567881CBBD36187D411265390745984</wsa:MessageID>
      <wsa:Action xmlns:wsa="http://www.w3.org/2005/08/addressing">
         http://addressbook.com/AddressBook/SaveAddressResponse</wsa:Action>
      <wsa:RelatesTo xmlns:wsa="http://www.w3.org/2005/08/addressing">
         urn:uuid:567881CBBD36187D411265390745216</wsa:RelatesTo>
      <wsrm:Sequence xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
         soapenv:mustUnderstand="1">
         <wsrm:Identifier>urn:uuid:567881CBBD36187D411265390745329
         </wsrm:Identifier>
         <wsrm:MessageNumber>1</wsrm:MessageNumber>
      </wsrm:Sequence>
      <wsrm:SequenceAcknowledgement
         xmlns:wsrm="http://docs.oasis-open.org/ws-rx/wsrm/200702"
         soapenv:mustUnderstand="1">
         <wsrm:Identifier>urn:uuid:567881CBBD36187D411265390745616
         </wsrm:Identifier>
         <wsrm:AcknowledgementRange Lower="1"
Upper="1"></wsrm:AcknowledgementRange>
      </wsrm:SequenceAcknowledgement>
   </soapenv:Header>
   <soapenv:Body>
      <SaveAddressResponse xmlns="http://addressbook.com/">true
      </SaveAddressResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

Investigate the other messages so that you understand the flows.

7.3.2 Example 2: Using transaction intents

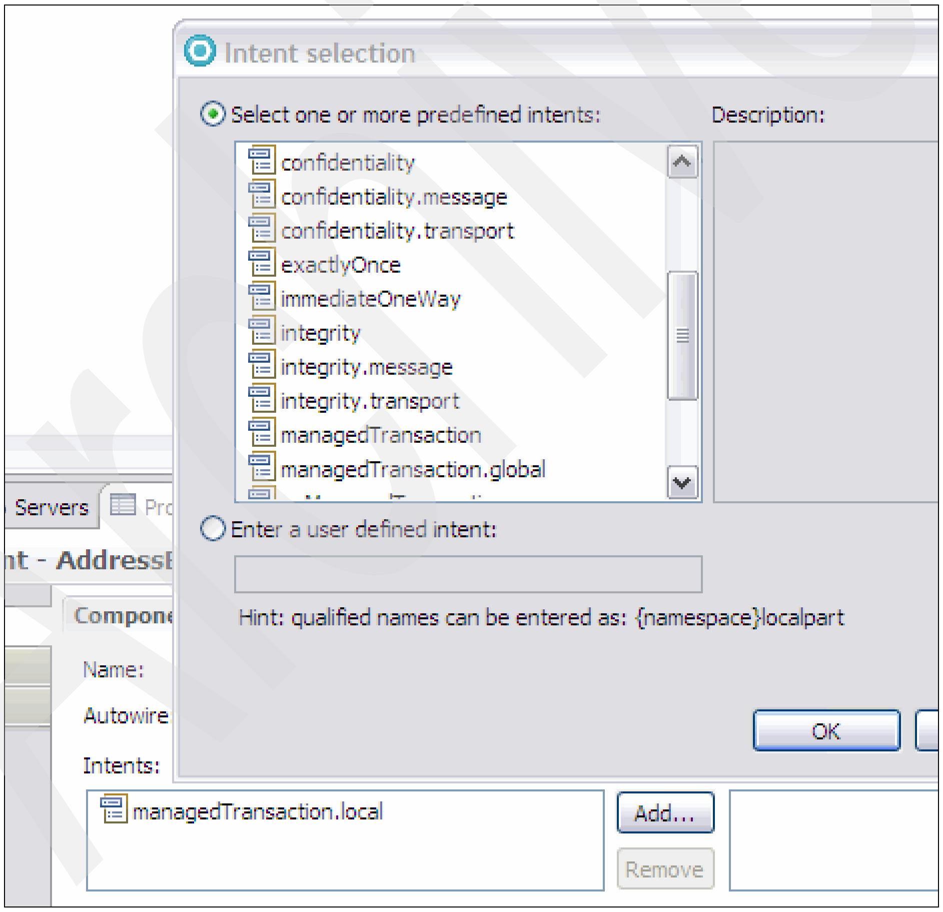
SCA intents are meant as a hint or suggestion to the deployer, representing the intentions of the developer of the code. In WebSphere Application Server, you need to configure the deployed application appropriately to meet these intents. However, for transactional intents on SCA components, the SCA run time will ensure that the correct transactional semantics are applied to the SCA component.

This example shows how to attach intents (and specific considerations) for Web services. Because the composite has only a single component, this example is not a good demonstration of transactional behavior, but that is not the purpose of the example.

For further information, the following article in the information center covers this topic in greater detail:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/csca_global_trans.html

If you are using the SCA tooling for Rational Application Developer 7.5, you can add intents to any elements in the SCA composite file from the Properties view (Core tab) by selecting **Add** next to the list of Intents, and then choosing the intent you want to attach (Figure 7-21).


Intent selection

● Select one or more predefined intents:

Description:

- confidentiality
- confidentiality.message
- confidentiality.transport
- exactlyOnce
- immediateOneWay
- integrity
- integrity.message
- integrity.transport
- managedTransaction
- managedTransaction.global

○ Enter a user defined intent:

Hint: qualified names can be entered as: {namespace}localpart

Intents:

- managedTransaction.local

Figure 7-21 Adding intents

In this example, we selected the managedTransaction.local intent, indicating that we want the SCA run time to provide a local transaction when this component is running. For now, we left the WS-RM protocol running over the Web service binding. Example 7-11 shows the resulting composite XML file.

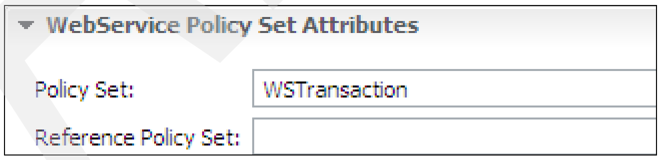
Example 7-11 Setting the intent on a component

