Oracle® Fusion Middleware

User's Guide for the Oracle Java CAPS Migration Tool

11*g* Release 1 (11.1.1.6.0)

E24884-01

November 2011

Documentation for developers that describes how to use the automated migration tool to migrate existing Oracle Java CAPS Repository projects with Java Collaboration Definitions to Oracle SOA Suite projects with Spring components, and to migrate Repository and JBI projects with BPEL business processes to Oracle SOA Suite BPEL projects.

ORACLE

Oracle Fusion Middleware User's Guide for the Oracle Java CAPS Migration Tool, 11*g* Release 1 (11.1.1.6.0)

E24884-01

Primary Author: C. Thom

Contributing Author:

Contributor: Oracle SOA Suite and Oracle Java CAPS development, product management, and quality assurance teams

Contents

Preface v

Audience v
Documentation Accessibility v
Related Documents vi
Conventions vi

1 Introduction to the Oracle Java CAPS Migration Tool

1.1 Oracle Java CAPS Common Abbreviations 1-1
1.2 Overview of the Migration Process 1-1
1.2.1 How Projects are Migrated 1-2
1.2.1.1 Repository Project Migration 1-2
1.2.1.2 JBI Project Migration 1-3
1.2.1.3 Oracle Java CAPS Repository Mapping to Oracle SOA Suite 1-3
1.2.1.4 Migration Tool Process Flow 1-3
1.2.2 Support for Repository Project Migration 1-4
1.2.2.1 Support for Java Collaboration Definition Migration 1-6
1.2.2.2 Support for Adapters and OTDs 1-6
1.2.2.3 Support for Oracle Java CAPS Framework Classes 1-8
1.2.3 OTD to XSD and XSD to JAXB Conversion 1-9
1.2.4 JAXB Generation During Migration 1-9
1.2.5 Support for JBI Project Migration 1-10
1.2.5.1 WS-I Version 1 Compliance 1-10
1.2.5.2 Same Target Namespace for Different Message Definitions 1-10
1.2.5.3 System Properties 1-11
1.2.5.4 BPEL 2.0 Constructs 1-11
1.3 About the Migrated Oracle SOA Suite Projects 1-11
1.3.1 About the Conversion to Spring 1-11
1.4 Migration Considerations 1-13
1.4.1 Deciding Whether to Migrate 1-13
1.4.2 Deciding How to Migrate 1-14
1.5 Limitations of the Migration Tool 1-14

2 Installing the Oracle Java CAPS Migration Tool

2.1 About the Installation 2-1
2.1.1 Prerequisites 2-1

2.2 Installing the Migration Tool 2-1

3 Migrating the Projects

3.1 Overview of the Migration Tool Process 3-1
3.2 Migrating Oracle Java CAPS Projects 3-1
3.2.1 Before you Begin 3-1
3.2.1.1 Verify JBI Projects for Compliance 3-2
3.2.1.2 Modify Business Processes 3-2
3.2.1.3 Modify Projects with File Write Operations 3-2
3.2.1.4 Rebuild and Redeploy the Projects to Migrate 3-2
3.2.2 Setting the Migration Logging Properties 3-2
3.2.3 Migrating a Project Using the Wizard 3-4
3.2.4 Migrating a Project Using the Command Line 3-7
3.2.4.1 Migration Tool Usage 3-7
3.3 Converting OTD to XSD Format 3-8
3.4 Converting XSD to JAXB Format 3-9

4 Post-Migration Tasks

4.1 Opening a Migrated Project in Oracle JDeveloper 4-1
4.2 Configuring Migrated JBI Projects 4-2
4.2.1 Configuring Migrated Binding Components 4-2
4.2.1.1 Changes for JMS Adapters 4-3
4.2.2 Adding Service Elements 4-3
4.2.3 Configuring Quality of Service Properties 4-4
4.2.4 Verifying the BPEL Structure 4-4
4.3 Configuring Migrated Adapters and OTDs 4-4
4.3.1 Enabling File or JMS Message Types as Opaque 4-5
4.3.2 Configuring a Project with the Same XSD and FCX OTDs for Inbound and Outbound .. 4-6
4.3.3 Configuring FCX OTDs 4-7
4.3.4 Adding Adapters not Converted by the Migration Tool 4-7
4.4 Configuring Converted Oracle SOA Suite Spring Components 4-8
4.4.1 Modifying the Spring Bean Java Class 4-9
4.4.2 Converting a Byte Array Input to String 4-9
4.4.3 Configuring the Spring Bean Class for File or JMS Outbound Adapters 4-10
4.4.4 Accessing JMS Header Properties 4-11
4.4.5 Configuring Sub-Collaborations Called from Java Collaboration Definitions 4-11
4.5 Configuring Business Processes 4-12
4.5.1 Migrating User Activities in Business Processes 4-12
4.5.2 Migrating Correlation Initialization in Marshal and Unmarshal Activities 4-12
4.6 Adding JAR Files to a Migrated Project 4-13
4.7 Creating JMS Resources 4-15
A.1 Sample Code for Migrating a Stand-Alone JCD A-1
A.2 Sample Code for Migrating a JCD Called from a Business Process A-4

Glossary

Preface

Oracle Fusion Middleware User's Guide for the Oracle Java CAPS Migration Tool describes how to migrate Repository and JBI projects from Oracle Java CAPS to Oracle SOA Suite.

Audience

This document is intended for developers responsible for migrating Oracle SOA Suite projects to Oracle SOA Suite projects.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users who are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so our documentation can be accessible to all of our customers. For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/accessibility`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/support/contact.html` or visit `http://www.oracle.com/accessibility/support.html` if you are hearing impaired.

Related Documents

For more information, see the following Oracle resources:

- *Oracle Java CAPS 6.3 Documentation Library* (http://download.oracle.com/docs/cd/E21454_01/index.html)
- Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1

Introduction to the Oracle Java CAPS Migration Tool

This chapter provides an overview of the process of migrating Oracle Java CAPS Repository and JBI projects to Oracle SOA Suite using the Oracle Java CAPS Migration Tool.

