Oracle® Fusion Middleware

Developing WebLogic SCA Applications for Oracle WebLogic Server
## Contents

### Preface
- Audience vi
- Documentation Accessibility vi
- Conventions vi

### 1 Introduction and Roadmap
- Document Scope and Audience 1-1
- Guide to This Document 1-1
- Related Documentation 1-2

### 2 Overview
- WebLogic Spring SCA Applications 2-1
- WebLogic SCA Runtime 2-1
- Limitations 2-2

### 3 Tools Support
- Standalone WebLogic Server 3-1
- Oracle SOA Suite 3-1
- Oracle JDeveloper 3-1
- Oracle Enterprise Pack for Eclipse (OEPE) 3-2

### 4 Deploying WebLogic SCA Runtime to WebLogic Server
- Deploying a Library Using the WebLogic Server Administration Console 4-1

### 5 WebLogic Spring SCA Application Overview
6 Configuring the Spring Application Context

Specifying References and Services  6-1
  sca:reference Element  6-1
  sca:service Element  6-1
  Binding Subelements  6-2
Sample Spring Application Context  6-2

7 Configuring EJB Session Bean Bindings

binding.ejb Element Attributes  7-1
Binding to Services  7-2
Binding to References  7-2
  EJB2 Programming Model  7-3
  EJB3 Programming Model  7-3

8 Configuring Web Service Bindings

binding.ws Element Attributes  8-2
binding.ws Subelements  8-3
  PolicyReference Element  8-3
  property Element  8-4
Configuring Security  8-4
  Security Configuration Examples  8-4
Configuring Databinding  8-6
  Configuring TopLink/EclipseLink JAXB Binding  8-6
  Configuring TopLink/EclipseLink SDO Databinding  8-6
  Configuring Glassfish JAXB Databinding  8-7
About Configuring Custom Databinding  8-7
  Configuring Custom Databinding  8-7
  Custom Databinding Examples  8-8
Configuring Databinding for SOAP Attachments  8-9
  Configuring Attachments Using MTOM  8-10
  Configuring Attachments Using SwA  8-12
Configuring Collection and Map Objects  8-14
  Externalizing Generic Type for Map  8-15
Precedence of Configuration Settings  8-17
Deployment  8-17
Runtime  8-17
9  Deploying WebLogic Spring SCA Applications

Preparing Deployment Units 9-1
Configuring the Deployment Descriptor 9-2
Bundling Libraries 9-2
Deploying in a Cluster 9-3
Deploying WebLogic Spring SCA Applications Using Other Tools 9-3
Runtime 9-3

10  Viewing WebLogic SCA Application Configurations

Prerequisites for Viewing Application Configurations 10-1
Enabling the WebLogic Server Administration Console Extension 10-1
Viewing Details about Configured Services and References 10-2

A  WebLogic SCA Schemas

WebLogic Spring SCA Schema (weblogic-sca.xsd) A-1
WebLogic SCA Binding Schema (weblogic-sca-binding.xsd) A-1
Web Service Policy Schema (ws-policy.xsd) A-1
WebLogic SCA Databindings Customization Descriptor Schema (weblogic-wsee-databinding.xsd) A-2
Preface

This preface describes the document accessibility features and conventions used in this guide—Developing WebLogic SCA Applications for Oracle WebLogic Server.

Audience

This document is intended for administrators who configure WebLogic Server to host applications and for application developers who develop WebLogic Spring SCA applications.

Documentation Accessibility

For information about Oracle’s commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Accessible Access to Oracle Support

Oracle customers who have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></td>
<td>Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.</td>
</tr>
<tr>
<td><em>italic</em></td>
<td>Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.</td>
</tr>
<tr>
<td>monospace</td>
<td>Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.</td>
</tr>
</tbody>
</table>
1

Introduction and Roadmap

WebLogic SCA provides support for creating and deploying WebLogic SCA applications. This book describes how to work with WebLogic SCA in Oracle WebLogic Server.

Document Scope and Audience

The following topics describe the contents and organization of this guide—Developing WebLogic SCA Applications for Oracle WebLogic Server:

• Document Scope and Audience
• Guide to This Document
• Related Documentation

Guide to This Document

• This chapter, Introduction and Roadmap describes the organization of this guide.
• Overview provides a brief overview of Oracle SCA.
• Tools Support describes the tools available for working with WebLogic SCA in Oracle WebLogic Server, Oracle SOA Suite, Oracle Enterprise Pack for Eclipse (OEPE), and Oracle JDeveloper.
• Deploying WebLogic SCA Runtime to WebLogic Server describes how to deploy WebLogic SCA Runtime to WebLogic Server as a shared application library.
• WebLogic Spring SCA Application Overview provides an overview of what is required to create and deploy a WebLogic Spring SCA application.
• Configuring the Spring Application Context tells how to configure the Spring application context to take advantage of WebLogic SCA features.
• Configuring EJB Session Bean Bindings tells how to configure EJB Session Bean bindings to expose services and references in WebLogic Spring SCA applications.
• Configuring Web Service Bindings tells how to configure Web Service bindings to expose services and references in WebLogic Spring SCA applications.
• Deploying WebLogic Spring SCA Applications describes how to deploy WebLogic Spring SCA applications to Oracle WebLogic Server.
• Viewing WebLogic SCA Application Configurations tells how to enable the WebLogic SCA console extension to view the services and references configured for a WebLogic Spring SCA application.
• WebLogic SCA Schemas shows the schemas used for WebLogic Spring SCA applications.
Related Documentation

See the following documentation for related information:

- For information about Oracle WebLogic Server, see *Understanding Oracle WebLogic Server*.
- For information about Oracle SOA Suite, see the Oracle SOA Suite Documentation page at:
  http://docs.oracle.com/docs/cd/E23943_01/soa.htm
- WebLogic SCA is based in part on the following specifications:
  - OASIS SCA Service Component Architecture Assembly Model Specification at:
    http://www.oasis-open.org/sca-assembly
  - OASIS SCA Service Component Architecture Spring Component Implementation Specification at:
  - OASIS SCA Service Component Architecture EJB Session Bean Binding specification at:
    http://www.oasis-open.org/sca-bindings
- See Spring Source Community for information about the Spring Framework at:
  http://www.springsource.org/
Overview

This chapter describes how Oracle WebLogic Server SCA provides a model for building enterprise applications and systems as modular business services that can be integrated and reused. WebLogic SCA provides support for developing and deploying SCA applications using Plain Old Java Objects (POJOs).

- WebLogic Spring SCA Applications
- WebLogic SCA Runtime
- Limitations

WebLogic SCA is based on a subset of the OASIS Service Component Architecture Spring Component Implementation Specification at http://www.oasis-open.org/committees/download.php/25529/sca-springci-draft-20070926.doc. Features not supported are listed in Limitations.

WebLogic Spring SCA Applications

In SCA, the implementation of a component and its communication are clearly separated. In WebLogic SCA, you can write Java applications using POJOs and, through different protocols, expose components as SCA services and access other components via references. You do this using SCA semantics configured in a Spring application context. In SCA terms, a WebLogic Spring SCA application is a collection of POJOs plus a Spring SCA context file that wires the classes together with SCA services and references.

WebLogic Spring SCA applications run seamlessly in WebLogic Server (via the WebLogic SCA Runtime) and can be used without modification, as components in Oracle SOA composites.