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
xmlns:was="http://www.ibm.com/xmlns/prod/websphere/sca/1.0/2007/06"
autowire="false" name="AddressBookComposite" targetNamespace="http://addressbook">
  <component name="AddressBookComponent" requires="managedTransaction.local">
      <implementation.java class="com.remoteutils.RemoteAddressBookImpl"/>
      <service name="RemoteAddressBook">
          <interface.java interface="com.remoteutils.RemoteAddressBook"/>
          <binding.ws/>
      </service>
      <reference name="addressBookReference" was:wsPolicySet="WSReliableMessaging
default">
          <interface.wsdl
interface="http://addressbook.com/#wsdl.interface(AddressBook)"/>
          <binding.ws
wsdlElement="http://addressbook.com/#wsdl.service(AddressBookService)"/>
      </reference>
  </component>
</composite>
```

However, to attach transactional behavior to the reference that uses a Web services binding, we need to attach the WSTransaction policy set so that the WS-AT protocol is enabled for interaction with the Web service. A couple of additional points to be aware of here:

- The Web service policy set must be updated so that the WSTransaction policy set is attached.
- WS-ReliableMessaging cannot run over the same message flow as WS-AtomicTransaction, so we must remove the WS-RM policy set from the reference and the Web service. The two protocols conflict and are mutually exclusive.

In Rational Application Developer, we can attach the policy set to the reference. Figure 7-22 shows the settings that are required to attach the WSTransaction policy set to a Web service binding. You can attach the policy using the Properties view for the reference (core tab) or the binding. If the policy set is specified in both places, the binding level specifications override the reference level.

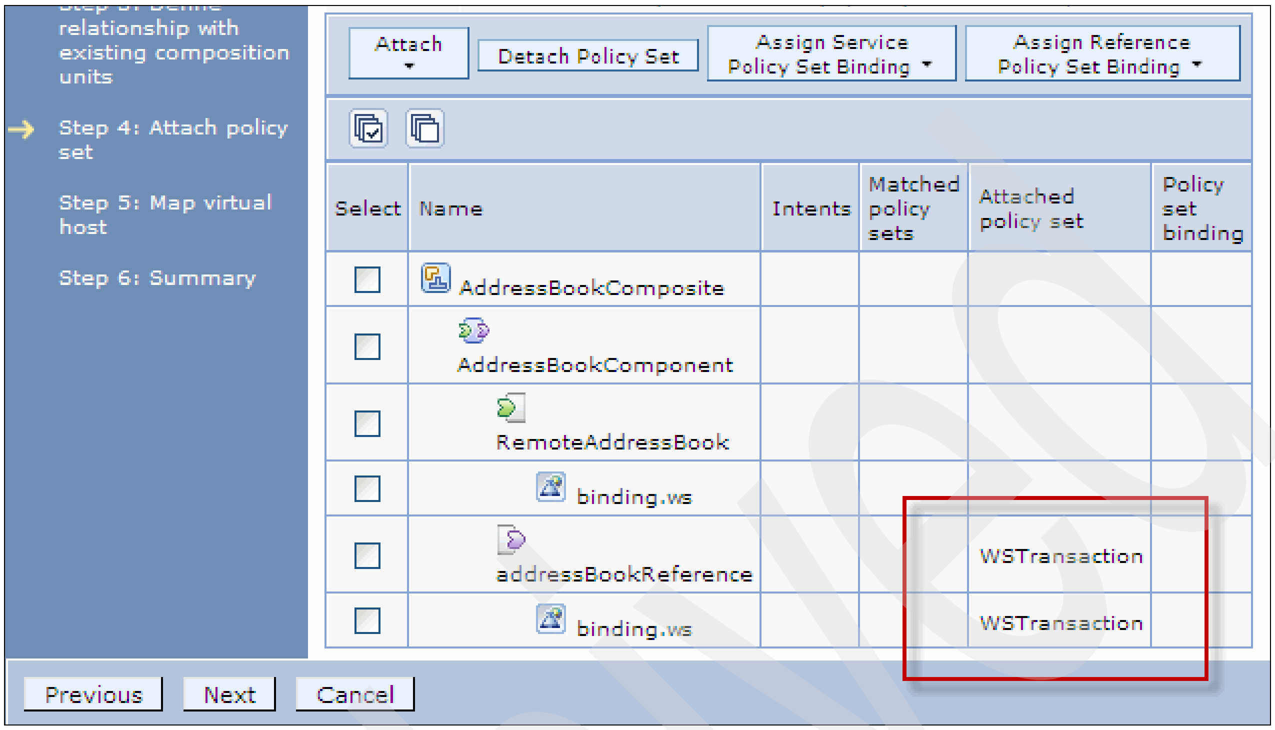

WebService Policy Set Attributes

Policy Set: WSTransaction

Reference Policy Set:

Figure 7-22 Web service policy set attributes for a binding

Alternatively, at deployment time in the administrative console, you have this option when you add an asset that has a policy set to a business-level application, as shown in Figure 7-23.


Step 3: Define relationship with existing composition units
Step 4: Attach policy set
Step 5: Map virtual host
Step 6: Summary
Attach
Detach Policy Set
Assign Service Policy Set Binding
Assign Reference Policy Set Binding
Previous
Next
Cancel

Select	Name	Intents	Matched policy sets	Attached policy set	Policy set binding
☐	AddressBookComposite				
☐	AddressBookComponent				
☐	RemoteAddressBook				
☐	binding.ws				
☐	addressBookReference			WSTransaction	
☐	binding.ws			WSTransaction	

Figure 7-23 Attaching policy sets at deploy time

Some thoughts on transactional intents

Do not forget that intents can apply at the implementation level (for example, managedTransaction.global or managedTransaction.local), and they can apply to the interaction, that is the reference or service (for example, propagatesTransaction or suspendsTransaction).

In a scenario where you have component A invoking component B, you need to configure both components with managedTransaction.global to coordinate A and B's resource access together. If you also apply propagatesTransaction, both components are part of the same global transaction. However, if you apply suspendsTransaction, then when B is invoked the container suspends the current transaction and starts a new global transaction under which B will run. Additionally, B might or might not in turn propagate this transaction downstream.

7.4 WS-Policy

WebSphere Application Server introduced support for WS-Policy in V7. WS-Policy provides a standardized method to share configuration dynamically and to agree upon a common configuration for a particular interaction. WebSphere Application Server uses policy sets to define that configuration on a Web service and then publishes (shares) that configuration through WS-Policy.

The Feature Pack for SCA v1.0.1 does not support the use of WS-Policy.

For more information about the support in WebSphere Application Server for WS-Policy, see:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.zseries.doc/info/zseries/ae/twbs_wsp_learning.html

8

Using JMS bindings

You can use the JMS binding on service and reference interfaces to enable asynchronous messaging between the SCA component and JMS providers. With asynchronous messaging, applications send messages to the messaging provider and continue processing without waiting for a response. The messaging provider delivers the message to the target application, using the application quality of service levels that specify how the message delivery is managed. The Feature Pack for SCA supports the WebSphere Application Server default messaging provider and WebSphere MQ as the messaging providers.

This chapter takes a look at the JMS binding use and properties.

8.1 When to use the JMS binding

Although it is possible to use a JMS binding to connect two SCA components, the most common use of the JMS binding is for integration with existing non-SCA JMS consumers and producers as shown in Figure 8-1.

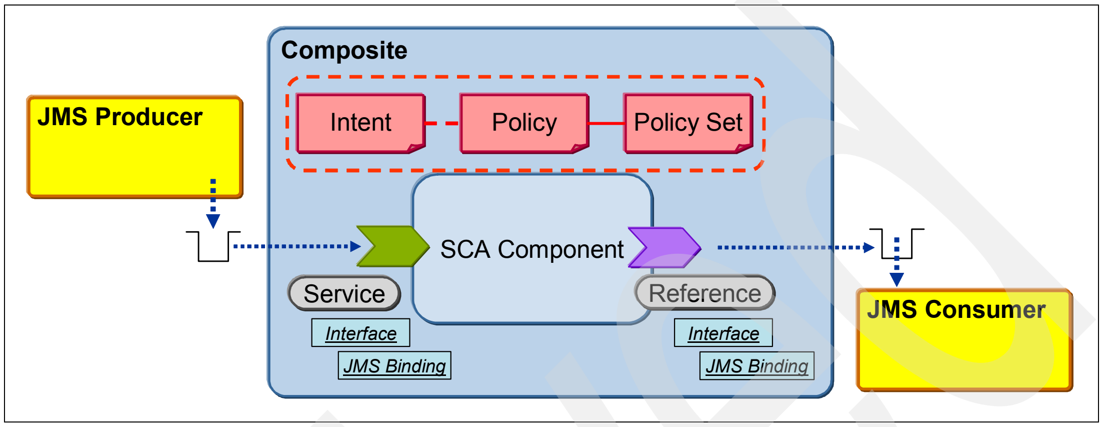


Figure 8-1 SCA components with JMS bindings

8.2 JMS message format

JMS messages contain a header, the properties, and body as shown in Figure 8-2. Only the header is required. The properties and the body are optional.

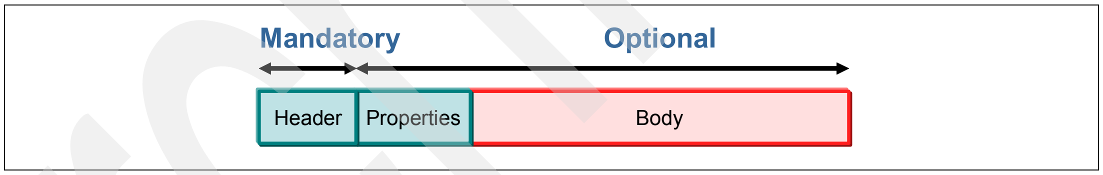


Figure 8-2 JMS message format