This chapter includes the following topics:

- Section 1.1, "Oracle Java CAPS Common Abbreviations"
- Section 1.2, "Overview of the Migration Process"
- Section 1.3, "About the Migrated Oracle SOA Suite Projects"
- Section 1.4, "Migration Considerations"
- Section 1.5, "Limitations of the Migration Tool"

1.1 Oracle Java CAPS Common Abbreviations

This section lists abbreviations and acronyms that are commonly used for Oracle Java CAPS components and the technologies used by Oracle Java CAPS. These are used throughout this document. You can find definitions for these components in the Glossary.

- JCD: Java Collaboration Definition
- OTD: Object Type Definition
- FCX OTD: XMLBean OTD, also called a First Class OTD
- UD OTD: User-defined OTD
- JBI: Java Business Integration
- JAXB: Java Architecture for XML Binding
- JAX-WS: Java API for XML Web Services
- DOM: Document Object Model

1.2 Overview of the Migration Process

The Oracle Java CAPS Migration Tool automates much of the transformation required to migrate existing Oracle Java CAPS Repository and JBI projects to Oracle SOA Suite projects, preserving mappings and business logic in the converted projects. The migration tool starts with a deployed Oracle Java CAPS JBI composite application (ZIP file) or a Repository project's archive (EAR file). The migration tool reads the input ZIP

or EAR file, and then generates the required project components for Oracle SOA Suite in a file structure compatible with Oracle JDeveloper.

Oracle Java CAPS Repository project components, such as BPEL 1.0 business processes, XSD files, WSDL files, messageable OTDs, and Java Collaboration Definitions (JCDs), are parsed and transformed to be compatible with Oracle SOA Suite components. JBI project components, such as BPEL 2.0 business processes, WSDL files, and XSD files, are transferred to an Oracle SOA Suite project structure with fewer changes since they are more compatible with Oracle SOA Suite.

The output of the Oracle Java CAPS Migration Tool is an Oracle SOA Suite project that can be opened, viewed, and edited in Oracle JDeveloper. The migrated project also includes Oracle Java CAPS JAR files on which the project depends. You can use the migration tool to migrate the following Oracle Java CAPS versions: 5.0.5, any update release; 5.1.3 any update release; and Release 6 or later.

1.2.1 How Projects are Migrated

Migrating Repository projects is a more complex process than migrating JBI projects, which have a more direct mapping to Oracle SOA Suite projects. In either case, the migration tool starts with the generated deployment file. For Repository projects, this is an EAR file; for JBI projects it is the build ZIP file. The migration tool does not perform any validation on the input project, which is assumed to be valid since it has been successfully built and deployed in Oracle Java CAPS.

1.2.1.1 Repository Project Migration

When you migrate a Repository project that contains a business process, the BPEL 1.0 code is transformed to BPEL 2.0 in the Oracle SOA Suite project. Java Collaboration Definitions (JCDs) are converted to Spring components and the associated Spring context files are generated. The required and available JAR files from the Oracle Java CAPS project are copied to the Oracle SOA Suite project. JCDs that are exposed as web services are converted to a Spring context exposed as a service in the migrated project. Projects with business processes that call JCDs are converted to business processes that call Spring components.

The migration tool can migrate File, JMS, and web service adapters, but does not migrate the code for other types of adapters. When you migrate a project that uses specialized adapters (such as database adapters), the migration tool generates placeholder endpoints in the migration project, which are just basic WSDL interfaces. To complete the conversion of these projects, you need to add the corresponding Oracle SOA Suite adapter to the migrated project in Oracle JDeveloper and map it to the corresponding Spring component or business process. For more information, see Section 4.3.4, "Adding Adapters not Converted by the Migration Tool."

Most marshal and unmarshal operations in business processes are converted into Java Embedding activities in the business process in the migrated project, including XSD, DTD, User-Defined, and SWIFT OTD marshal and unmarshal operations. In addition, all JCDs that are converted into Spring components can marshal and unmarshal the generated JAXB objects using the helper methods from the `OTDUtil` class.

When you migrate a Repository project, the folder structure for the new Oracle SOA Suite project is created in a directory you specify. This folder includes all the required Oracle SOA Suite files, such as BPEL files, WSDL documents, the composite file, the project file, and so on. The migration tool also creates a temporary folder in the migration tool directory to store the original code from the migrated Oracle Java CAPS project. This allows you to compare the common files between the Oracle Java CAPS

and Oracle SOA Suite projects, compare the JCD source code with the migrated code, compare the BPEL files, and so on.

1.2.1.2 JBI Project Migration

The mapping between project components for Oracle SOA Suite and Oracle Java CAPS JBI is more direct and requires fewer transformations. Instead of using OTDs, as in Oracle Java CAPS Repository projects, both JBI and Oracle SOA Suite projects convert native message formatting to XML and back again. When you migrate a JBI project, the migration tool extracts the information from the composite application's build ZIP file. It retrieves the endpoint and service connection information from the `jbi.xml` file, and then generates the BPEL file, associated WSDL and JCA files, the Oracle JDeveloper `composite.xml` file, and the project JPR file.

> **Note:** In the migrated Oracle SOA Suite project, the concrete binding information and service elements are moved from the WSDL files to the composite and binding configuration (JCA) files. In Oracle SOA Suite, WSDL files are abstract, and binding type information is specified directly in the `composite.xml` and adapter JCA files.

Not all Oracle Java CAPS JBI projects types can be converted using the migration tool. The projects that are most directly converted use business processes with the File, JMS, or HTTP binding components. Specialized service engines (such as Worklist Manager and Data Mashup) and binding components (such as Scheduler and HL7) are not migrated.

1.2.1.3 Oracle Java CAPS Repository Mapping to Oracle SOA Suite

Table 1–1 lists each Oracle Java CAPS Repository component along with the corresponding Oracle SOA Suite component to which it is migrated. As mentioned earlier, Oracle Java CAPS JBI projects have a more direct one-to-one mapping with Oracle SOA Suite projects.