WebLogic SCA Runtime

In WebLogic Sever, WebLogic Spring SCA applications run in the WebLogic SCA Runtime. The runtime must be deployed to WebLogic Server as a shared Web application library before applications can be deployed to it. WebLogic SCA Runtime includes the following:

- A Spring container with support for configuring SCA references and services using the WebLogic SCA schemas.
- A message processor that routes incoming messages (only Java parameters supported) to the appropriate services.
- SCA binding component implementations responsible for listening for requests, publishing services, and invoking references. Two binding component implementations are included in this release: Web Service and EJB session bean.

Figure 2-1 shows a representation of a deployed WebLogic Spring SCA application.
Limitations

WebLogic SCA does not support some features from the OASIS Service Component Architecture Spring Component Implementation Specification. The limitations are as follows:

- For WebLogic SCA in standalone WebLogic Server, service and reference bindings are specified in the Spring context only, not in a separate SCDL (.composite) file. Using SCDL files to specify service and reference bindings is available only in Oracle SOA Suite.
- SCA annotations are not supported in this release.
- The `<sca:property>` element is not supported in this release.

The following are not supported for EJB session bean bindings in this release:

- EJB policies
- Stateful EJB bindings
- EJB 2.x service bindings
- `<ejb-link>` element
- Local EJB service bindings

The following limitations apply to Web Service bindings in this release:

- For SCA references, the type used must be a JAX-WS compatible interface generated from the external WSDL, using a JAX-WS compatible client generation tool such as the JAX-WS `wsimport` tool, the WebLogic `clientgen` Ant task, Oracle JDeveloper, or Oracle Enterprise Pack for Eclipse (OEPE).
This chapter describes tools that support Oracle WebLogic Server SCA, that is, Oracle SOA Suite, Oracle JDeveloper, and Oracle Enterprise Pack for Eclipse (OEPE).

- Standalone WebLogic Server
- Oracle SOA Suite
- Oracle JDeveloper
- Oracle Enterprise Pack for Eclipse (OEPE)

## Standalone WebLogic Server

In addition to all the other WebLogic Server features for developing, deploying, managing, and monitoring applications—including the WebLogic SCA Runtime—the WebLogic Server Administration Console provides runtime monitoring features for WebLogic Spring SCA applications.

See [Viewing WebLogic SCA Application Configurations](#).

## Oracle SOA Suite

WebLogic Spring SCA applications can be used as components in SCA assembled as components of SCA composites in Oracle SOA.

See [Developing SOA Applications with Oracle SOA Suite](#).

## Oracle JDeveloper

You can use Oracle JDeveloper to do the following:

- Create Spring SCA context files using the `sca:service`, `sca:reference`, and binding elements
- Define and generate WebLogic SCA service definitions
- Add WebLogic SCA support to existing project types
- Configure the project classpath and the `weblogic.xml` configuration file to have access to the WebLogic SCA shared library
- Bundle WebLogic Spring SCA applications as part of any Java EE deployment unit, including EAR and WAR archives


See the online documentation included in JDeveloper.
Oracle Enterprise Pack for Eclipse (OEPE)

You can use Oracle Enterprise Pack for Eclipse to do the following:

- Deploy the WebLogic SCA Runtime as a WebLogic Server shared library
- Create Spring SCA context files using the `sca:service`, `sca:reference`, and binding elements
- Define and generate WebLogic SCA service definitions
- Add WebLogic SCA support to existing project types
- Configure the project classpath and the `weblogic.xml` configuration file to have access to the WebLogic SCA shared library
- Bundle WebLogic Spring SCA applications as part of any Java EE deployment unit, including EAR and WAR archives

For more information about WebLogic SCA support in Oracle Enterprise Pack for Eclipse, see “Oracle WebLogic Server Support: Configuring a Project to Use WebLogic SCA” at:

Deploying WebLogic SCA Runtime to WebLogic Server

This chapter describes WebLogic SCA Runtime, which is installed by default with Oracle WebLogic Server. However, it must be deployed as a shared Web application library.

Note:

WebLogic SCA applications are expected to bundle spring.jar, so the Spring Framework does not have to be installed separately during WebLogic SCA Runtime deployment.

Deploying a Library Using the WebLogic Server Administration Console

In WebLogic Server, you can deploy the library using the WebLogic Server Administration Console or the command-line weblogic.Deployer tool.

For information about weblogic.Deployer, see Deploying Applications and Modules with weblogic.Deployer in Deploying Applications to Oracle WebLogic Server.

To deploy the library using the WebLogic Server Administration Console:

1. If you have not already done so, in the Change Center of the WebLogic Server Administration Console, click Lock & Edit.
2. In the left pane of the WebLogic Server Administration Console, click Deployments.
3. In the right pane, click Install.
4. In the Install Application Assistant, navigate to the library file:
   
   WL_HOME/common/deployable-libraries/weblogic-sca-1.0.war
5. Select the file and click Next.
6. Select Install this deployment as a library.
7. Click Next.
8. Select the servers and/or clusters to which you want to deploy the application or module.
9. Click Next.
10. Optionally update additional deployment settings.
11. Click Next.
12. Review the configuration settings you have specified, and click Finish to complete the installation.

13. To activate these changes, in the Change Center of the WebLogic Server Administration Console, click Activate Changes.

For more information about deployment in WebLogic Server, see Understanding WebLogic Server Deployment in Deploying Applications to Oracle WebLogic Server.

**Note:**

You can deploy the WebLogic SCA Runtime library automatically by using the Oracle Enterprise Pack for Eclipse and the Oracle JDeveloper.

See Tools Support for more information on WebLogic SCA support in these tools.
WebLogic Spring SCA Application Overview

This chapter describes the WebLogic Spring SCA applications that are deployed to Oracle WebLogic Server. WebLogic Spring SCA applications can be run in a standalone instance of Oracle WebLogic Server or can be assembled as components of SCA composites in Oracle SOA.

Note:

WebLogic SCA applications need to bundle their own `spring.jar`, version 2.0.6 or higher. Oracle recommends Spring 2.5.6.

WebLogic Spring SCA applications have the following characteristics:

- The Spring application context file includes SCA elements to configure SCA references, services, and their bindings. See Configuring the Spring Application Context, Configuring EJB Session Bean Bindings, and Configuring Web Service Bindings.

- Deployment units are organized following guidelines described in Deploying WebLogic Spring SCA Applications.
6

Configuring the Spring Application Context

This chapter describes how the Oracle WebLogic Server SCA services and references are configured in the Spring application context. An SCA service represents program code that provides a business function that is offered for use by other components. Services can be invoked as SCA references by the other components.