A JMS message header contains values that both clients and providers use to identify and route messages. The header has several fields, including the following fields:

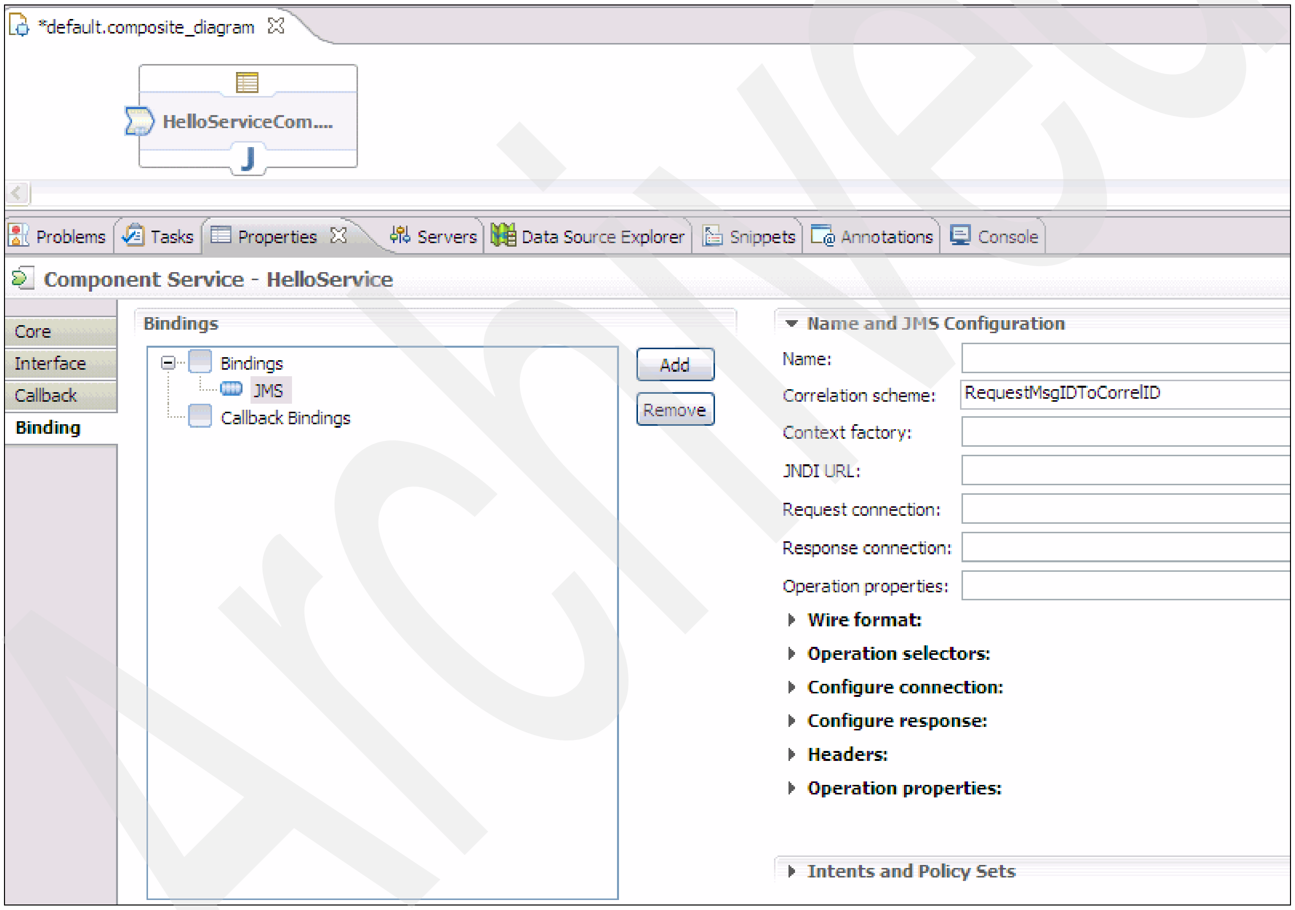
- `JMSMessageID`: The identifier of the message that is sent to the queue or topic.
- `JMSCorrelationID`: Used to provide an application with some means of matching a response with the request message. Usually the correlation ID of the reply is equal to the message ID of the request message.
- `JMSDestination`: Represents the queue or the topic to which the message is sent.
- `JMSReplyTo`: Contains the name of the queue where the response should be put. It is set by the sending application.

8.3 JMS binding properties

The JMS binding can contain a number of properties, including those that describe the message data format, that relate the interface to request and response queues and to activation specifications and that apply policies sets and intents to the interface. For details about how to define the binding and set the properties, see:

http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/tsca_scajmsbinding.html

The SCA tools in Rational Application Developer provide a graphical interface to specify these properties. The fields in the Binding tab of the Properties view (Figure 8-3) correspond to values in the composite XML file. You can access these fields by opening the composite in the work area and then clicking the interface or reference that use the binding. In the Properties view Binding tab, add the JMS binding and define the properties.



Name:	
Correlation scheme:	RequestMsgIDToCorrelID
Context factory:	
JNDI URL:	
Request connection:	
Response connection:	
Operation properties:	

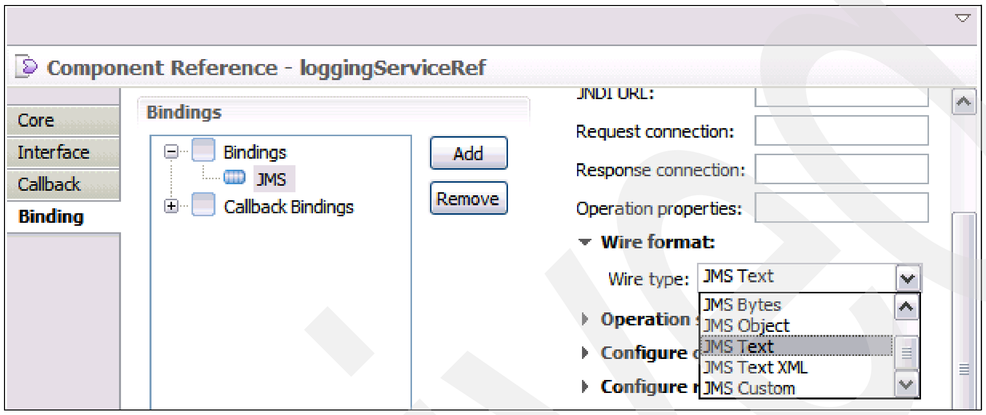
Figure 8-3 JMS binding properties - Rational Application Developer

Request and response:

- A *request* is a message that is sent to an SCA service or sent by an SCA reference.
- A *response* is a message received from a reference (that is, a reply from an invoked service) or a message sent by a service in response to a previous request message. A response in SCA is always a reply to a previous request.

8.3.1 Wire format

When you use a JMS binding, you need to specify the *wire format*. The wire format defines the message type and data format that the JMS binding supports. If you are using an existing messaging infrastructure, the wire format is predetermined, and you need to select the corresponding wire format on the binding. If the SCA component is part of a new messaging infrastructure, you need to determine the wire format that best suits your needs. The SCA service, client, and interface must all be implemented to map data using this format. You can select the wire format on the Binding tab as shown in Figure 8-4.


Component Reference - loggingServiceRef

Core
Interface
Callback
Binding

Bindings
Bindings
JMS
Callback Bindings
Add
Remove

JNDI URL:
Request connection:
Response connection:
Operation properties:
Wire format:
Wire type: JMS Text
JMS Bytes
JMS Object
JMS Text
JMS Text XML
JMS Custom
Operation
Configure
Configure

Figure 8-4 Wire format selection

Supported message types include:

- JMS BytesMessage format (JMS Bytes option)

  The message body is sent as a stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.

- JMS ObjectMessage (JMS Object option) (wrap single true or false)

  The message body is sent as a serializable Java object (`java.lang.Object`)

- JMS TextMessage

  The message body is sent as a `java.lang.String` type.

  – JMS Text

    The message body maps to single string messages.

  – JMS Text XML

    The text message contains application data in serialized XML format that maps data using JAXB. This is the default wire format.

- JMS Custom

  Provide a class name

- JMS Default (select format type of text or bytes)

For more information about wire formats and determining the best to use, see:

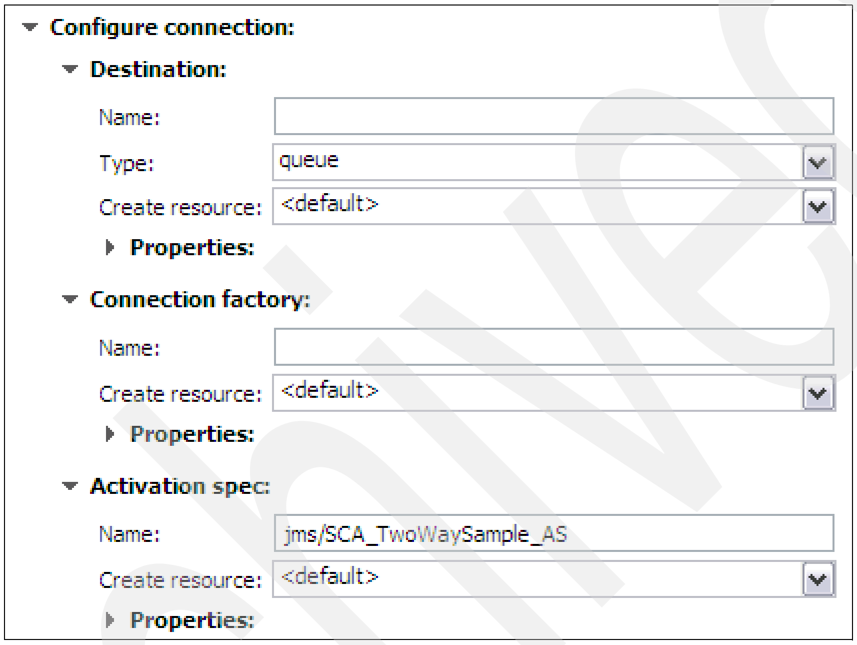
http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp?topic=/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/rsca_jmsbinding_msg_type.html

8.3.2 Connection properties

For the purposes of this discussion, we focus on the following settings that define the connectivity to the messaging infrastructure. These properties are found on the Bindings tab of the Properties view for the interface or reference.

- Configure connection (Figure 8-5)

  This section is used to configure the resources used for handling request messages that are received from a service or are sent to a reference. Specify the destination type and JNDI name. For reference bindings, specify the connection factory. For service bindings, specify the activation specification.



Configure connection:

Destination:

Name:	
Type:	queue
Create resource:	<default>

Properties:

Connection factory:

Name:	
Create resource:	<default>

Properties:

Activation spec:

Name:	jms/SCA_TwoWaySample_AS
Create resource:	<default>

Properties:

Figure 8-5 Configure connection properties

You define the following values in this section:

– Destination name: (Optional) Specifies the JNDI name of a destination to which the binding is connected. The JNDI name is defined by a JMS resource definition.

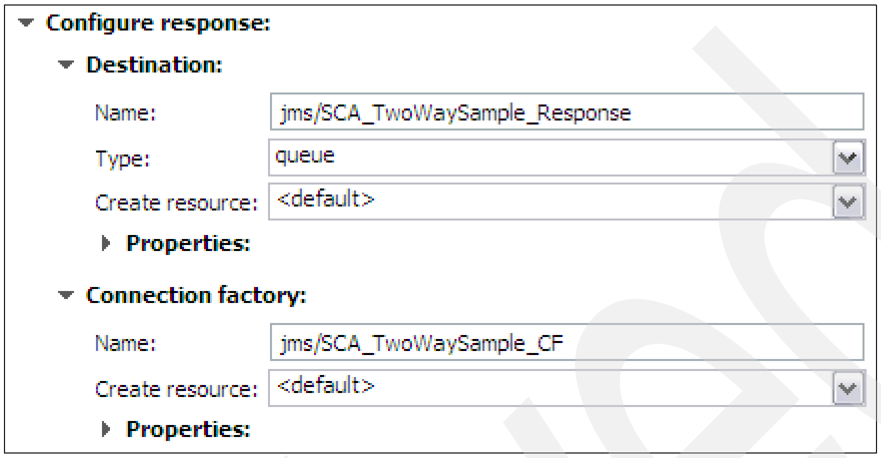
– Destination type: Specifies the type (queue or topic) of the request destination. When topic is specified, then all the operations in the interface that correspond to the binding must be one-way.