Table 1–1 Oracle Java CAPS Repository Component Mapping to Oracle SOA Suite

This Oracle Java CAPS Component	Maps to This Oracle SOA Suite Component
Canonical data: OTD	Canonical data: XML Object
Canonical interface: OTD, WSDL	Canonical interface: WSDL
Proprietary	Standard
JCD	Spring Context
Connectivity Map	Composite
Business Process Invoking JCD	Business Process Invoking Spring Context
JCD as a Service	Spring Context as a Service

1.2.1.4 Migration Tool Process Flow

1. The migration tool runs in one of two modes: command-line or wizard mode. The input is the Oracle Java CAPS archive file (either a ZIP or EAR file), the output directory, and the Oracle SOA Suite project name for the migrated project.
2. The migration tool extracts the contents of the input file into a temporary location and selects the required files to start the migration process.

3. The migration tool derives the project's endpoints, which determine the wires in the Oracle SOA Suite composite.
4. The migration tool invokes a series of parsers:
   - The BPEL parser retrieves partner link definitions and, for Repository projects, transforms the BPEL 1.0 code to BPEL 2.0.
   - The WSDL parser maps the partner link defined in the WSDL document to the converted BPEL. It also converts the message types so they are compatible with WS-I Basic Profile.
   - The JCD parser parses the JCD Java code to derive the inbound and outbound. It adds the JAX-WS proxy method and all the accessor methods for the Logger, CollaborationContext, and TypeConverter.
   - The Connectivity parser parses the endpoints and builds the wire information in the Oracle SOA Suite composite.
   - The Descriptor parser reads the Oracle Java CAPS endpoints properties file for File and JMS adapters, and creates the required JCA files for the Oracle SOA Suite project. The parser reads through the project's `ra.xml` and `ejb.xml` files to derive the endpoint properties and create the metadata needed to generate the JCA files.
5. The migration tool generates the following project components:
   - Java Embedding activity in migrated business process for each of the marshal/unmarshal processes for messageable OTDs.
   - JAX-WS interface for JCDs invoked by a business process and JCDs exposed as web services.
   - XSD files for the messageable OTDs in the project.
   - Spring contexts and component types for each JCD.
   - Oracle SOA Suite `composite.xml` file.
   - JCA files.
   - Oracle SOA Suite project file (.jpr), which is required to open the migrated project in Oracle JDeveloper.
6. The migration tool extracts the required Oracle Java CAPS libraries from the source file and copies them to the `/SCA-INF/lib` directory of the migrated project.

1.2.2 Support for Repository Project Migration

For Repository projects, both business process and JCD projects can be migrated. However, there are cases where Oracle SOA Suite does not provide an exact correspondence to the adapters is use. The migration tool can migrate certain adapters and OTDs fully, but others require manual adjustments after the migration tool completes its conversion.

Business Processes and Java Collaboration Definitions

The migration tool converts projects that include standalone business processes or JCDs, or that include combinations of business processes and JCDs, such as calling sub-processes or sub-collaborations. For example, the following are supported:

- A business process (BPEL 1.0) invokes a sub-process or a sub-collaboration.

- A JCD invokes a sub-collaboration (this requires manual changes to reconnect the JCDs).
- A JCD is exposed as a web service. Any web service outbound in the JCD is not migrated.

File, JMS, and Web Service OTDs

The migration tool supports the following types of File, JMS, and Web Service OTD projects:

- Repository projects that include business processes and File, JMS, or web services endpoints (both inbound and outbound).
- Repository projects that include a business process with File or JMS adapters as inbound and other adapters, such as Oracle or SAP, as outbound. The outbound adapters are not be automatically migrated, but placeholder endpoints are generated for the migrated project.
- Repository projects that include a Java Collaboration Definition (JCD) with inbound and outbound File or JMS adapters.
- Repository projects that include a JCD with File or JMS adapters as inbound and other adapters, such as Oracle or SAP, as outbound. The outbound adapters are not be automatically migrated, but you can create and wire a corresponding Oracle SOA Suite adapter in the migration project.

XSD, DTD, and UD OTDs

The migration tool supports migrating Repository projects with marshal and unmarshal operations in XSD, DTD, and User-Defined OTDs.

First Class (FCX) OTDs

The migration tool supports the following types of FCX OTD projects:

- Repository projects with FCX OTDs as inbound and outbound in the JCD.
- Repository projects that include a JCD exposed as a web service with FCX OTDs as inbound and outbound.
- Repository projects that include a business process that invokes a JCD with FCX OTDs as inbound and outbound.

HL7 OTDs

The migration tool supports the following types of HL7 OTD projects:

- Repository projects that include marshal and unmarshal operations in an HL7 OTD message in a business process. Note that data mappings for HL7 OTD message are preserved in the converted project.
- Repository projects that include a business process that invokes a JCD with HL7 as both the input and output OTD message.

SWIFT OTDs

Oracle Java CAPS Repository projects with SWIFT OTDs in business processes and JCDs are automatically migrated by the migration tool. Some of the supported models with the SWIFT OTDs include:

- Repository projects that include a business process with marshal and unmarshal activities from a SWIFT OTD.

- A standalone JCD with File or JMS inbound and outbound and SWIFT OTD as the other OTD of the JCD. The File or JMS content is unmarshaled into the SWIFT OTD.

1.2.2.1 Support for Java Collaboration Definition Migration

The migration tool converts Oracle Java CAPS projects that contain Java Collaboration Definitions (JCDs) implemented in the ways described in Section 1.2.2, "Support for Repository Project Migration." When a JCD is migrated, the migration tool reads and parses the JCD source code from the EAR file. Based on the type of JCD implementation, the JCD parser modifies the original source code.