• Specifying References and Services
• Sample Spring Application Context


Specifying References and Services

WebLogic Spring SCA supports the following elements to specify SCA references and bindings:

• sca:reference Element
• sca:service Element

sca:reference Element

The <sca:reference> element declares a Spring bean representing an SCA service external to the Spring application context. This element takes the following attributes:

name
Required. The name of the reference.

type
Required. The fully-qualified Java type of the interface or class representing the remote service. For example, if the external reference is to a Web Service, this would be the type of the client-side proxy to the Web Service. For SCA references using a Web Service binding, the type used must be a JAX-WS compatible interface. The type used must be a JAX-WS compatible interface, generated from the external WSDL, using a JAX-WS compatible client generation tool such as the JAX-WS wsimport tool, the WebLogic clientgen Ant task, Oracle JDeveloper, or Oracle Enterprise Pack for Eclipse (OEPE).

default
Optional. The target bean for the reference if none is specified. This will improve performance by wiring to a local service, ignoring associated bindings.

sca:service Element

The <sca:service> element declares a Spring bean that WebLogic SCA exposes as a service. This element takes the following attributes:
name
Required. The name of the service.
If a name is not specified in the name attribute of a binding.ws subelement (see binding.ws Element Attributes), the name specified in the name attribute of the sca:service is published as the service name in the WSDL. However, if a binding.ws specifies a name, that name is published in the WSDL as the service name for that binding.

type
Required. The fully-qualified Java type of the Java class to be exposed as an SCA service.

target
Required. The bean to be exposed as a service.

See WebLogic Spring SCA Schema (weblogic-sca.xsd) for the WebLogic Spring SCA schema.

Binding Subelements
sca:reference elements and sca:service elements contain binding subelements to specify the binding(s) for the reference or service. An sca:reference element can have only one binding subelement. If more are specified, only the first one is used. An sca:service element can have none, one, or more binding subelements.

WebLogic Spring SCA supports the following binding elements:
• <binding.ws> specifies that the binding is a Web Service binding.
• <binding.ejb> specifies that the binding is an EJB session bean binding.
• <binding.sca> is the default. If binding.sca is specified or if no binding is specified, WebLogic SCA Runtime defaults to binding.ws.

See Configuring EJB Session Bean Bindings and Configuring Web Service Bindings for more information on the binding elements and the WebLogic SCA Runtime binding component implementations.

See WebLogic SCA Schemas for the binding element schemas.

Sample Spring Application Context

The following example shows a spring application context. The context includes two Spring beans, beanX and beanY. The bean beanX represents the entry point from WebLogic Spring SCA into the Spring application context and the bean beanY includes a reference to an external SCA service.

<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:sca="http://xmlns.oracle.com/weblogic/weblogic-sca"
xmlns:wlsb="http://xmlns.oracle.com/weblogic/weblogic-sca-binding"
binding.xsd">
    <!-- Expose the bean "X" as an SCA service named "SCAService"-->
    <sca:service name="SCAService"
        type="org.xyz.someapp.SomeInterface"
        target="X">
      <wlsb:binding.ws uri="/testService"/>
    </sca:service>
    <sca:reference name="SCAReference" type="org.xyz.someapp.SomeOtherInterface">
      <wlsb:binding.ws
          location="http://localhost:7001/jscaliteapp/myrefsvcnameuri"
          port="http://test.oracle.com#wsdl.endpoint(SCAService2/myrefportname)"/>
    </sca:reference>
    <bean id="X" class="org.xyz.someapp.SomeClass">
      <property name="foo" ref="Y"/>
    </bean>
    <bean id="Y" class="org.xyz.someapp.SomeOtherClass">
      <property name="bar" ref="SCAReference"/>
    </bean>
</beans>
This chapter describes how to configure EJB session bean bindings for SCA services and SCA references in Oracle WebLogic Server.

- **binding.ejb Element Attributes**
- **Binding to Services**
- **Binding to References**

The EJB session bean binding can be applied to both SCA services and SCA references. Configure EJB session bean bindings in the `<binding.ejb>` element in the Spring application context file for your application.

- For general information about configuring the Spring application context configuration for a WebLogic Spring SCA application, see Configuring the Spring Application Context.
- For the schema that defines the `<binding.ejb>` element, see WebLogic SCA Binding Schema (weblogic-sca-binding.xsd).

**Note:**

The EJB Binding Component implementation is based on the SCA EJB Session Bean Binding specification at [http://www.oasis-opencsa.org/sca-bindings](http://www.oasis-opencsa.org/sca-bindings).

However, in this release of WebLogic SCA, the following features are not supported:

- Stateful session bean binding (conversations are not supported)
- `<ejb-link>` elements
- Local EJB service binding
- Local EJB reference binding

**binding.ejb Element Attributes**

Attributes of the `binding.ejb` element are as follows:

- **name**
  Optional. Specifies the name of the binding.

- **uri**
  Required. For EJB bindings on references, `uri` specifies the JNDI name of the target EJB. For EJB bindings on services, the `uri` value is the JNDI name at which the EJB is bound.
Advanced CORBA name URIs are not supported. The following two simplified patterns are supported. Both result in the service implementation being bound to the JNDI name ejb/MyHome:

- uri="corbaname:rir:#ejb/MyHome"
- uri="ejb/MyHome"

**dispatchPolicy**

Optional. Used with service bindings. Specifies the name of a WorkManager to be used for incoming invocations.

### Binding to Services

When `binding.ejb` is configured on an SCA service, an implementation of the service interface becomes available in JNDI when the application is deployed. That service can then be invoked using the EJB programming model. For example, this allows a client to look up and invoke the service as if the underlying service were a deployed EJB. This may be useful in a situation where you want to replace an existing EJB service with an SCA implementation, without requiring clients of the legacy EJB to be updated.

Only the EJB3 programming model is supported for EJB session bean service bindings.

Use the `uri` attribute of `binding.ejb` to specify the JNDI name.

See [binding.ejb Element Attributes](#).

All EJB service bindings are remote and can therefore be looked up and invoked remotely.

Remote service implementations are clusterable. You can configure a client timeout for service bindings. To specify a client timeout, edit the `EJBServiceDelegateImplRTD.xml` file located in the `binding.ejb` jar file.

EJB service bindings can optionally specify a dispatch policy. To configure a dispatch policy, set the `dispatchPolicy` attribute of `<binding.ejb>` to the name of the WorkManager to be used for incoming invocations. If the configured WorkManager is not found, a warning is issued at runtime and the default WorkManager is used.

### Binding to References

When `binding.ejb` is configured on an SCA reference, the target EJB is resolved and invoked without any knowledge of the EJB programming model. (WebLogic SCA Runtime determines the programming model of the target EJB based on whether the object returned from the JNDI lookup implements `javax.ejb.EJBHome`.)

The target EJB is resolved lazily, that is, it is resolved at runtime as necessary to service an invocation.

This allows an SCA POJO implementation to invoke a deployed EJB in the same way it invokes any other SCA reference. WebLogic SCA Runtime looks up the target EJB from JNDI and delegates any method calls to the EJB. Using `binding.ejb` in this way provides the flexibility to replace an EJB service with a non-EJB service without having to update the dependent component implementation.
The EJB2 and EJB3 programming models are supported for reference bindings, as described below.

**EJB2 Programming Model**

If the target EJB uses an EJB 2.x client view, the binding implementation invokes the `home.create()` method to obtain the EJB's remote interface implementation. The binding implementation also translates reference interface method invocations to EJB remote interface invocations.

Methods of the reference interface are mapped to the remote interface according to the rules in *SCA EJB Session Bean Binding* specification at [http://www.oasis-opencsa.org/sca-bindings](http://www.oasis-opencsa.org/sca-bindings).