– Connection factory name: (References only) JNDI name of the connection factory for the JMS provider. (Mutually exclusive with the activationSpec property.)

– Activation specification name: (Services only) Used by the binding to connect to a JMS destination to process request messages.

- Configure response (Figure 8-6)

  This section is used to configure the resources used for handling response messages, either when receiving responses from a reference, or for sending responses from a service. Specify the destination and the connection factory.



Configure response:

Destination:

Name:	jms/SCA_TwoWaySample_Response
Type:	queue
Create resource:	<default>

Properties:

Connection factory:

Name:	jms/SCA_TwoWaySample_CF
Create resource:	<default>

Properties:

Figure 8-6 Configure response properties

  You define the following values in this section:

  – Destination name: Specifies the JNDI name of the destination that is to be used to process responses by this binding.
  – Destination type: Specifies the type (queue or topic) of the response destination. When topic is specified, then all the operations in the interface that correspond to the binding must be one-way.
  – Response connection factory name: Specifies the JNDI name of the connection factory that the binding uses to process response messages.

Optionally, you can create resources for the binding dynamically. The destination, activation specification, and connection factory settings all provide an option to have the resource that is created dynamically when the SCA composite is added to the business-level application. The default is for the resources to be created dynamically if they do not exist.

8.4 Message exchange patterns

This section looks at the message exchange patterns that are used with the JMS binding in the Feature Pack for SCA. This section focuses on the typical uses and the connection properties that you configure.

8.4.1 One-way messaging

In this pattern, a request message is sent and a response is not expected. When defining the binding, configure the following properties:

- Service interface bindings:
  – Destination type
  – Destination name
  – Activation specification

- Reference interface:
  - Destination type
  - Destination name
  - Connection factory

8.4.2 Request-response messaging

In this pattern, a request message is sent and a response is returned.

Service interface

Figure 8-7 shows a request-response message flow on a service interface.

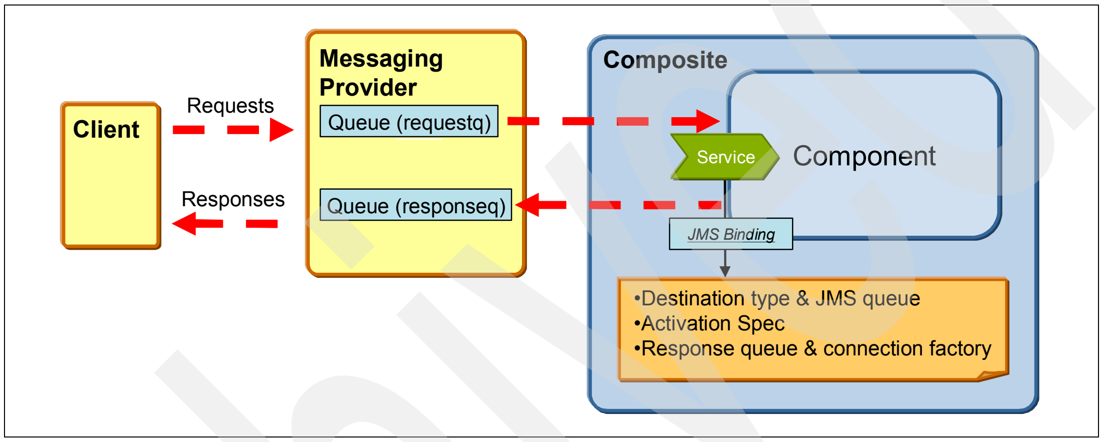


Figure 8-7 Request-response message flow on a service interface

Configure the following binding properties:

- Destination type
- Destination name
- Activation specification

For the response, the binding must specify:

- Response destination
- Connection factory

If the request includes the JMSReplyTo attribute, that value overrides the response destination.

Reference interface

Figure 8-8 shows a request-response message flow on a reference interface.

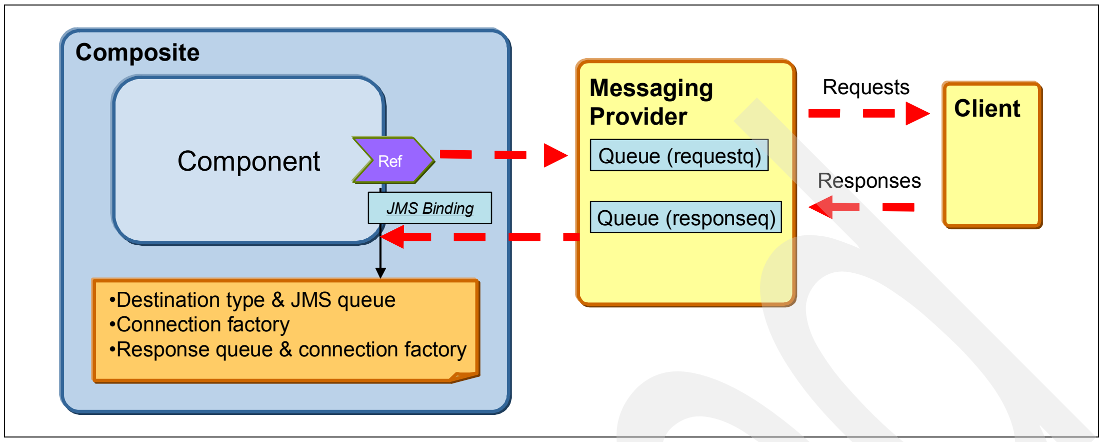


Figure 8-8 Request-response message flow on a reference interface

Configure the following binding properties:

- Destination type
- Destination name
- Connection factory

For the response, the binding must specify:

- Response destination
- Connection factory

If the request includes the JMSReplyTo attribute, that value overrides the response destination.

8.4.3 One-way messaging with callback

By applying both a JMS binding and a JMS callback binding on the same interface, you enable the interface for bi-directional communication. A typical use would be in an application that sends a request and then waits for a response. An activation specification notifies the application that a request message has arrived on the queue. A timeout can be used to prevent the application from waiting for a callback message that never arrives.

A JMS callback binding has the same properties as a JMS binding, including the ability to do request-response messaging.

Reference interface

Figure 8-9 shows the message flow on a service interface when using the one-way messaging with callback pattern.

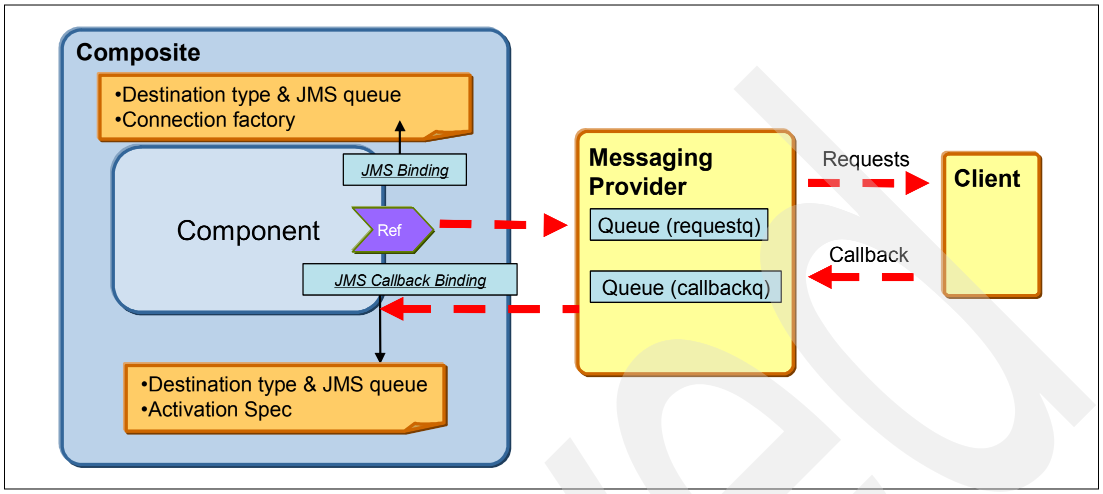


Figure 8-9 JMS callback binding on a reference

The JMS binding is configured for one-way messaging. Specify the following properties:

- Destination type
- Destination name
- Connection factory

The JMS callback binding is configured for incoming request messages. Specify the following properties:

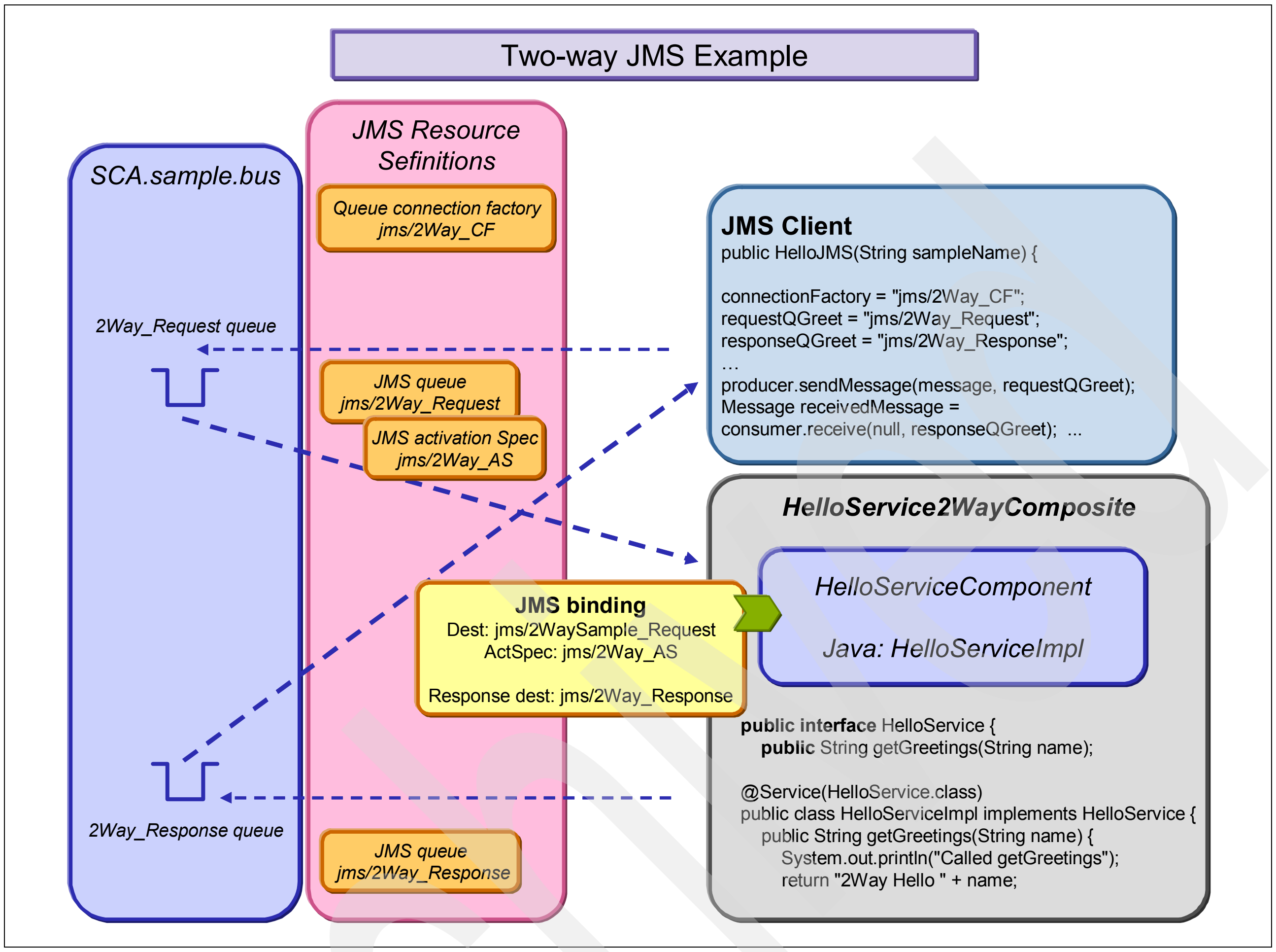
- Destination type
- Destination name
- Activation specification

8.5 Example: Receiving messages from a JMS producer

Example source: The TwoWayService example ships with the Feature Pack for SCA. This example illustrates a request-response (two-way) message pattern.

The TwoWayService sample illustrates the use of a JMS binding to expose a service to a non-SCA application. In this example, a JMS client sends a message to the HelloService implemented as an SCA composite in WebSphere Application Server. The SCA service returns a response.

Figure 8-10 shows the mechanics of this example.


Two-way JMS Example
SCA.sample.bus
2Way_Request queue
2Way_Response queue
JMS Resource Sefinitions
Queue connection factory jms/2Way_CF
JMS queue jms/2Way_Request
JMS activation Spec jms/2Way_AS
JMS queue jms/2Way_Response
JMS binding
Dest: jms/2WaySample_Request
ActSpec: jms/2Way_AS
Response dest: jms/2Way_Response
JMS Client
HelloService2WayComposite
HelloServiceComponent
Java: HelloServiceImpl