For standalone JCDs that use either the File Adapter or JMS adapter for the inbound or outbound connection, the parser identifies the inbound connection type and modifies the JCD to implement the corresponding Oracle SOA Suite adapter interface. The parser creates a method for the interface using pseudocode, in which a call to the original JCD method is made.

For JCDs that are called from a business process and for standalone JCDs that use File or JMS adapters for the inbound connection but not the outbound connection, the parser modifies the JCD to implement a JAX-WS interface, which is generated by the Oracle SOA Suite JAX-WS tool. The original JCD input and output OTD types are converted to JAXB. The generated JAXB objects are represented by several Java classes.

All Spring Beans created during the migration from JCDs can marshal and unmarshal the generated JAXB objects using the helper methods from the `OTDUtil` class.

For additional information and sample code for converted JCDs, see Appendix A, "Examples of Java Collaboration Definition Conversions."

1.2.2.2 Support for Adapters and OTDs

The Oracle Java CAPS Migration tool supports the migration of the following Object Type Definitions (OTDs): User-defined, XSD, DTD, SWIFT, and HL7. The migration tool converts all OTDs used in a Oracle Java CAPS project into XSD files, and provides utility methods to convert the XML data (JAXB/DOM defined by the XSD) to OTD and the OTD data back to XML data. The migration tool's OTD parser reads OTD metadata from the OTD JAR files in the input EAR file. It then creates an XSD file that represents the OTD and places the XSD file in the migrated Oracle SOA Suite project's `xsd` folder.

The migration tool names the XSD file based on the name of the OTD from which it was generated. The generated WSDL documents are modified to use these XSDs to define the message type so any business processes using the OTDs can resolve OTD message types.

The generated XSD files use the OTD field names for the element names unless the element name would be invalid. BPEL XPath expressions that use OTD types are transformed to match the generated XSD file. You can use the provided `OTDUtil` methods to convert data between OTD and XML data. The migration tool includes the `OTDUtil` class, and packages it in the migrated Oracle SOA Suite project. The `OTDUtil` class provides utility methods to perform native OTD operations by invoking the OTD operations on the OTD implementation and convert data back to XML defined by generated XSD. For more information, see the `migration.caps.runtime.OTDUtil` javadocs, which are packaged in the `caps.migration.runtime.util.jar` file packaged with the migration tool (in *`migration_tool_home`*`/oracle.migrationtool.jcaps.libraries/lib`). This JAR file is also included with the migrated Oracle SOA Suite project.

1.2.2.2.1 Migrating Projects with JMS Adapter receiveWait Operations

The Java CAPS JMS Adapter supports `receiveWait` web service operations, which have no equivalent in Oracle SOA Suite. Projects that include an outbound JMS Adapter with `receiveWait` and `send` operations can be migrated to Oracle SOA Suite and deployed successfully in Oracle JDeveloper. The Oracle Java CAPS Migration Tool removes the receiveWait operation for outbound JMS Adapters.

1.2.2.2.2 Migrating Projects with HL7 OTDs

The following Oracle Java CAPS project models with HL7 OTDs are good candidates for migration.

- The project contains a business process with HL7 OTD marshal and unmarshal activities.
- The project contains a business process that invokes a JCD with HL7 OTDs as both inbound and outbound connections.
- The project contains a standalone JCD unmarshaling and marshalling into an HL7 OTD structure from a File or JMS inbound adapter.

When you migrate projects that have HL7 OTDs as input and output in a JCD, do not use the `-usejaxb` option. Using the `-usejaxb` option in this case could cause the compilation of the migrated JCD Java code to fail because not all methods are available in the JAXB objects. This can also occur with other types of OTDs.

1.2.2.2.3 Marshal and Unmarshal Migration

OTD marshal and unmarshal operations and functoids in business processes are modified by the migration tool to include embedded Java code that uses `OTDUtil` methods to perform `marshalToString` and `unmarshalFromString` operations. These operations are incorporated as BPEL business process Java Embedding activities. You can leave the Java Embedding activities as is in the business process, or you can modify the project's input JCA endpoint using the Adapter Configuration Wizard to use a different message schema. This internally converts the messages to the specified schema type so the marshal and unmarshal operations can be removed. If the input is a user-defined OTD, you can use the embedded Java implementation, modify the input to be an XML message, or define a schema for native format (nxsd).

1.2.2.2.4 About the Marshal and Unmarshal Processes

The marshal/unmarshal process has three primary steps:

- Call the `getVariableData` method of Oracle SOA Suite's `BPELXExecLet` class to retrieve the input variable.
- Call the `OTDUtil.otdOperation` method, using the OTD class name, OTD operation name, input element, and output container part name as parameters. This creates the OTD object, invokes the `unmarshalFromString` and `marshalToString` operations, and wraps the result in a DOM element with the root as the output container part name.
- Call the `setVariableData` method of Oracle SOA Suite's `BPELXExecLet` class to set the output to a BPEL variable.

1.2.2.2.5 Marshal and Unmarshal Examples for BPEL

The following examples illustrate migrated unmarshal and marshal operations.

Example 1–1 Migrated Unmarshal Operation

```
//1: Get OTD marshal/unmarshal operation input by passing inputvariable and part
name
org.w3c.dom.Element inputelem = (org.w3c.dom.Element)getVariableData
    ("customerunmarshalFromStringInput","text");
String otdClass = "ud1.customer689211042.Customer";
    //Name of OTD class generated by Java CAPS

//2: Invoke OTDUtil.otdOperation by passing OTDClassType, operationname, and input
element
Object output = migration.caps.runtime.OTDUtil.otdOperation(otdClass,
    "unmarshalFromString",inputelem,"customer");

//3 : Set OTD marshal/unmarshal operation output by passing outputvariable, part
name, and data element
setVariableData("customerunmarshalFromStringOutput","customer",output,false);
```

Example 1–2 Migrated Marshal Operation

```
//1: Get OTD marshal/unmarshal operation input by passing inputvariable and part
name
org.w3c.dom.Element inputelem =
  (org.w3c.dom.Element)getVariableData("customermarshalToStringInput","customer");
String otdClass = "ud1.customer689211042.Customer";
    //Name of OTD class generated by Java CAPS

//2:Invoke OTDUtil.otdOperation by passing OTDClassType, operationname, and input
element
Object output = migration.caps.runtime.OTDUtil.otdOperation(otdClass,
    "marshalToString",inputelem,"text");

//3:Set OTD marshal/unmarshal operation output by passing outputvariable, part
name, and data element
setVariableData("customermarshalToStringOutput","text",output,false);
```

The `otdClass` parameter is null for FCX OTDs because they cannot be loaded in Oracle SOA Suite due to XMLBean incompatibility. The `OTDClass` parameter can be null for XML-based OTDs, in which case the marshal/unmarshal operation is done without loading the actual OTD. In this case, `OTDUtil` transforms the XML DOM to a string (marshal) or a string XML back to XML DOM (unmarshal).