**EJB3 Programming Model**

If the EJB3 programming model is used, the target EJB may implement the reference interface, but it is not required to so. If it does not implement the reference interface, the rules in Section 2.2 of the *SCA EJB Session Bean Binding* specification apply. The EJB binding code is responsible for translating reference interface method invocations to EJB business interface invocations.
This chapter describes how to configure Web Service bindings for SCA services and SCA references in Oracle WebLogic Server.

- **binding.ws Element Attributes**
- **binding.ws Subelements**
- **Configuring Security**
- **Configuring Databinding**
- **Precedence of Configuration Settings**
- **Deployment**
- **Runtime**

You can configure Web Service bindings in the `<binding.ws>` element in the Spring application context file for your application.

- For general information about configuring the Spring application context for a WebLogic Spring SCA application, see Configuring the Spring Application Context.
- For the schema that defines the `<binding.ws>` element, see WebLogic SCA Binding Schema (weblogic-sca-binding.xsd).

You can apply the Web Service binding to SCA services and SCA references as follows:

- Parses the `<binding.ws>` element and, for service bindings, generates a WSDL of the service to be published.
- For service bindings, publishes the "Plain Old Java Object" (POJO) as a Java API for XML-Based Web Services (JAX-WS) Web Service.
- Accepts requests for the published services and performs reference invocations to the Web Services.

An example of a Web Service binding on a service is shown in the following example:

```xml
<sca:service name="SCAService" type="com.oracle.test.SayHello" target="hello">
    <wlsb:binding.ws name="mysvcname" port="myportname" uri="/mysvcnameuri"/>
</sca:service>
```

The following example represents a Web Service binding on a reference:

```xml
<sca:reference name="SCAReference" type="com.oracle.test.SayHelloRef">
</sca:reference>
```
For reference bindings that refer to services outside the current application, it is expected that the contract class used for the reference is generated from the WSDL using client tools such as the WebLogic clientgen Ant task or Oracle JDeveloper.

⚠️ **Caution:**

If the contract class is not generated from JAX-WS compatible client tools, certain types of functionality may not work correctly.

The type that is specified in the `type` attribute of `sca:reference` must be a JAX-WS compatible interface, generated from the external WSDL using a JAX-WS compatible client generation tool such as the JAX-WS wsimport tool, the WebLogic clientgen Ant task, Oracle JDeveloper, or Oracle Enterprise Pack for Eclipse (OEPE).

### binding.ws Element Attributes

Attributes of the `<binding.ws>` element are as follows:

- **databinding**
  
  Optional. Specifies the type of databinding to use for converting SOAP messages to and from Java. Valid values are:
  
  - `toplink.jaxb` (default)
    
    Specifies that the databinding use TopLink/EclipseLink JAXB (JAXB2) from EclipseLink 2.0. For information about those technologies, see the EclipseLink developer guides at [http://www.eclipse.org/eclipselink/#documentation](http://www.eclipse.org/eclipselink/#documentation).
  
  - `toplink.sdo` - Toplink/EclipseLink SDO from EclipseLink 2.0
    
  
    Depending on the type of databinding, you may also have to use the `<property>` subelement of `binding.ws`.
    
    - See [About Configuring Custom Databinding](#)
    
    - See [Configuring TopLink/EclipseLink SDO Databinding](#)
    
    - See [Configuring Glassfish JAXB Databinding](#)
  
  - `glassfish.jaxb`
    
    Specifies that the databinding use the Java Architecture for XML Binding 2.x (JAXB2) Reference Implementation (JAXB RI). For information about the JAXB RI, see [Glassfish > Metro > JAXB](http://jaxb.java.net/).

- **location**
  
  Required for `sca:reference` bindings only. Specifies the location (that is, URL) where the external reference can be found. The WSDL must be made available by appending `?wsdl` to this location.
**name**

Optional. Specifies the name of the binding.

For *sca:service* bindings, this name is published as the service name in the WSDL. It overrides the name specified in `name` attribute of the *sca:service* element.

**port**

Optional for *sca:service* bindings. Specifies the port name to use for the service endpoint.

Required for *sca:reference* bindings. Specifies the WSDL port that this reference points to in the external Web Service.

This should be of the form `namespace uri#wsdl.endpoint (servicename/portname)`.

**soapversion**

Optional. Specifies the SOAP version of the Web Service. Valid values are 1.1 and 1.2. Defaults to 1.1.

**uri**

Required for *sca:service* bindings only. Specifies the location, relative to the context-root of the SCA application, where the Web Service must be published for this SCA service.

### binding.ws Subelements

The `<binding.ws>` element can have the following subelements:

- `PolicyReference` Element
- `property` Element

### PolicyReference Element

The `<PolicyReference>` subelement of `<binding.ws>` specifies a reference to the security policy to use. It has the following attributes:

**uri**

Specifies the location of the policy. Only built-in WebLogic Server security policies are supported in this release.

**direction**

Optional. Specifies whether the policy is inbound, outbound, or both. The default is both.

See Configuring Security for more information about using the PolicyReference element to configure security policies.
property Element

The <property> subelement of <binding.ws> has a name attribute that accepts the following property names:

- **weblogic.sca.binding.ws.sdoSchemaFile**
  Specifies the location of the schema file for SDO bindings.
  See Configuring TopLink/EclipseLink SDO Databinding.
- **weblogic.sca.binding.ws.externalCustomizationFile**
  Specifies an external databinding customizing XML file.
  See About Configuring Custom Databinding.
- **weblogic.sca.binding.ws.referenceWsdlCacheTimeoutMins**
  Configures WSDL caching when invoking references. To enable caching and to specify the caching timeout period, specify a positive number to indicate the number of minutes. To disable WSDL caching, specify zero or a negative value. If this property is not set, caching is enabled with a default timeout of 60 (minutes).

 Configuring Security

You need to use the <PolicyReference> element to configure security policies for Web Services bindings. See PolicyReference Element.

The following security mechanisms are supported:

- Username token with message protection (WS-Security 1.1)
- X509 certificate authentication with message protection (WS-Security 1.1)
- Anonymous with message protection (WS-Security 1.1)
- ID Propagation using SAML token (sender-vouches) with message protection (WS-Security 1.1)
- Username token over SSL
- SAML token (Sender Vouches) over SSL

 Security Configuration Examples

The following is an example of a security configuration for an SCA service:

```xml
<sca:service
    name="SCAServicePolicy"
    type="com.oracle.test.SayHelloRefImpl"
    target="hello2">
    <binding.ws
        xmlns="http://xmlns.oracle.com/weblogic/weblogic-sca-binding"
        name="mypolicysvcsname"
        port="uri:myns#wsdl.endpoint(mypolicysvc/mypolicyport)"
        url="/mypolicyuri"
    >
        <PolicyReference
            xmlns="http://schemas.xmlsoap.org/ws/2004/09/policy"
            target="mypolicyuri"
        />
    </binding.ws>
</sca:service>
```
If you want to specify that a SCA service or reference uses an Oracle Web Services Manager (OWSM) security policy instead of a WebLogic security policy, you need to set the optional `weblogic.sca.binding.ws.usingOwsmPolicies` property to `true`.

See Using Oracle Web Services Manager Security Policies in *Securing WebLogic Web Services for Oracle WebLogic Server*.