```
public HelloJMS(String sampleName) {

connectionFactory = "jms/2Way_CF";
requestQGreet = "jms/2Way_Request";
responseQGreet = "jms/2Way_Response";
…
producer.sendMessage(message, requestQGreet);
Message receivedMessage =
consumer.receive(null, responseQGreet);  ...
```

```
public interface HelloService {
   public String getGreetings(String name);

@Service(HelloService.class)
public class HelloServiceImpl implements HelloService {
   public String getGreetings(String name) {
      System.out.println("Called getGreetings");
      return "2Way Hello " + name;
```

Figure 8-10 2WaySample

8.5.1 Messaging infrastructure

The messaging infrastructure for this example is provided by the service integration bus and default messaging provider in WebSphere Application Server, with the following definitions:

- Service integration bus
  - Name: SCA.sample.bus
- JMS queue connection factory for the bus
  - Name: 2Way_CF
  - JNDI: jms/2Way_CF
- Service integration bus queues

  Two queue destinations are defined on the bus. The request queue is for messages from the client to the server:

  - Name: 2Way_Request

  The response queue holds messages from the server to the client.

  - Name: 2Way_Response

- JMS queue definitions

  Two JMS queue resource definitions are defined to provide access to the queues on the bus. The first definition is for the request queue:

  – Name: 2Way_Request
  – JNDI: jms/2Way_Request
  – Queue: 2Way_Request

  The second definition is for the response queue.

  – Name: 2Way_Response
  – JNDI: jms/2Way_Response
  – Queue: 2Way_Response

- JMS activation specification

  The JMS activation specification is associated with the JMS binding for the service interface of the SCA component and provides the configuration necessary for it to receive unsolicited messages.

  – Name: 2Way_AS
  – JNDI: jms/2Way_AS
  – Destination queue: jms/2Way_Request

8.5.2 The JMS client

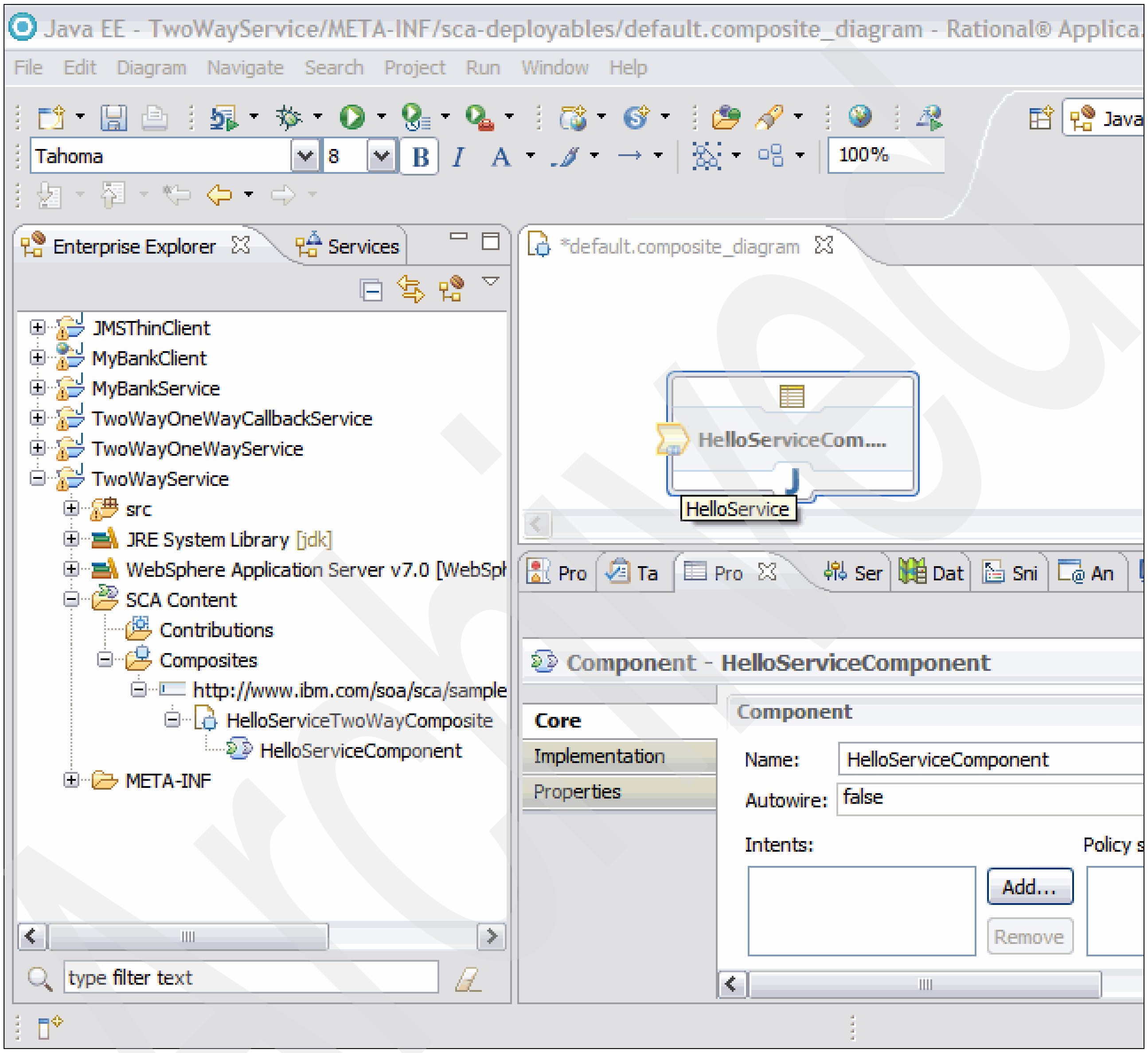
The Thin Client for JMS is used to provide a simple client in this example, as shown in Example 8-1. The client uses the connection factory to access the default messaging provider and service integration bus. It uses the JNDI name for the JMS request queue to send messages to HelloService and the JNDI name for the JMS response queue to receive the response.

Example 8-1 Excerpt from the JMS client

```
public class HelloJMS {
...
   public HelloJMS(String sampleName) {

      connectionFactory = "jms/2Way_CF";
      requestQGreet = "jms/2Way_Request";
      responseQGreet = "jms/2Way_Response";
      setUp();
   }

    public void testGreetings() {
      try {
..       producer.sendMessage(message, requestQGreet);
         Message receivedMessage = consumer.receive(null, responseQGreet);
...   }
   }
```

8.5.3 The HelloService composite

The HelloServiceTwoWayComposite composite, shown in Figure 8-11, consists of one component and a service interface.


Java EE - TwoWayService/META-INF/sca-deployables/default.composite_diagram - Rational® Applica
File Edit Diagram Navigate Search Project Run Window Help
Tahoma
8
100%
Enterprise Explorer
Services
*default.composite_diagram
HelloServiceCom....
HelloService
Pro
Ta
Pro
Ser
Dat
Sni
An
type filter text

JMSThinClient
MyBankClient
MyBankService
TwoWayOneWayCallbackService
TwoWayOneWayService
TwoWayService
src
JRE System Library [jdk]
WebSphere Application Server v7.0 [WebSph
SCA Content
Contributions
Composites
http://www.ibm.com/soa/sca/sample
HelloServiceTwoWayComposite
HelloServiceComponent
META-INF

Component - HelloServiceComponent

Core
Implementation
Properties

Component

Name: HelloServiceComponent
Autowire: false
Intents:
Policy s
Add...
Remove

Figure 8-11 *TwoWayServiceComposite*

HelloServiceComponent

The composite consists of one component that is implemented by the HelloService Java code, as shown in Example 8-2. The code receives the message, writes to the system out log, and then returns a message. The composite includes the following properties:

- Name: HelloServiceComponent
- Implementation type: Java
- Implementation: soa.sca.samples.jms.HelloServiceImpl

Example 8-2 Two-way HelloService implementation code

```
package soa.sca.samples.jms;
import org.osoa.sca.annotations.Reference;
import org.osoa.sca.annotations.Service;
import soa.sca.samples.jms.HelloService;

@Service(HelloService.class)
public class HelloServiceImpl implements HelloService {

    public String getGreetings(String name) {
        System.out.println("Called getGreetings");
        return "2Way Hello " + name;
    }
}
```

Service interface

The service interface has the following properties, as shown in Example 8-3:

- Name: HelloService
- Implementation type: Java
- Implementation: soa.sca.samples.jms.HelloService

Example 8-3 HelloService interface

```
package soa.sca.samples.jms;

import org.osoa.sca.annotations.Remotable;