1.2.2.2.6 Marshal and Unmarshal Operations in JCDs

The Oracle Java CAPS Migration Tool converts OTDs used in JCDs to JAXB objects. Migrated Spring classes can use these JAXB objects in most cases. You can also use the `jaxbMarshalToOTD` and `jaxbUnmarshalFromOTD` methods from `OTDUtil` to convert JAXB to OTD and OTD to JAXB if required.

1.2.2.3 Support for Oracle Java CAPS Framework Classes

During the migration, code is generated to handle Oracle Java CAPS framework classes in Oracle SOA Suite. This includes the logger, collaboration context, and type converter classes. Alerter classes are not handled by the migration tool. The migrated code includes access methods for these classes, and the references to these classes are instantiated by Spring context injection. The classes are located in `jcaps_interfaces.jar` in the `Application Sources` folder of the migration project.

1.2.3 OTD to XSD and XSD to JAXB Conversion

The migration tool provides can be run in different modes to convert OTDs to XSD format, and XSD format to JAXB objects after the initial project migration is complete.

- Use the `-xsd2jaxb` option to convert XSD to JAXB. For more information, see Section 3.4, "Converting XSD to JAXB Format."
- Use the `-otd2xsd` option to convert OTD to XSD. For more information, see Section 3.3, "Converting OTD to XSD Format."

By default, the migration tool retains the original JCD interface. An additional option, `-usejaxb`, indicates to use JAXB objects in the JCD's input and output arguments instead of converting them to OTDs to gain slightly better performance during runtime.

1.2.4 JAXB Generation During Migration

Oracle Java CAPS projects that contain a business process that invokes a JCD or that contain a JCD exposed as web service are special migration cases. These projects contain a WSDL interface that is converted to a JAX-WS proxy interface during migration. The OTD message types defined in the WSDL document are converted to JAXB objects.

To generate the JAXB objects, the migration tool uses a specific version of the following JAXB and XJC libraries. The versions listed below cannot be used within WebLogic Server, and are solely included for use with the Oracle Java CAPS Migration Tool.

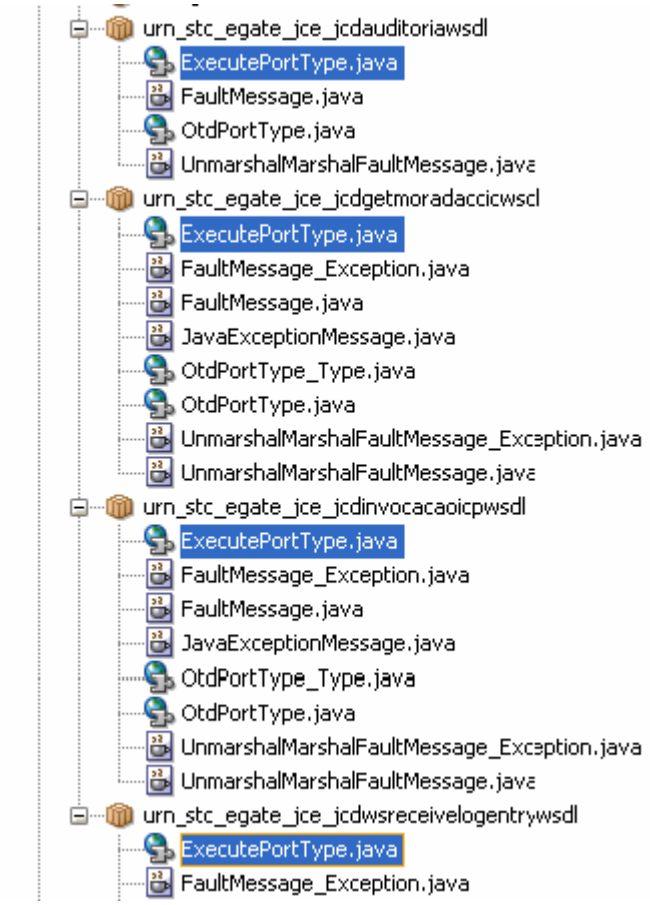
- *Oracle_SOA_Home*`/oracle_common/modules/glassfish.jaxb.xjc_1.0.0.0_2-1-12.jar`
- *Oracle_SOA_Home*`/oracle_common/modules/glassfish.jaxb_1.0.0.0_2-1-12.jar`

The above versions of these libraries include the following processing rules in order to remain consistent with the original OTD behavior.

- Underscores in the original XSD files are retained in the generated JAXB objects.
- Lowercase letters after an underscore remain in lowercase in the generated JAXB objects.
- When there are nested complex types in the original XSD file, the getter method for the generated JAXB object checks if the object is null and creates a new object if it is.

The generated JAX-WS proxy classes are located in the Oracle SOA Suite projects `src` folder under the WSDL packages. The file name for the proxy classes is `ExecutePortType.java`.

Figure 1–1 Generated JAX-WS Proxy Classes in an Oracle SOA Suite Project



1.2.5 Support for JBI Project Migration

The migration tool migrates all Oracle Java CAPS JBI projects that use business processes with File, JMS, or HTTP binding components. It also migrates business processes that call subprocesses. JBI projects are more similar to Oracle SOA Suite projects, and require less transformation than Repository projects.