Security Configuration for SCA Service and SCA Reference Using OWSM Policy

**Example 8-1  Security Configuration for SCA Service Using an OWSM Policy**

The following is an example of security configuration for SCA service using an OWSM policy:

```
<sca:service name="SCAServicePolicy"
    type="com.oracle.test.SayHelloRefImpl"
    target="hello2">
    <binding.ws xmlns="http://xmlns.oracle.com/weblogic/weblogic-sca-binding"
        name="mypolicysvcname"
        port="uri:myns#wsdl.endpoint(mypolicysvc/mypolicyport)"
        uri="/mypolicyuri">
            uri="policy:oracle/wss_username_token_service_policy" />
        <property name="weblogic.sca.binding.ws.usingOwsmPolicies">true</property>
    </binding.ws>
</sca:service>
```

**Example 8-2  Security Configuration for SCA Reference Using an OWSM Policy**

The following is an example of security configuration for SCA reference using an OWSM policy:

```
<sca:reference name="SCAReferenceOwsmUNTPolicy"
    type="com.oracle.test.SimpleSayHello">
    <binding.ws xmlns="http://xmlns.oracle.com/weblogic/weblogic-sca-binding"
        location="http://adc2101232:7001/jscaowsm/myowsmuriunt12"
        port="http://test.oracle.com#wsdl.endpoint(myowsmsvcname/myowsmport)"
        soapVersion="1.2">"}
            url="policy:oracle/wss_username_token_client_policy" />
        <property name="weblogic.sca.binding.ws.usingOwsmPolicies">true</property>
    </binding.ws>
</sca:reference>
```
Configuring Databinding

You need to implement databinding by:

- Specifying the databinding type to use in the `databinding` attribute of `<binding.ws>`.
- Specifying additional details in the `<property>` subelement of `<binding.ws>` for SDO and customization files.

For information about configuring different kinds of databinding, see the following topics:

- Configuring TopLink/EclipseLink JAXB Binding.
- Configuring TopLink/EclipseLink SDO Databinding.
- Configuring Glassfish JAXB Databinding.
- About Configuring Custom Databinding.

If a databinding type is not specified, `toplink.jaxb` is used.

Configuring TopLink/EclipseLink JAXB Binding

To configure TopLink/EclipseLink JAXB databinding, enter `toplink.jaxb` as the value for the `databinding` attribute in the `<binding.ws>` element.

Note:

This is the default databinding type. If nothing is specified for the `databinding` attribute, `toplink.jaxb` is used.

The following example represents TopLink/EclipseLink JAXB databinding:

```xml
<sca:service
    name="SCAComplexService" type="com.oracle.test.ComplexHello"
    target="complexHello">
    <wlsb:binding.ws
        name="mycomplexsvc"
        port="mycomplexexport" uri="/mycomplexsvcuri"
        databinding="toplink.jaxb"/>
</sca:service>
```

Configuring TopLink/EclipseLink SDO Databinding

To configure databinding TopLink/EclipseLink SDO:

1. In the `<binding.ws>` element, enter `toplink.sdo` as the value for the `databinding` attribute.
2. In the `<property>` subelement of `<binding.ws>`:
a. Enter `weblogic.sca.binding.ws.sdoSchemaFile` as the value for the `name` attribute.

b. Enter the location of the schema file for the SDO bindings as the content of the element. The path must be relative to the application root, and the schema file must be bundled with the application.

The following example represents TopLink/EclipseLink SDO databinding:

```xml
<sca:service name="SCASDOService" type="com.oracle.test.sdo.HelloSDO" target="sdoHello">
  <wlsb:binding.ws name="mysdosvc" port="mysdoport" uri="/mysdosvcuri"
                   databinding="toplink.sdo">
    <property name="weblogic.sca.binding.ws.sdoSchemaFile">MySDO.xsd</property>
  </wlsb:binding.ws>
</sca:service>
```

**Configuring Glassfish JAXB Databinding**

If you need to configure Glassfish JAXB databinding, enter `glassfish.jaxb` as the value for the `databinding` attribute in the `binding.ws` element.

The following example represents GlassFish JAXB binding:

```xml
<sca:service name="SCAComplexService" type="com.oracle.test.ComplexHello" target="complexHello">
  <wlsb:binding.ws name="mycomplexsvc" port="mycomplexport" uri="/mycomplexsvcuri"
                   databinding="glassfish.jaxb"/>
</sca:service>
```

**About Configuring Custom Databinding**

You can provide an external databinding customization XML file that provides additional information on the Web Service binding. This file provides mapping metadata to define the attributes of a Java Web Service endpoint. The external customization file can be used to customize both WSDL and schema.

One example scenario of the use of the customization file is when the contract class for an SCA service contains overloaded methods that you want to expose as Web Service operations. The customization file can be used to disambiguate operation names for the Web Service.

Another example scenario (schema customization) is to change the name or the order of elements in a generated complex type.

**Configuring Custom Databinding**

To configure custom databinding:

1. Create the customization file in the same location as the implementation class file. The schema for the customization file, `weblogic-wsee-databinding.xsd`, must also be bundled with the application.
See WebLogic SCA Databindings Customization Descriptor Schema (weblogic-wsee-databinding.xsd) for the schema.

This schema defines three kinds of XML constructs:

a. Constructs that are analogous to those in JAX-WS and JSR 181 (Web Services Metadata for the Java Platform). These constructs (a) override or define attributes on the Service Endpoint Interface (SEI) and (b) override or specify information that would normally be part of Java Architecture for XML Binding (JAXB) annotations for the value types used in the interfaces of the SEI.

b. Additional mapping specifications not available using standard JAX-WS or JAXB annotations, primarily for use with the java.util.Collections API.

c. References to external JAXB mapping metadata from a Toplink Object-XML (OXM) file. This is only relevant for the toplink.jaxb databinding.

When a construct is the direct analog of a JAX-WS, JSR 181, or JAXB annotation, the schema includes a comment with notation such as:

Corresponding Java annotation javax.jws.WebParam.Mode

2. In the <property> subelement of <binding.ws>:

- Enter weblogic.sca.binding.ws.externalCustomizationFile as the value for the name attribute.
- Enter the name of the customization file as the content of the <property> element.

Custom Databinding Examples

The following example represents the <property> element that is configured to point to a customization file:

```xml
<sca:service name="SCAServiceOverloaded" type="com.oracle.test.SayHelloOverloaded" target="overloadedHello">
    <wlsb:binding.ws name="myoverloadedsvcname" port="myoverloadedportname" uri="/myoverloadedsvcnameuri">
        <property name="weblogic.sca.binding.ws.externalCustomizationFile">overloading_mapping_file.xml</property>
    </wlsb:binding.ws>
</sca:service>
```

A sample customization file is shown in the following example. It shows how an overloaded sayHello() method in the POJO is to be supported. The methods are mapped to two operations in the WSDL called sayHello and sayHelloWithString.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```
The customization file shown below in the following example illustrates the use of an inline `<toplink-oxm>` element to customize the order of elements in the generated schema. This example specifies that the schema generated for the `ShoppingCartItem` object includes the `quantity`, `price`, and `id` properties in the specified order instead of the default ordering.

```xml
<java-wsdl-mapping name="com.oracle.test.GetPriceRemote"
    xmlns="http://xmlns.oracle.com/weblogic/weblogic-wsee-databinding"
    xmlns:oxm="http://www.eclipse.org/eclipselink/xsds/persistence/oxm"
    databinding="toplink.jaxb">
  <xml-schema-mapping>
    <toplink-oxm java-package="com.oracle.test">
      <oxm:xml-bindings>
        <oxm:xml-schema/>
        <oxm:java-types>
          <oxm:java-type name="com.oracle.test.ShoppingCartItem">
            <oxm:xml-type prop-order="quantity price id"/>
          </oxm:java-type>
        </oxm:java-types>
      </oxm:xml-bindings>
    </toplink-oxm>
  </xml-schema-mapping>
</java-wsdl-mapping>
```