/**
 * The interface for the helloworld service
 */
@Remotable
public interface HelloService {
    public String getGreetings(String name);
}
```

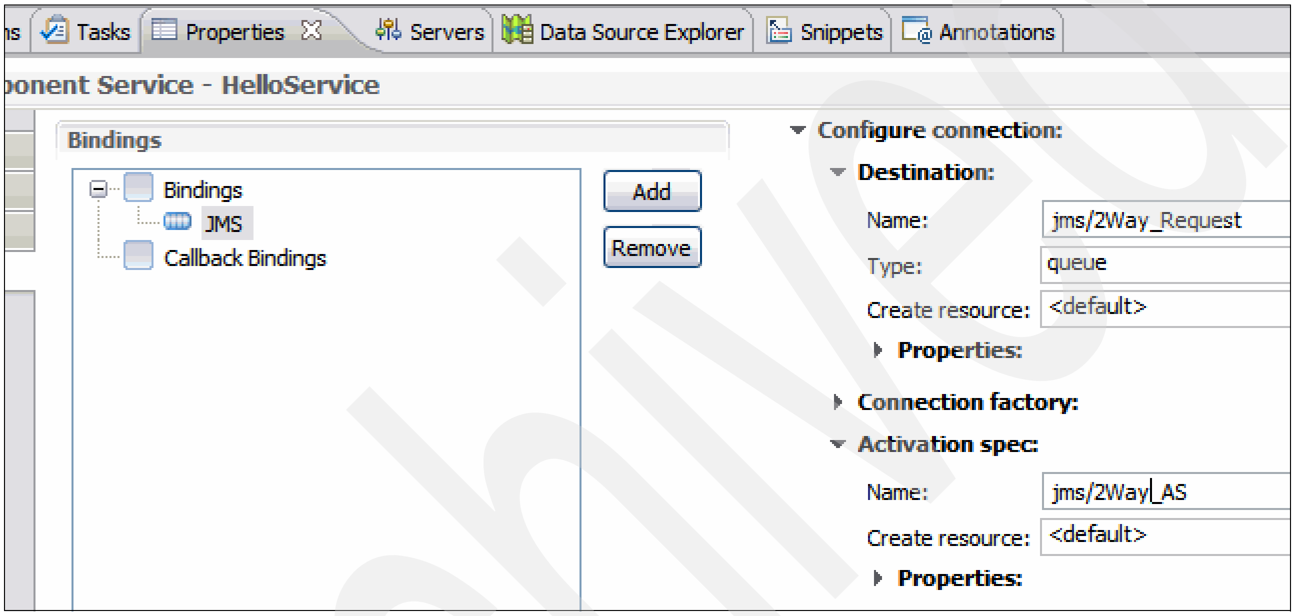
JMS binding

A JMS binding is applied to the service interface. The application uses a request-response pattern, which makes it necessary to specify a request queue, an activation specification, and a response queue.

To handle incoming messages, the binding specifies the following properties (Figure 8-12):

- `jms/2Way_Request`: The JNDI name of the JMS queue for message sent to the component
- `jms/2Way_AS`: The JNDI name of the activation specification

When a message that is defined by the activation specification arrives on the queue, the component is invoked to execute and process the message.



ponent Service - HelloService

Bindings
- Bindings
  - JMS
- Callback Bindings

Configure connection:

Destination:

Name:	jms/2Way_Request
Type:	queue
Create resource:	<default>

Properties:

Connection factory:

Activation spec:

Name:	jms/2Way_AS
Create resource:	<default>

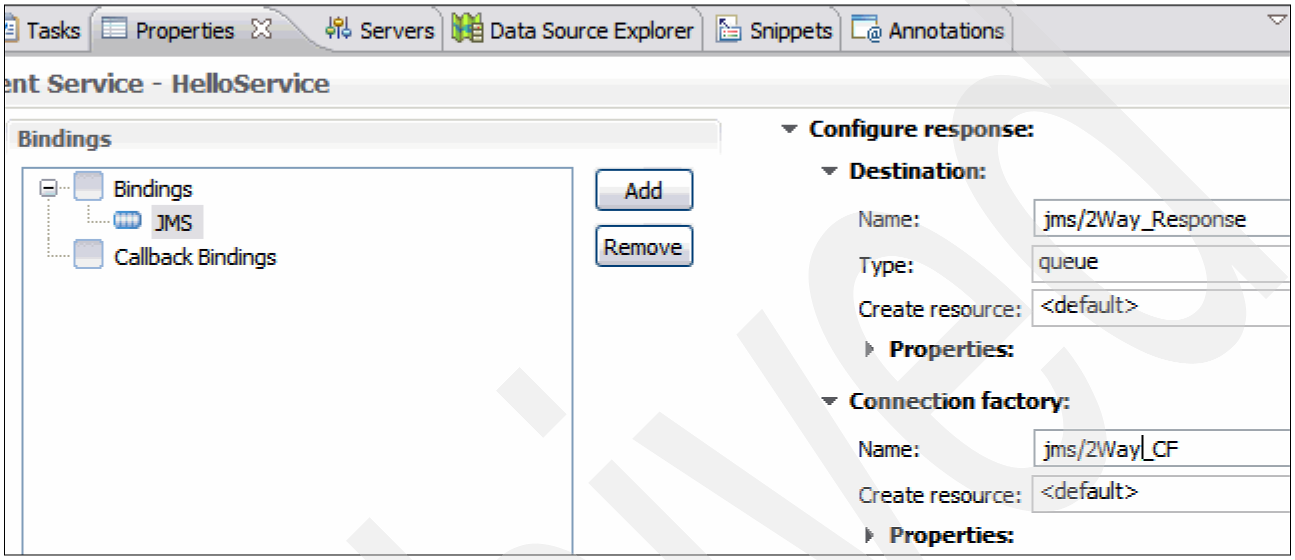
Properties:

Figure 8-12 JMS binding connection properties for incoming request messages

To handle the response messages that go back to the client, the bindings specify the following properties (Figure 8-13):

- `jms/2Way_Response`: The JNDI name of the response JMS queue
- `jms/2Way_CF`: The JNDI name of the connection factory

The component uses the connection factory to access the bus and the JNDI name of the JMS queue to put responses on this queue.



Name: jms/2Way_Response
Type: queue
Create resource: <default>

Name: jms/2Way_CF
Create resource: <default>

Figure 8-13 JMS binding response properties

9

Using a Spring implementation as an SCA component

The Spring Framework is an open source project that provides a Dependency Injection framework for Plain Old Java Objects (POJOs) and enables them to use the Java EE container through wrapper classes and XML configuration. For Feature Pack for SCA v1.0.1, you must use Spring Framework 2.5.5 or later.

The example in this chapter describes how to deploy an application that is developed with the Spring Framework as an SCA component without any changes to the original application and with minimal changes to the application context configuration file.

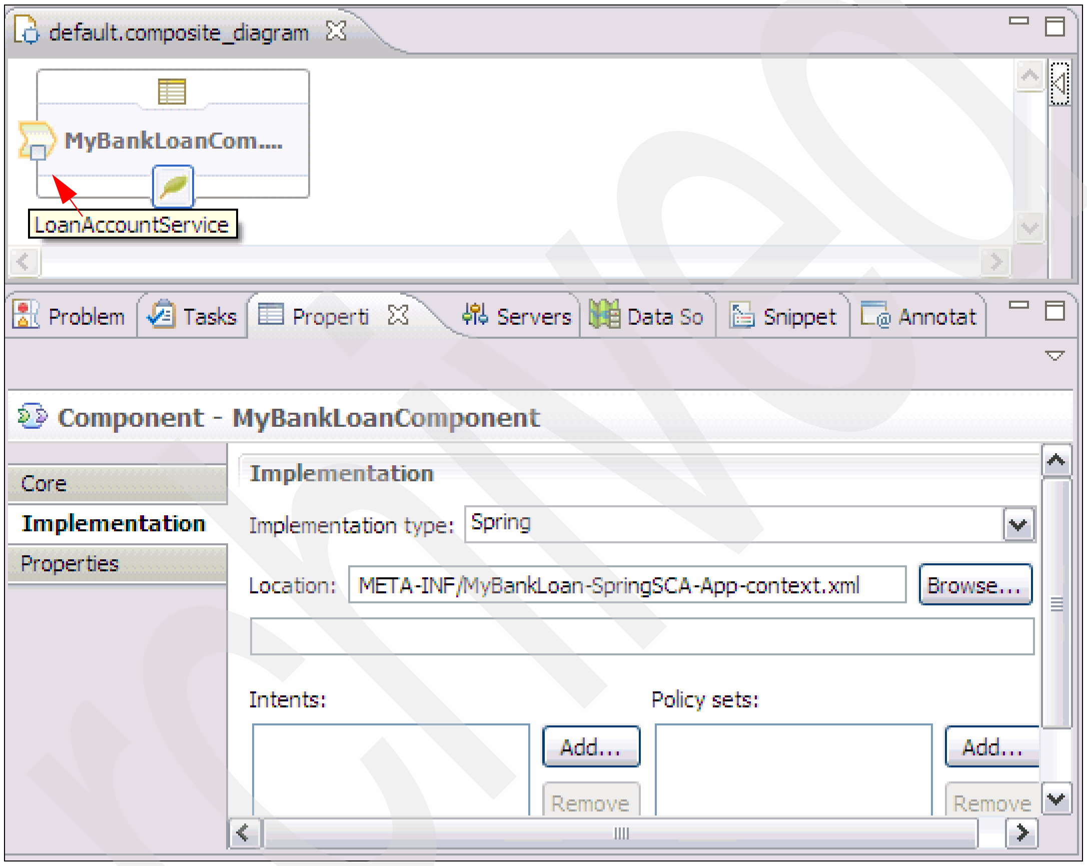
For more information about the Spring Framework, see:

http://www.springframework.org

9.1 Composite overview

In this example, we create a new composite that provides a loan account service, which allows users to view their loan balance. The composite can be invoked from the JSP that acts as the user interface.

Figure 9-1 shows the new MyBankLoanComposite in the Rational Application Developer work area. It has one component, MyBankLoanComponent, that is implemented with Spring. The service interface, LoanAccountService, can be used to invoke the component.



Component - MyBankLoanComponent

Implementation

Implementation type: Spring

Location: META-INF/MyBankLoan-SpringSCA-App-context.xml

Intents:

Policy sets:

Figure 9-1 MyBankLoanComposite

9.2 The Loan Account Spring application

This section describes the various artifacts that make up the Loan Account application. This application was created using the Spring Application Framework.

9.2.1 Implementation

Example 9-1 shows the `LoanAcctImpl.java` file. It contains a single Java bean that returns the current balance of the user's loan account.

Example 9-1 LoanAcctImpl.java

```
package com.mybank.loan.spring;
public class LoanAcctImpl implements LoanAcctInterface{
   private String balance;

   public LoanAcctImpl(){}

   public LoanAcctImpl  (String balance) {
      this.balance = balance;
   }
   public void getBalance() {
      System.out.println("Your Current Loan Account Balance is $" + balance);

   }
   public void setBalance (String balance){
      this.balance = balance;
   }

}
```

9.2.2 Interface

Example 9-2 shows the LoanAcctInterface, which is called from the service interface.

Example 9-2 LoanAcctInterface.java

```
package com.mybank.loan.spring;

public interface LoanAcctInterface {
   public void getBalance ();
}
```

9.2.3 Application context

Example 9-3 shows the Spring specific MyBankLoan application context file. This application context is used by the Spring container to load bean definitions and wire beans together. This example uses the constructor injection LoanAcctBalance bean.

Example 9-3 MyBankLoan-SpringApp-context.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Application context for the Loan Account -->
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:sca="http://www.springframework.org/schema/sca"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-2.5.xsd ">

   <bean id="LoanAcctBalance"
      class="com.mybank.loan.spring.LoanAcctImpl">
      <constructor-arg>
         <value>45000.00</value>
      </constructor-arg>
    </bean>
</beans>
```

Later, we demonstrate how to expose the beans that are declared in the application context as SCA services.

9.2.4 Application client for testing

Example 9-4 shows the application client used to test the Loan Account application.

Example 9-4 LoanAcctClient

```
public class LoanAcctClient {
   public static void main(String[] args) {

      try {
         System.out.println("Starting the lookup of Application Context");
         ApplicationContext appContext = new FileSystemXmlApplicationContext

("C:/projects/SCA-spring/workspace/MyBankLoanSpring/MyBankLoan-SpringApp-context.x
ml");
         if (appContext == null) {
               System.out.println("The appContext failed to Load " );
         }
         else {
            System.out.println("The appContext object is loaded");
      }
         LoanAcctInterface loanAcctInterface = (
               LoanAcctInterface)appContext.getBean("LoanAcctBalance");
         if (loanAcctInterface == null) {
            System.out.println("NO loanacctinterface object " );
         }
         else {
            System.out.println("The loanAcctinterface object is loaded");
         loanAcctInterface.getBalance();
      }
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

Invoking the client results in a message similar to the following message:

```
   Your current Loan Account Balance is $45000.00
```

9.3 Using the application as an SCA component

This section demonstrates how to implement a Java application based on the Spring framework as an SCA component. You can find the specification for the Spring component implementation at:

http://www.osoa.org/download/attachments/35/SCA_SpringComponentImplementationSpecification-V100.pdf?version=1

The Loan Account application was created using the Spring Framework with a single bean named LoanAcctBalance. This bean was declared in the application context along with a value. The sample client got a handle of the bean (with a lookup of LoanAcctBalance bean name) to invoke its method, `get Balance()`. The next step is to take the application and create it as an SCA component.

An SCA component implemented using implementation.spring is a Spring application that is created using a Spring application context. The Spring application context can contain one or more beans, and we show this in our example. In Spring, a bean is the most basic configuration unit. In SCA, a component is the most basic configuration unit.

The Loan Account Spring application participates in an SCA composite by exposing its LoanAcctBalance bean through a Java interface. The mechanism by which beans are mapped to component service and reference interfaces is the same as for the Java implementation, `implementation.java`. Spring beans can participate in an SCA composite as references or services without introducing any new metadata into the existing Spring application.

To use the implicit mapping from the Spring resources to the SCA resources there is no requirement to modify the existing application context file (Example 9-3 on page 125). To explicitly map resources, you need to add the following SCA schema to the application context:

```
http://www.osoa.org/xmlns/sca/1.0/spring-sca.xsd
```

For the Loan Account application, the SCA schema is added to the existing application context and the `<sca:service>` tag to explicitly define an SCA service called LoanAcctImpl, which is implemented by the LoanAcctBalance bean.

Example 9-5 shows the amended application context, stored as `MyBankLoan-SpringSCA-App-context.xml`. The updates are in bold font.

Example 9-5 Amended application context

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Revised SCA Application context for the Loan Account -->
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:sca="http://www.springframework.org/schema/sca"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
   http://www.springframework.org/schema/sca
   http://www.osoa.org/xmlns/sca/1.0/spring-sca.xsd">

   <sca:service name="LoanAccountService"
      type="com.mybank.loan.spring.LoanAcctInterface" target="LoanAcctBalance"/>

   <bean id="LoanAcctBalance"
      class="com.mybank.loan.spring.LoanAcctImpl">
      <constructor-arg>
         <value>45000.00</value>
      </constructor-arg>
    </bean>
</beans>
```

Example 9-6 shows the composite file for MyBankLoanComposite. MyBankLoanComponent. uses the implementation.spring implementation. The service name is consistent with the Service name defined in the SCA application context (see Example 9-5 on page 127).