While both Oracle Java CAPS JBI and Oracle SOA Suite support BPEL 2.0, certain constructs are not supported in Oracle SOA Suite and will require some manual changes after the project is converted by the migration tool.

1.2.5.1 WS-I Version 1 Compliance

Oracle SOA Suite is compliant with the Web Services Interoperability Organization's (WS-I) Basic Profile, version 1.0. This means that the WSDL document can only have one part, which must have a type of `element`. It cannot be `complextype` or `simpletype`. Make sure any JBI projects to be migrated are WS-I compliant; otherwise the migrated project might fail during build, deployment, or runtime. The migration utility does not check for WS-I compliance.

1.2.5.2 Same Target Namespace for Different Message Definitions

Oracle Java CAPS JBI projects allow WSDL documents to use the same target namespace but override the message definitions. In other words, the element name is the same but the structure is different. In Oracle SOA Suite, only the first WSDL document is loaded and the overridden definitions are lost. If the JBI project you are

migrating contains multiple WSDL documents with the same target namespace, modify the target namespace so it has a unique name, or make sure that the WSDL documents using the same target namespace do not define message type definitions with same name.

1.2.5.3 System Properties

In Oracle Java CAPS, additional Quality of Service (QoS) properties can be configured on the service connections, such as redelivery and throttling. Oracle SOA Suite does not have corresponding properties. The migration tool ignores these configurations during the migration. You can configure the project after the migration using available Oracle SOA Suite properties. For more information, see Section 4.2.3, "Configuring Quality of Service Properties."

1.2.5.4 BPEL 2.0 Constructs

Some of the BPEL 2.0 constructs that Oracle Java CAPS JBI supports are not available in the Oracle SOA Suite. During migration, the BPEL 2.0 code is converted without any modifications to the BPEL constructs. Some constructs that worked in Oracle Java CAPS will not work in Oracle SOA Suite, including the following:

- Compensation and termination handlers
- The standard faults: forcedTermination, repeatedCompensation, and invalidReply
- The partner concept (a partner groups several partnerLink elements, but it did not have any executable properties so it was removed from BPEL 2.0)

For compensation and termination handlers, the partner links are migrated as placeholders that you need to recreate. The `switch` element was also removed from BPEL 2.0, but the migration tool replaces the `switch` element with the if element in the Oracle SOA Suite project.

For post-migration steps to address unsupported constructs, see Section 4.2.4, "Verifying the BPEL Structure."

1.3 About the Migrated Oracle SOA Suite Projects

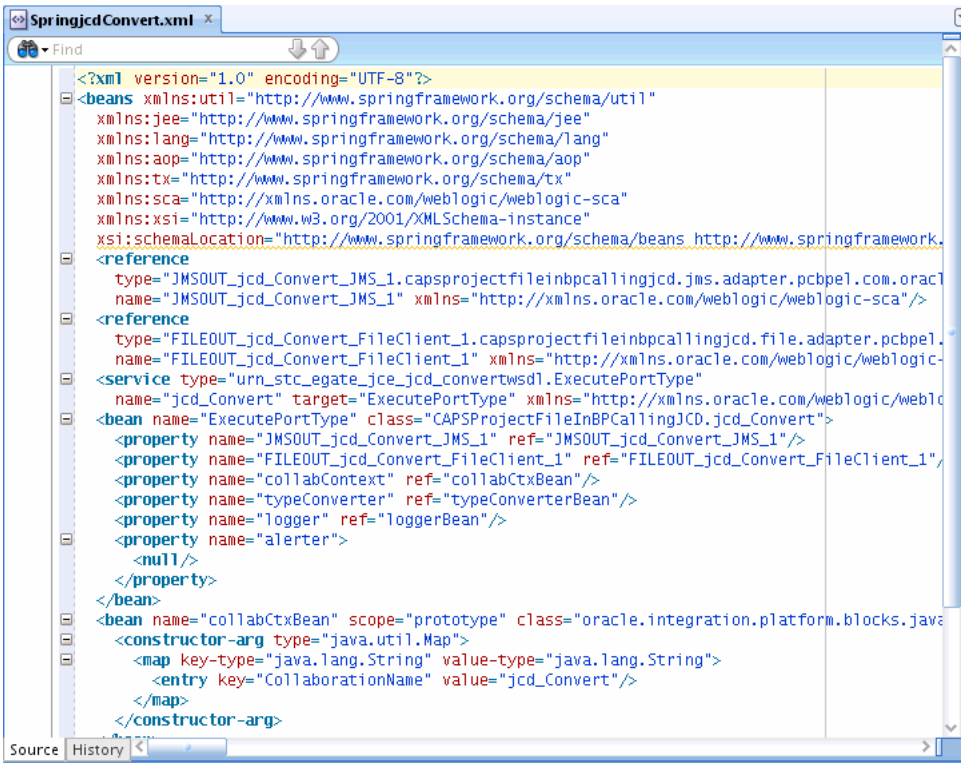
In the migrated Oracle SOA Suite project, all JCDs are now Spring Context components, and any BPEL 1.0 business processes are now BPEL 2.0 processes. For every supported JCD in the original Oracle Java CAPS project, the Oracle SOA Suite project includes a Spring component and context XML file with the respective services and references nodes. The business processes and Spring Contexts are automatically wired to the inbound and external systems, though additional wiring might be required after the project is migrated.

1.3.1 About the Conversion to Spring

Each JCD is converted to a Spring component with an associated Spring context XML file. The JCD Java class is also brought over to the Oracle SOA Suite project and is modified to implement the JAX-WS proxy method along with all the accessor methods for the logger, collaboration context, and type converter.