## Configuring Databinding for SOAP Attachments

Simple Object Access Protocol (SOAP) attachments are supported when using TopLink/EclipseLink JAXB databinding. These types are supported:

- SOAP Message Transmission Optimization Mechanism (MTOM)
- SOAP Messages with Attachments (SwA)

To configure databinding for SOAP attachments:

1. Specify TopLink/EclipseLink JAXB databinding by entering `toplink.jaxb` as the value for the `databinding` attribute in the `binding.ws` element.
2. Configure for MTOM or SwA, as described in the following topics:
   - See Configuring Attachments Using MTOM.
   - See Configuring Attachments Using SwA.

### Configuring Attachments Using MTOM

You can enable MTOM in either of the following ways:

1. **Put an `@MTOM` annotation on an SEI or contract class**, as shown in the following example. This enables MTOM for `base64Binary` types.

   ```java
   @MTOM
   public class SayHelloMtom {
   public HasArray modifyArray(String name, int b) {
     HasArray ha = new HasArray();
     ha.b = b;
     ha.arr = ("<?xml version="1.0"?>"+ name + "/z").getBytes();
     return ha;
   }
   }
   ```

2. **Include a `<mtom>` element as an immediate child of the `<java-wsdl-mapping>` element in the external mapping file**, as shown in the following example. The attributes for the `<mtom>` element are optional. The default for `enabled` is `true`, and the default for `threshold` is `0`.

   ```xml
   <java-wsdl-mapping name="com.hello.sei.MyServiceEndpointInterface"
   <web-service name="hello-ws" target-namespace="hello-ns"/>
   <mtom enabled="true" threshold="50"/>
   ...
   ```

The examples illustrated in Configuring Attachments Using MTOM: Examples use an external customization file to enable MTOM on an SCA POJO to be used as a Web Service endpoint.

### Configuring Attachments Using MTOM: Examples

#### Example 8-3  Spring Context

The following example shows the Spring context, with a reference to the external customization file `mtomCustomizationMapping.xml`.

```xml
<sca:service name="SCAMtomService" type="com.oracle.test.SayHelloMtom"
   target="helloMtom">
   <binding.ws
      xmlns="http://xmlns.oracle.com/weblogic/weblogic-sca-binding"
Example 8-4 External Customization File mtomCustomizationMapping.xml

The following example shows the external customization file mtomCustomizationMapping.xml.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<java-wsdl-mapping name="com.oracle.test.SayHelloMtom"
 xmlns="http://xmlns.oracle.com/weblogic/weblogic-wsee-databinding"
 xmlns:oxm="http://www.eclipse.org/eclipselink/xsds/persistence/oxm"
 databinding="toplink.jaxb">
<xml-schema-mapping>
<toplink-oxm java-package="com.oracle.test">
<xml-bindings xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/oxm">
<java-types>
<java-type name="com.oracle.test.HasArray">
<java-attributes>
<xml-element java-attribute="arr" xml-mime-type="text/xml" />
</java-attributes>
</java-type>
</java-types>
</xml-bindings>
</toplink-oxm>
</xml-schema-mapping>
<mtom threshold="2"/>
</java-wsdl-mapping>
```

Example 8-5 Java Classes

The following example shows the Java classes.

```java
//SayHelloMtom.java
public class SayHelloMtom {
    public HasArray modifyArray(String name, int b) {
        HasArray ha = new HasArray();
        ha.b = b;
        ha.arr = ("<?xml version='1.0' ?><z>" + name + "</z>").getBytes();
        return ha;
    }
}
```
Configuring Attachments Using SwA

The SwA attachment type SwaRef is supported in two ways:

1. Use the @XmlAttachmentRef annotation on a DataHandler type parameter and/or return type, as shown in the following example:

```java
@XmlAttachmentRef
dataHandler(DataHandler echoDataHandler(@XmlAttachmentRef DataHandler dh));
```

2. Use the xml-attachment-ref attribute in the external customization file, as shown in the following example.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<java-wsdl-mapping name="com.oracle.test.SayHelloSwa"
xmlns="http://xmlns.oracle.com/weblogic/weblogic-wsee-databinding"
xmlns:oxm="http://www.eclipse.org/eclipselink/xsds/persistence/oxm"
databinding="toplink.jaxb">
  <xml-schema-mapping>
    <toplink-oxm java-package="com.oracle.test">
      <xml-bindings xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/oxm">
        <java-types>
          <java-type name="com.oracle.test.HasDataHandler">
            <java-attributes>
              <xml-element java-attribute="data" xml-attachment-ref="true" />
            </java-attributes>
          </java-type>
        </java-types>
      </xml-bindings>
    </toplink-oxm>
  </xml-schema-mapping>
</java-wsdl-mapping>
```

For examples, see Configuring Attachments Using SwA: Examples.

Configuring Attachments Using SwA: Examples

Example 8-6 and Example 8-7 show SwA enabled on an SCA POJO to be used as a Web Service endpoint. An alternative to using a wrapper class like the HasDataHandler class is to use the DataHandler as a parameter to the Web Service directly. To do so, you must modify the customization file as shown in Example 8-8. The corresponding SayHelloSwa.java file is shown in Example 8-9.
Example 8-6  Spring Context with Reference to External Customization File

The following example shows the Spring context with reference to an external customization file.

```xml
<sca:service name="SCAMtomService" type="com.oracle.test.SayHelloSwa" target="helloSwa">
  <binding.ws
    xmlns="http://xmlns.oracle.com/weblogic/weblogic-sca-binding"
    name="mySwaSvc"
    port="mySwaPort"
    uri="/mySwaSvcUri"
    databinding="toplink.jaxb">
    <property
      name="weblogic.sca.binding.ws.externalCustomizationFile">swaCustomizationMapping.xml</property>
  </binding.ws>
</sca:service>
```

Example 8-7  Java Classes

This example demonstrates using Java Classes to enable SwA on an SCA POJO to be used as a Web Service endpoint.