Example 9-6 default.composite

```
<?xml version="1.0" encoding="UTF-8"?>
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" autowire="false" name="MyBankLoanComposite"
targetNamespace="http://mybankloan">
  <component name="MyBankLoanComponent">
    <implementation.spring  location="META-INF/MyBankLoan-SpringSCA-App-context.xml"/>
    <service name="LoanAccountService">
      <interface.java interface="com.mybank.loan.spring.LoanAcctInterface"/>
    </service>
  </component>
</composite>
```

Consider the following list of constraints when implementing Spring applications as SCA components:

- Exposing a Spring bean as a service is not supported when the bean implements multiple interfaces. To resolve this problem, explicitly define a `<sca:service>` element in the Spring application context file. If no explicit definition of `<sca:service>` is available, the problems remains by default when you expose all the beans as defined in the Spring context as services.
- Callbacks are not supported.
- Pass-by-reference is not supported.
- The only supported bindings are Web services, EJB, JMS and SCA default binding.
- The support provided is for Spring applications that follow the Java SE model. The Spring Framework in SCA uses the Java SE model and, similar to SCA applications, does not have access to the local namespace, which is required for elements accessed using `java:comp`.
- Users must consider using explicit or direct JNDI lookup.
- SCA annotations are not supported in Spring implementations.

9.4 Packaging the application for deployment

Two packages are required for this application:

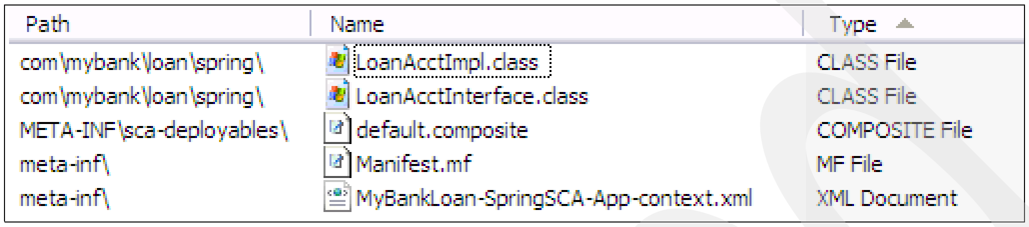
- The JAR file with the SCA composite file for the application
- A JAR file that contains the Spring and Feature Pack for SCA runtime libraries for Spring

9.4.1 Create the JAR file with the composite

The first step is to package the SCA application into an SCA archive (JAR) file. To ensure successful deployment ensure that the composite deployment descriptor is named `default.composite` and that the `default.composite` file is located in the `META-INF/sca-deployables` directory.

Note: For more information about the `default.composite` file and the `sca-deployables` directory, see the Note box on page 12. If you are using Rational Application Developer, see the Tip box on page 35 for information about setting preferences to help you create this structure.

The `loan-account-spring.jar` file contains the files shown in Figure 9-2.



Path	Name	Type
com\mybank\loan\spring\	LoanAcctImpl.class	CLASS File
com\mybank\loan\spring\	LoanAcctInterface.class	CLASS File
META-INF\sca-deployables\	default.composite	COMPOSITE File
meta-inf\	Manifest.mf	MF File
meta-inf\	MyBankLoan-SpringSCA-App-context.xml	XML Document

Figure 9-2 The loan-account-spring.jar file

9.4.2 Create the JAR file with the runtime libraries

The second step is to create a runtime JAR file that contains the Spring run time and Spring Feature Pack for SCA files:

1. Create a temporary directory, for example, the `c:\temp\spring` directory.
2. Download the `spring-framework-2.5.5.zip` file from the Spring Community downloads at:

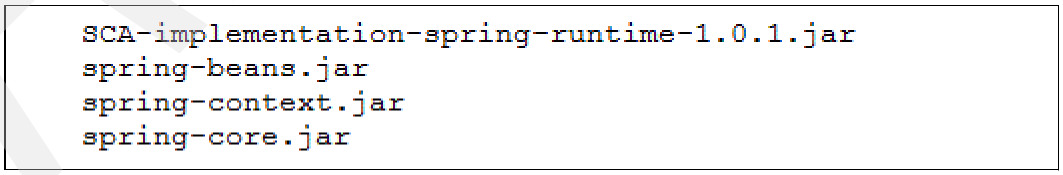
   `http://www.springsource.com/products/spring-community-download`

   Then, extract that file to a temporary directory.
3. Extract the following Spring files to the temporary directory:
   - `spring-beans.jar`
   - `spring-context.jar`
   - `spring-core.jar`
4. Copy the following Feature Pack for SCA file to the temporary directory:

   `app_server_root``\feature_packs\sca\optionalLibraries\`
   `SCA-implementation-spring-runtime-1.0.1.jar`
5. In the temporary directory, run the Java Jar utility to create the `SpringSharedLibAsset.jar` file:

   ```
   jar cvf SpringSharedLibAsset.jar *
   ```

   Figure 9-3 shows a list of the files in the `SpringSharedLibAsset.jar` file.



```
SCA-implementation-spring-runtime-1.0.1.jar
spring-beans.jar
spring-context.jar
spring-core.jar
```

Figure 9-3 The SpringSharedLibAsset.jar file

9.5 Deploying the application

To deploy the application to the SCA run time in WebSphere Application Server:

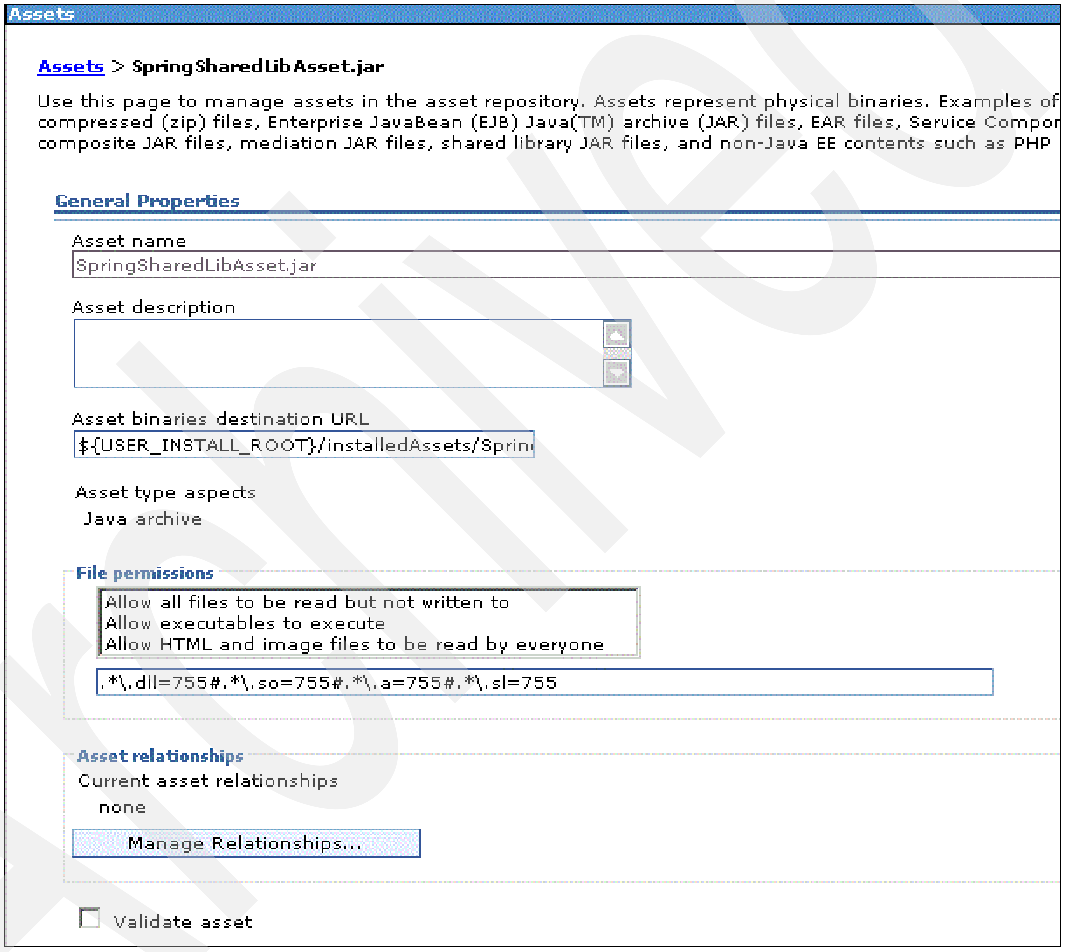
1. Import the Spring run time library JAR file as an asset.

   a. From the administrative console select **Applications** → **Applications Types** → **Assets**. Click **Import**.

   b. Locate the `SpringSharedLibAsset.jar` file from the file system, and click **Next**.

   c. Proceed through the remainder of the wizard screens taking the default options and values. Click **Finish** on the last panel.

   Figure 9-4 shows the resulting asset.



Assets

Assets > SpringSharedLibAsset.jar

Use this page to manage assets in the asset repository. Assets represent physical binaries. Examples of compressed (zip) files, Enterprise JavaBean (EJB) Java(TM) archive (JAR) files, EAR files, Service Compon composite JAR files, mediation JAR files, shared library JAR files, and non-Java EE contents such as PHP

General Properties

Asset name
SpringSharedLibAsset.jar

Asset description

Asset binaries destination URL
${USER_INSTALL_ROOT}/installedAssets/Sprin

Asset type aspects
Java archive

File permissions
Allow all files to be read but not written to
Allow executables to execute
Allow HTML and image files to be read by everyone
.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755

Asset relationships
Current asset relationships
none

Manage Relationships...