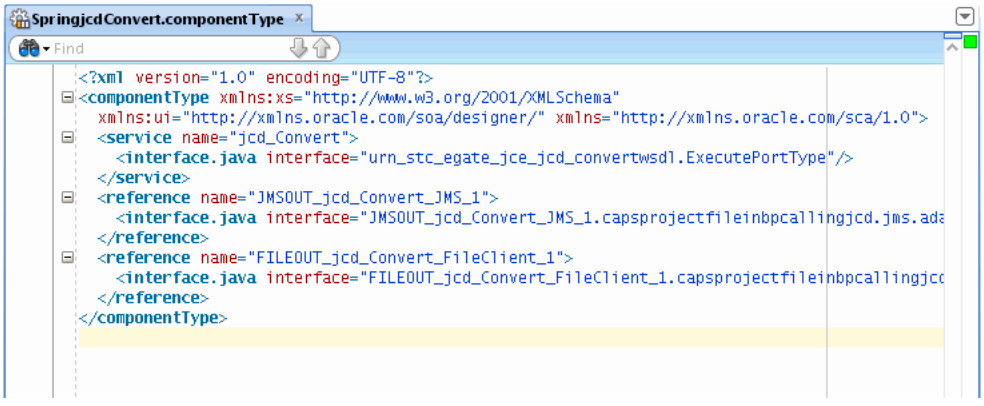
The generated Spring Context includes a `bean` element that represents the original JCD class. In addition to defining JCD logic, the Bean defines additional properties such as the logger, collaboration context, and type converter, which are injected into the Spring context at runtime.

Figure 1–2 Spring Context XML File



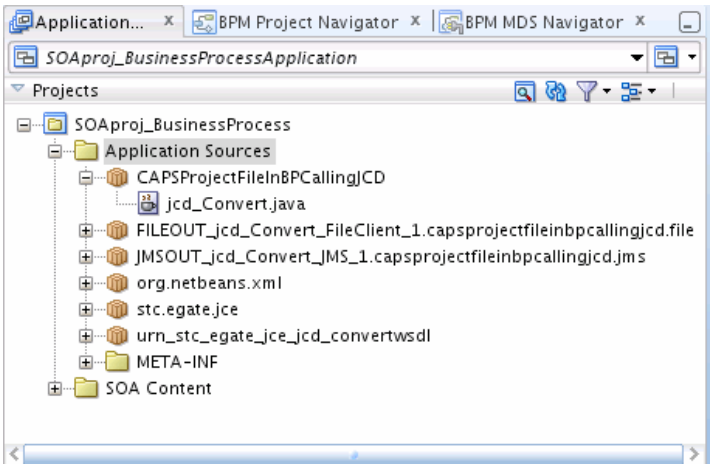
For each JMS or File adapter outbound interaction in the original JCD code, the Spring Context includes an additional `bean` element reference, which is also injected at runtime. You need to invoke the Bean's API to interact with the outbound adapters.

Figure 1–3 Spring Component Type File



For each JCD, a Spring component type is also generated. The component type defines the JAX-WS interface class that was implemented by the JCD class. All the JAX-WS proxy classes and SOA Suite adapter interface classes are placed in the same Application Sources folder in the Oracle SOA Suite project.

Figure 1–4 Application Sources in the Oracle SOA Suite Project



1.4 Migration Considerations

The Oracle Java CAPS Migration Tool supports business process and JCD Repository projects at versions 5.0.5 at any update level, 5.1.3 at any update level, and 6.0 or later. The migration tool supports JBI projects at version 6.0 or later. Depending on the individual project, there are variable levels of reuse for the mappings and other business logic in the project. The recommended reuse level depends on many factors, including the number of mappings, the complexity of the mappings, the types of OTDs used, the adapters and binding components used, and so on. Although all mappings can be retained as-is in most cases, this might not be the best approach.

Generally, the best candidates for a JCD-to-Spring migration are those that meet the following criteria:

- The Java code is longer than 100 lines and contains several mappings.
- Mappings are complex.
- Adapters in the project include File, JMS, or Web Service.
- The project includes user-defined OTDs or complex message OTDs, such as X12 or HL7.
- The project uses custom JAR files that are invoked inline within several mappings.

1.4.1 Deciding Whether to Migrate

Below are a few considerations to take into account when making migration decisions for Oracle Java CAPS projects.

- Is this a trivial JCD project with, for example, fewer than 100 lines of code and with simple mappings? If so, consider rewriting the JCD using Oracle SOA Suite components and native XML instead of reusing the legacy Oracle Java CAPS OTDs. Exclusive use of Oracle SOA Suite components is likely to outperform a blend of Oracle Java CAPS and Oracle SOA Suite due to the conversion overhead to and from the Oracle Java CAPS OTD format.
- Is there a better way to do the necessary processing using just Oracle SOA Suite components that were not available in Oracle Java CAPS? For example, routing logic could be abstracted into a higher level Oracle Mediator project for improved design.

- The effort to migrate certain Oracle Java CAPS projects to Oracle SOA Suite can be very large, depending on the complexity of the project.
- Migrating Repository projects that include business processes requires transforming the BPEL 1.0 code to BPEL 2.0. This conversion is performed automatically by the migration tool, but might require some manual changes due to support for different BPEL 2.0 constructs between Oracle Java CAPS and Oracle SOA Suite. A manual transformation process would require additional resources and time to complete a seamless integration with the remaining components.
- While much of the migration is performed automatically, the migration of Repository projects that include JCD's require to some manual conversion of the JCD Java code to Oracle SOA Spring component.
- Migrating Repository projects with certain messageable OTDs could require a significant amount of rework and redesign to work in Oracle SOA Suite.

1.4.2 Deciding How to Migrate

Once the decision is made to migrate an Oracle Java CAPS JCD project, ask yourself the following questions to determine how to proceed:

- Are there relatively few JCD mappings that rely on Oracle Java CAPS adapter methods?

  If the number of mappings affected is minimal, replace the Oracle Java CAPS OTD used in the Oracle Java CAPS mappings in the Oracle SOA Suite project with the corresponding Oracle SOA Suite adapter API. This eliminates the need for Oracle Java CAPS adapter code from your JCD, which means there will be no runtime dependencies on legacy Oracle Java CAPS adapter APIs.