```java
//SayHelloSwa.java
package com.oracle.test;

public class SayHelloSwa {
  public HasDataHandler echoDataHandler(HasDataHandler dh) {
    return dh;
  }
}

//HasDataHandler.java
package com.oracle.test;

import javax.activation.DataHandler;

public class HasDataHandler {
  public DataHandler data;
}
```

Example 8-8  Customization File Using a DataHandler as a Parameter to a Web Service

This example shows how to use DataHandler as a parameter to a Web Service. This is an alternative to using a wrapper class like the HasDataHandler class.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<java-wsdl-mapping name="com.oracle.test.SayHelloSwa"
  xmlns="http://xmlns.oracle.com/weblogic/weblogic-wsee-databinding"
Example 8-9 Java Class

This example shows how to use the Java file, SayHelloSwa.java, corresponding to the DataHandler parameter used in the above example.

//SayHelloSwa.java
package com.oracle.test;

import javax.activation.DataHandler;
import javax.mail.util.ByteArrayDataSource;

public class SayHelloSwa {
    public DataHandler echoDataHandler(DataHandler dh) {
        byte[] b = new byte[1024];
        try {
            int len = dh.getInputStream().read(b);
            String resp = new String(b, 0, len) + "</swarefresponse>";
            return new DataHandler(new
            ByteArrayDataSource(resp.getBytes(), "text/xml"));
        } catch (Exception e) {
            String err = "<err>An error occurred: " + e.getClass().getName() + " - " + e.getMessage() + "</err>";
            return new DataHandler(new
            ByteArrayDataSource(err.getBytes(), "text/xml"));
        }
    }
}

Configuring Collection and Map Objects

Java Collection and Map objects are supported when using TopLink/EclipseLink JAXB databinding.
Note:

Built in Java subclasses of `java.util.List` and `java.util.Map` are supported. Multi-dimensional support (that is, Array of Arrays, Array of HashMaps, List of Lists, etc) will be supported in a future release.

You can express the generics type argument of a `Collection` (or members of the `Collection` family) with either an annotation or an external mapping file. For example, the two methods in the interface that is shown in Example 8-10 are equivalent in their input types if the `processItems2` method uses the external mapping file that is shown in Example 8-11.

**Example 8-10  Methods in an Interface (1)**

The following example shows two methods in an interface.

```java
public interface CollectionProcessor {
    public String processItems1(Collection<ItemType> items);
    public String processItems2(Collection items);
}
```

**Example 8-11  External Mapping File for Configuring Collection Objects [1]**

The following example shows the fragment of the external mapping file for the `processItems2` method shown in

```xml
<java-method name="processItems2">
    <java-params>
        <java-param>
            <oxm:xml-element type="mypackage.ItemType"/>
        </java-param>
    </java-params>
</java-method>
```

**Example 8-12  External Mapping File for Configuring Collection Objects [2]**

The following is another example of an external mapping file for configuring `Collection` objects.

```xml
<java-method name="testListOfCustomer">
    <java-params>
        <java-param>
            <oxm:xml-element type="mypackage.Customer"/>
        </java-param>
    </java-params>
</java-method>
```

**Externalizing Generic Type for Map**

Currently, externalizing the generic types is not directly supported for `java.util.Map` types.
However, if it is not possible to specify the generic type for the Map directly in the Java class, it can be indirectly supported by using the Java XML type adapter feature of the tolink.jaxb binding. This involves writing custom serializers for the desired generic Map types and specifying the custom serializers in the external mapping file, as shown in.

**Example 8-13    POJO to be Exposed as a Web Service (Without Generic Types on the Map)**

The following example demonstrates POJO that can be exposed as a Web Service.

```java
//Implementation class
public class CollectionMapExtTypeArgImpl {
    public Map testMapOfCustomAdapters(Map map) {
        //implementation goes here
    }
}
```

**Example 8-14    Custom Adapter Classes**

The following example specifies the custom adapter classes.

```java
//Custom Adapter class for Map<String, Integer> (JAXB Xml Adapter)
public class MapStringIntegerAdapter extends XmlAdapter<MapStringInteger, HashMap> {
    public HashMap unmarshal(MapStringInteger m) throws Exception {
        HashMap map = new HashMap();
        for (StringIntegerEntry e: m.entry) map.put(e.key, e.value);
        return map;
    }
    public MapStringInteger marshal(HashMap m) throws Exception {
        MapStringInteger map = new MapStringInteger();
        map.entry = new ArrayList<StringIntegerEntry>();
        for (Object k: m.keySet()) {
            StringIntegerEntry e = new StringIntegerEntry();
            e.key = (String) k;
            e.value = (Integer) m.get(k);
            map.entry.add(e);
        }
        return map;
    }
}
```

```java
//MapStringInteger.java
public class MapStringInteger {
    public static class StringIntegerEntry {
        @XmlAttribute
        public String key;
        @XmlValue
        public Integer value;
    }
    public List<StringIntegerEntry> entry;
}
```

//Similar implementation for MapStringCustomerAdapter would be needed
Example 8-15  Customization file fragment showing return type of Map<String, Integer> and parameter type of Map<String, Customer>

The following example demonstrates the customization file fragment showing return type of Map<String, Integer> and parameter type of Map<String, Customer>.

```java
<java-method name="testMapOfCustomAdapters">
    <oxm:xml-element xmlns="http://www.eclipse.org/eclipselink/xsds/persistence">
        <oxm:xml-java-type-adapter value="com.oracle.test.MapStringIntegerAdapter"/>
    </oxm:xml-element>
    <java-params>
        <java-param>
            <oxm:xml-element xmlns="http://www.eclipse.org/eclipselink/xsds/persistence">
                <oxm:xml-java-type-adapter value="com.oracle.test.MapStringCustomerAdapter"/>
            </oxm:xml-element>
        </java-param>
    </java-params>
</java-method>
```

### Precedence of Configuration Settings

You can specify several configuration settings in more than one place, including databinding mode, SOAP version, target namespace, service name, and port name. If one of those settings is specified in multiple places, they are evaluated in the following order of precedence:

1. If a configuration option is specified (and is not empty) in the Spring context in the `binding.ws` configuration, that setting is used. This setting takes precedence over all others.
2. If a configuration option is not specified in the Spring context in the `binding.ws` configuration, the value in the external customization file is used. This setting takes precedence over the next setting mentioned below.
3. If a configuration option is not specified in the Spring context or the external customization file but is specified in annotations on the POJO, the annotations on the POJO are used.

### Deployment

At deployment, WebLogic SCA Runtime:

1. Uses the service interface information to generate a WSDL.
2. Publishes the service at the specified URI (relative to the context root) as a JAX-WS endpoint.

### Runtime

The runtime behavior of Web Service bindings is described as follows:
• **Service Requests**— Requests to services are handled in the following manner:
  1. The appropriate data binding converts the incoming SOAP message payload into Java objects.
  2. Any policies are handled by the JAX-WS runtime.
  3. The POJO implementing the service is invoked by the WebLogic SCA Runtime.

• **Reference Invocations**— WebLogic SCA Runtime recognizes that a reference invocation is for an external reference. It acts as a JAX-WS client to the external Web Service, unmarshals the result to Java and returns it in a manner transparent to the calling Java code.
Deploying WebLogic Spring SCA Applications

This chapter describes how to deploy WebLogic Spring SCA applications to Oracle WebLogic Server.

- Preparing Deployment Units
- Configuring the Deployment Descriptor
- Bundling Libraries
- Deploying in a Cluster
- Deploying WebLogic Spring SCA Applications Using Other Tools
- Runtime

Note:

Support for using WebLogic Spring SCA applications as components in SCA composites is available as a Technical Preview in the current release of Oracle SOA Suite. However, documentation for that feature is not yet available in Oracle SOA Suite.

Preparing Deployment Units

WebLogic SCA Runtime supports Enterprise Archive (EAR) and Web Archive (WAR) formats as deployment units.

A typical WebLogic SCA application bundles the Spring application context file in the META-INF/jsca/ directory, along with the application's POJO classes.

The organization of a sample WebLogic Spring SCA application EAR archive is shown in the following example:

META-INF/application.xml
META-INF/weblogic-application.xml
META-INF/jsca/spring-context.xml
MyEJBApp.jar
MyWebApp.war
APP-INF/lib/MyScaClasses.jar
META-INF/jsca/spring-context.xml
APP-INF/classes/MyScaPojo.class

The organization of a sample WebLogic Spring SCA application WAR archive is shown in the following example:

WEB-INF/web.xml
WEB-INF/weblogic.xml
WEB-INF/lib/MyScaClasses.jar
Configuring the Deployment Descriptor

You must add a `<library-ref>` element to the deployment descriptor for a application.

When referring to a shared Web application library, you can also add an optional `<context-root>` element to declare the context root of the library module. This ensures that the context root is set to the configured value and will not conflict with other WebLogic SCA applications deployed on the same server and referring to the same shared library.

**Note:**

Oracle recommends that you provide a unique `<context-root>` for each WebLogic Spring SCA application. When multiple applications are deployed into a single server or a cluster, each application has its own library reference to `weblogic-sca-1.0.war`. Not specifying different context roots for each application results in a context root conflict because every WebLogic SCA application uses the default context root specified in `weblogic-sca-1.0.war`.

The following example represents a `<library-ref>` and `<context-root>` in an EAR's `weblogic-application.xml` descriptor:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<weblogic-application xmlns="http://www.bea.com/ns/weblogic/90">
  <library-ref>
    <library-name>weblogic-sca</library-name>
    <context-root>weblogic-sca-ctx-root1</context-root>
  </library-ref>
</weblogic-application>
```