☐ Validate asset

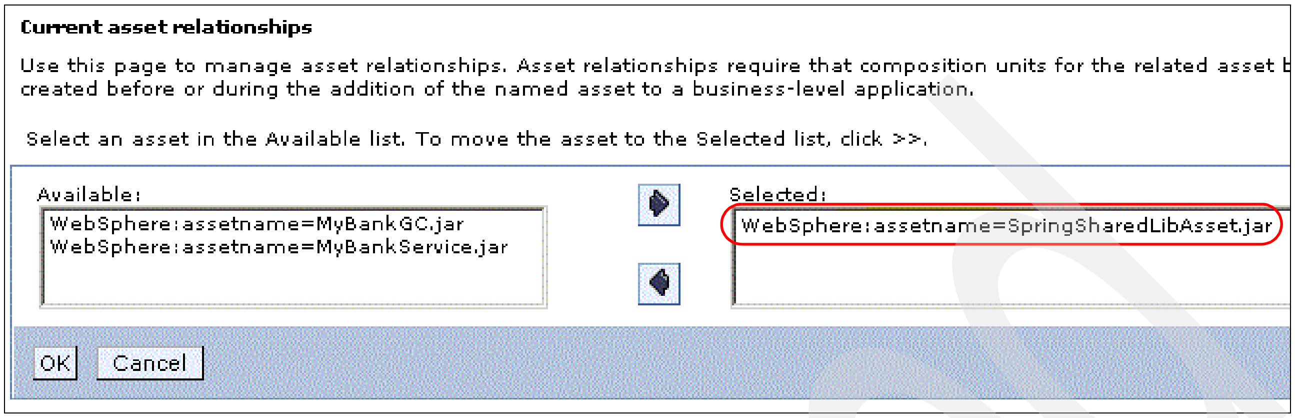
Figure 9-4 SpringSharedLibAsset asset properties

2. Import the application JAR file.

   a. From the administrative console navigate to **Applications** → **Applications Types** → **Assets**. Click **Import**.

   b. Locate the `loanaccount-spring.jar` file from the file system, and click **Next**.

c. Click the **Manage Relationships** key to open the panel shown in Figure 9-5. Select the `SpringSharedLibAsset.jar` file in the list of available assets and click the top arrow to move it to the list of selected assets.



Current asset relationships

Use this page to manage asset relationships. Asset relationships require that composition units for the related asset b
created before or during the addition of the named asset to a business-level application.

Select an asset in the Available list. To move the asset to the Selected list, click >>.

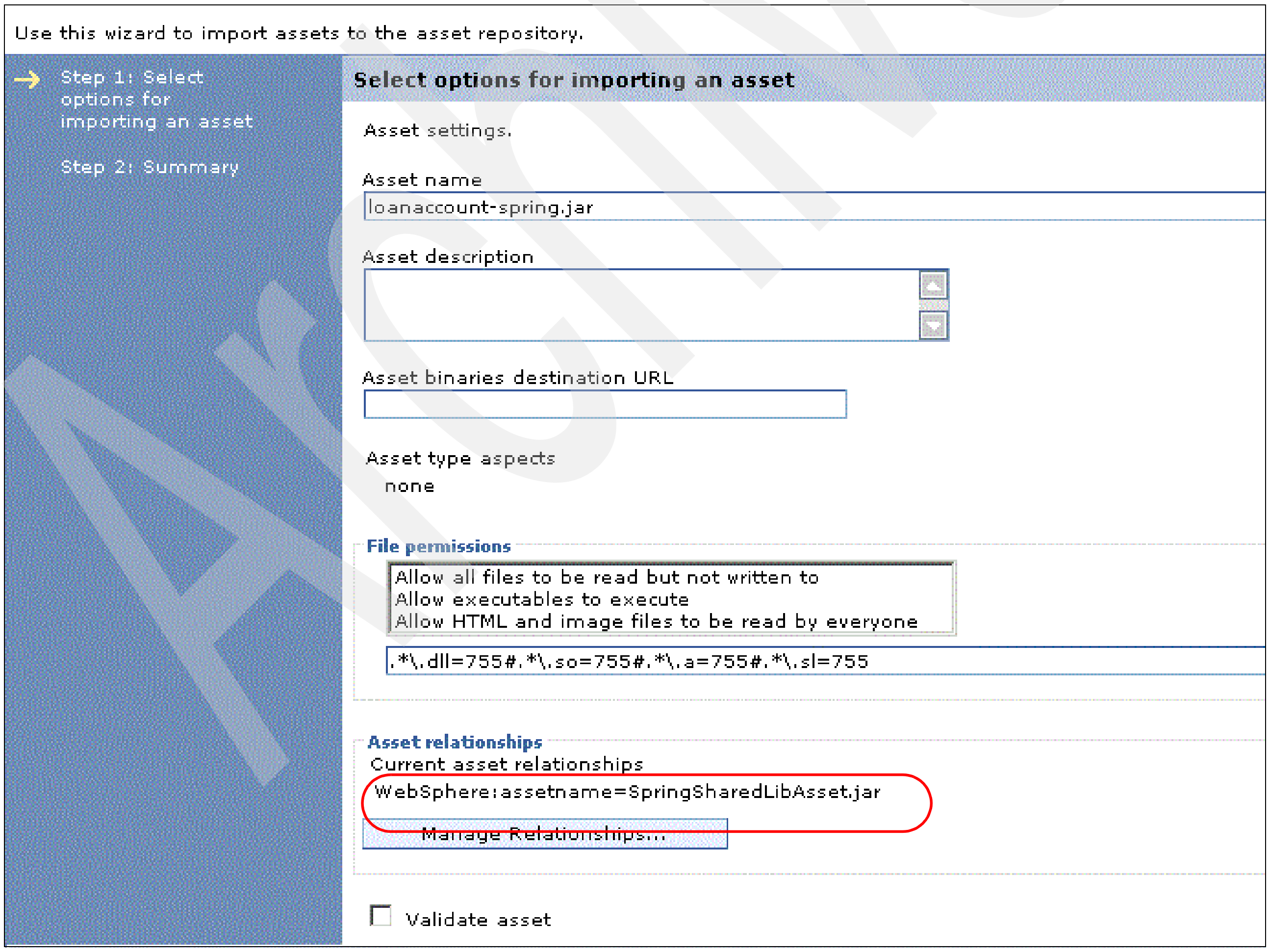
Available:
WebSphere:assetname=MyBankGC.jar
WebSphere:assetname=MyBankService.jar

Selected:
WebSphere:assetname=SpringSharedLibAsset.jar

Figure 9-5 Set the relationship between the application and the runtime library

d. Click **OK**. The wizard window (Figure 9-6) now shows the relationship that is established between the application and the runtime files.

e. Click **Next** and then click **Finish** to complete the import.



Use this wizard to import assets to the asset repository.

Select options for importing an asset

Asset settings.

Asset name
loanaccount-spring.jar

Asset description

Asset binaries destination URL

Asset type aspects
none

File permissions
Allow all files to be read but not written to
Allow executables to execute
Allow HTML and image files to be read by everyone
.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755

Asset relationships
Current asset relationships
WebSphere:assetname=SpringSharedLibAsset.jar

☐ Validate asset

Figure 9-6 Setting the relationships in the import wizard

3. Create a business-level application, and add the `loan-account-spring.jar` file as an asset to add `theSpingSharedLibAsset.jar` file as a shared library automatically.

   a. Using the administrative console select **Applications** → **Applications Types** → **Business Level Applications**.

   b. Create a new business-level application.

   c. Add the `loanaccount-spring.jar` file to the application.

   d. Start the business-level application.

Related publications

We consider the publications that we list in this section particularly suitable for a more detailed discussion of the topics that we cover in this paper.

IBM Redbooks publications

For information about ordering these publications, see "How to get IBM Redbooks publications" on page 134. Note that some of the documents referenced here might be available in softcopy only.

- *IBM WebSphere Application Server V7.0 Web Services Guide*, SG24-7758

  `http://www.redbooks.ibm.com/abstracts/sg247758.html?Open`

Online resources

The following Web sites are also relevant as further information sources:

- Feature Pack for SCA, Version 1.0.1 Information Center

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.soafep.multiplatform.doc/info/ae/ae/welcome_fepsca.html`
- OSOA home page

  `http://osoa.org`
- *Exploring the WebSphere Application Server Feature Pack for SCA, Part 1: An overview of the Service Component Architecture feature pack*, developerWorks article

  `http://www.ibm.com/developerworks/websphere/library/techarticles/0812_beck/0812_beck.html`
- *Exploring the WebSphere Application Server Feature Pack for SCA, Part 2: Web services policy sets*, developerWorks article

  `http://www.ibm.com/developerworks/websphere/library/techarticles/0901_coats/0901_coats.html`
- *Exploring the WebSphere Application Server Feature Pack for SCA, Part 3: Intents and policies*, developerWorks article

  `http://www.ibm.com/developerworks/websphere/library/techarticles/0902_beck/0902_beck.html`
- *Exploring the WebSphere Application Server Feature Pack for SCA, Part 4: SCA Java annotations and component implementation*, developerWorks article

  `http://www.ibm.com/developerworks/websphere/library/techarticles/0902_beck2/0902_beck2.html`
- *Exploring the WebSphere Application Server Feature Pack for SCA, Part 5: Protocol bindings for Service Component Architecture services*, developerWorks article

  `http://www.ibm.com/developerworks/websphere/library/techarticles/0904_beck/0904_beck.html`

- *Exploring the WebSphere Application Server Feature Pack for SCA, Part 6: Using Spring with Service Component Architecture*, developerWorks article

  http://www.ibm.com/developerworks/websphere/library/techarticles/1001_beck6/1001_beck6.html

- *Exploring the WebSphere Application Server Feature Pack for SCA, Part 7: Using Atom and JSON-RPC for Web 2.0 support*, developerWorks article

  http://www.ibm.com/developerworks/websphere/library/techarticles/1001_beck7/1001_beck7.html

- IBM Education Assistant

  http://publib.boulder.ibm.com/infocenter/ieduasst/v1r1m0/index.jsp?topic=/com.ibm.iea.wasfpsca/wasfpsca/1.0.1/Overview/WASv7SCA101_Overview_SCAfepIntroduction/player.html

- WebSphere Application Server v7 Feature Pack for Service Component Architecture (SCA)

  http://www-01.ibm.com/software/webservers/appserv/was/featurepacks/sca/features/

- Trace Analyzer for WebSphere Application Server

  http://www.alphaworks.ibm.com/tech/ta4was

- Recommended fixes for WebSphere Application Server

  http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg27004980#ver70

- IBM Installation Manager for WebSphere

  http://www-01.ibm.com/support/docview.wss?rs=180&uid=swg24023498

- SCA Service Component Architecture, Spring Component Implementation Specification

  http://www.osoa.org/download/attachments/35/SCA_SpringComponentImplementationSpecification-V100.pdf?version=1

- Spring Community Downloads

  http://www.springsource.com/products/spring-community-download

How to get IBM Redbooks publications

You can search for, view, or download IBM Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Getting Started with WebSphere Application Server Feature Pack for Service Component Architecture

Learn about the new features

Build, deploy, and manage SCA applications

Apply Qualities of Service

Service Component Architecture (SCA) defines a service-based model for building business process applications using an SOA approach. This ability to drive a business process using individual, reusable services is the heart of the SOA concept. With IBM WebSphere Application Server Feature Pack for Service Component Architecture, you can deploy SCA applications to WebSphere Application Server.

This IBM Redpaper publication provides a starting point for using the Feature Pack for SCA. It provides an architectural view of SCA and of the Feature Pack. In addition, this paper explains how to create simple SCA components from existing Java and Spring implementations. It discusses how to apply quality of service to applications, and how to deploy and manage SCA artifacts in WebSphere Application Server. The examples in this paper use Rational Application Developer to illustrate how to create and package SCA applications.

REDP-4633-00