The following example represents a `<library-ref>` and `<context-root>` in a WAR's `weblogic.xml` descriptor.

```xml
<?xml version="1.0" encoding="UTF-859-1"?>
<weblogic-application xmlns="http://www.bea.com/ns/weblogic/90">
  <library-ref>
    <library-name>weblogic-sca</library-name>
    <context-root>weblogic-sca-ctx-root1</context-root>
  </library-ref>
</weblogic-application>
```

**Bundling Libraries**

WebLogic Spring SCA applications must bundle `spring.jar` under `APP-INF/lib` for EARs and under `WEB-INF/lib` for WARs. Spring has a dependency on `commons-logging`. Therefore the jar must also be bundled with the application.

WebLogic SCA supports Spring 2.0.6 or later. Spring 2.5.3 is recommended.
Deploying in a Cluster

A WebLogic Spring SCA application can be deployed in a homogenous clustered environment. Cluster deployment deploys the application to every node, and since most request processing is stateless (stateful ones use the database for storing their state), a load balancer or a plug-in can route the request to any node.

Deploying WebLogic Spring SCA Applications Using Other Tools

You can also deploy WebLogic Spring SCA applications in standalone WebLogic Server using Oracle Enterprise Pack for Eclipse (OEPE) and Oracle JDeveloper. See Tools Support for more information about WebLogic SCA support in those tools.

Runtime

When a WebLogic Spring SCA application is deployed, its services are exposed by the appropriate binding component implementations. For a service defined with binding.ws, the Web Service binding component implementation publishes a Web Services endpoint. For a service defined with binding.ejb, the EJB binding component makes the EJB available in JNDI.

If multiple Spring application context files are used during a deploy within the same application, WebLogic SCA Runtime wires the matching services and references, ignoring the bindings for internal wiring. This requires use of the optional default attribute in an sca:reference element. See sca:reference Element.

Note:

A WebLogic Spring SCA application with two or more Spring application context files can be nested into other composites only in Oracle SOA, since this requires an enclosing SCDL file.
Viewing WebLogic SCA Application Configurations

This chapter describes how to view the configuration details of a WebLogic Spring SCA application in the Oracle WebLogic Server Administration Console. These configuration details can be viewed in the Administration Console after the WebLogic SCA console extension has been enabled.

- Prerequisites for Viewing Application Configurations
- Enabling the WebLogic Server Administration Console Extension
- Viewing Details about Configured Services and References

Prerequisites for Viewing Application Configurations

You must complete the following tasks before you can view an application’s configuration in the console extension:

- Deploy the WebLogic SCA Runtime module if it is not already deployed.
  See Deploying WebLogic SCA Runtime to WebLogic Server.
- Enable the weblogic-sca-console extension.
  See Enabling the WebLogic Server Administration Console Extension.
- Deploy the application, if it is not already deployed.
  See Deploying WebLogic Spring SCA Applications.

Enabling the WebLogic Server Administration Console Extension

You must enable the Administration Console extension before you can view details about configuration services and references.

To enable the extension, perform the following steps:

1. Start the WebLogic Server Administration Console.
2. In the banner toolbar region at the top of the right pane of the Console, click Preferences.
3. Click the Extensions tab.
4. Select the checkbox next to weblogic-sca-console.
5. Click Enable.
6. Restart the Administration Server.
To view details about configured services and references, perform the following steps:

1. In the left pane of the Console, select **Deployments**.

2. In the Deployments table, click the name of the WebLogic Spring SCA application whose configuration you want to view.

3. On the Settings for application_name page, click the WebLogic SCA tab.

4. In the WebLogic SCA Artifacts table, click the name of the service or reference you want to view.

   For explanations of the fields, click **Help** in the banner toolbar which is a region at the top of the Console.
WebLogic SCA Schemas

This chapter describes the WebLogic SCA schemas in Oracle WebLogic Server.

- WebLogic Spring SCA Schema (weblogic-sca.xsd)
- WebLogic SCA Binding Schema (weblogic-sca-binding.xsd)
- Web Service Policy Schema (ws-policy.xsd)
- WebLogic SCA Databindings Customization Descriptor Schema (weblogic-wsee-databinding.xsd)

WebLogic Spring SCA Schema (weblogic-sca.xsd)

WebLogic SCA provides extensions to the OASIS Spring SCA schema to support the sca:reference and sca:service elements used in Spring application contexts to configure WebLogic SCA applications. The schema is available at http://www.oracle.com/technology/weblogic/weblogic-sca/1.0/weblogic-sca.xsd.

See Configuring the Spring Application Context for information about how this schema is used.

Note:


WebLogic SCA Binding Schema (weblogic-sca-binding.xsd)

The binding.ejb element and binding.ws elements are defined in the weblogic-sca-binding.xsd schema. The schema is available at http://xmlns.oracle.com/weblogic/weblogic-sca-binding/1.0/weblogic-sca-binding.xsd.

See Configuring EJB Session Bean Bindings for information on how the binding.ejb element is used to configure EJB session bean bindings.

See Configuring Web Service Bindings for information on how the binding.ws element is used to configure Web Service bindings.

Web Service Policy Schema (ws-policy.xsd)

Policies for Web Service bindings are based on the ws-policy.xsd schema, as shown in the following example.

See Configuring Security for information on how this schema is used.
WebLogic SCA Databindings Customization Descriptor Schema (weblogic-wsee-databinding.xsd)

The schema for providing an external databinding customization XML file is available at http://xmlns.oracle.com/weblogic/weblogic-wsee-databinding/1.2/weblogic-wsee-databinding.xsd.

The customization file provides mapping metadata for databinding. The data is used to define the attributes of a Java Web Service endpoint. See About Configuring Custom Databinding.