- Is there a large number of JCD mappings that rely on Oracle Java CAPS methods?

  If the number of mappings affected by the adapter code is very large with Oracle Java CAPS adapter methods heavily embedded in the mappings, you might want to leave the Oracle Java CAPS adapter OTD as is to allow maximum reuse of the existing mappings. The downside is that the migrated code would have dependencies on legacy Oracle Java CAPS adapter libraries. You would also need to create additional code outside of the mapping to translate the Oracle Java CAPS adapter OTD to the Oracle SOA Suite adapter's XML object. For example, you might need to create additional code to map database OTD fields to Database Adapter XML format.

1.5 Limitations of the Migration Tool

While the Oracle Java CAPS Migration tool automates several tasks for migrating existing Oracle Java CAPS projects to Oracle SOA Suite, there are some limitations in the project migration due to the wide variety of possible component combinations in an Oracle Java CAPS project. As a result, manual steps are required to complete most migrations. The required changes might be due to the nature of the artifacts to migrate, or they might be because the tool cannot convert the artifacts to corresponding Oracle SOA Suite artifacts. It is also possible that the artifact relies on features or constructs that are not supported in Oracle SOA.

Below are some of the limitations that we came across when developing the Migration tool.

Oracle Java CAPS Repository Project Migrations

- The input message content needs to be wrapped with an element if the message type is not defined as opaque. Both File and JMS messages require this wrapper node when not defined as opaque. In Oracle SOA Suite, only XML messages are processed when the message type is not defined as opaque.

  For more information, see Section 4.3.1, "Enabling File or JMS Message Types as Opaque."

- Certain Repository products are not supported by the migration tool at this time. This includes components such as Oracle Java CAPS Master Index, Oracle Java CAPS Data Integrator, Oracle Java CAPS B2B Suite, Oracle Java CAPS BAM, and so on.

  Adapters other than File, JMS, and web services are migrated as placeholders, which can later be replaced with the corresponding Oracle SOA Suite adapters.

- Certain JBI products are not supported by the migration tool at this time, including service engines other than the Oracle Java CAPS BPEL Service Engine.

  Binding components other than File, HTTP, and JMS are migrated as placeholders, which can later be replaced with the corresponding Oracle SOA Suite adapters.

- The migrated JCD code requires some modification in the Spring Bean when there are methods with multiple return statements. An example would be to have a conditional returns within a method.

- Receive operations on Flow branches with the `createInstance` set to yes are not permitted in Oracle SOA Suite and will fail during the build process.

- When a project includes multiple business processes, the target namespace and the business process name cannot be the same.

- Oracle SOA Suite expects the start activity to be a message activity. No other activities are allowed before the start activity that creates the business process instance.

- Only inbound Web Service adapters, inbound and outbound File adapters, and inbound and outbound JMS adapters are supported with a standalone JCD.

- The migration tool cannot migrate user activities (Worklist Manager) in business processes. The business process would contain BPEL constructs specific to Oracle Java CAPS, which would need to be manually replaced with the Oracle SOA Suite work flow.

- When correlation initialization for a business process is in the marshal and unmarshal Invoke activities, it is ignored because marshal and unmarshal activities are converted to Java Embedding activities in the migrated business process. Manual modifications are required in the migrated project.

- Version control for Repository projects is not migrated.

- Concatenating optional nodes in an assign activity succeeds in the migrated project even when one or more nodes in the concatenated from expression is missing or has no data. This is due to the `ignoreMissingFromData` property being set to yes in Oracle SOA Suite, as shown below.

```
<assign name="SetdataNascimento">
  <copy ignoreMissingFromData="yes">
    <from>concat(concat(concat(concat(
      $DateunmarshalFromStringOutput1.Date/YEAR, '-'),
      $DateunmarshalFromStringOutput1.Date/Month), '-'),
      $DateunmarshalFromStringOutput1.Date/Day)
```

```
  </from>
  <to>$ICObterNomeDnNatMorValPortTypeICObterNomeDnNatMorValInput.
      ICObterNomeDnNatMorValResponse/qual3:dadosPesquisa/qual3:dataNascimento
  </to>
</copy>
```

Note: The above code has been wrapped for ease of viewing.

2

Installing the Oracle Java CAPS Migration Tool

This chapter describes how to install the Oracle Java CAPS Migration Tool and associated JAR files, and how to configure the Oracle Java CAPS environment for migration.

This chapter includes the following topics:

- Section 2.1, "About the Installation"
- Section 2.2, "Installing the Migration Tool"

2.1 About the Installation

The migration tool is provided in a ZIP file, and installing the tool involves simply downloading and extracting the file. The installation includes the following files:

- **`MigrationTool.jar`**: This utility converts an Oracle Java CAPS project to an Oracle SOA Suite project. It can also be used to convert OTD format to XSD, and XSD format to JAXB objects.
- **`logging.properties`**: This file sets the logging levels for the migration process.
- **`oracle.migrationtool.jcaps.libraries`**: This folder contains all the required libraries for the migration tool.

Before you can use the migration tool, you need to install additional JAR files and configure the Oracle Java CAPS IDE. Both processes are described in this chapter.

2.1.1 Prerequisites

In order to run the migration tool, you must have the following installed and accessible from the location of the migration tool:

- Oracle Fusion Middleware (specifically, the oracle_common directory)
- Oracle SOA Suite

Additionally, you need to have access to Oracle Java CAPS 6.x or 5.x because certain projects need to be rebuilt and redeployed prior to migration.

2.2 Installing the Migration Tool

Installing the migration tool includes extracting the tool to a location on your computer, installing additional JAR files, and configuring the Oracle Java CAPS environment.

To install the migration tool

1. Download the `MigrationTool.zip` to a folder on your computer where you want to store the tool.
2. Extract the ZIP file.

To install third-party JAR files

- Download the following three files to the `oracle.migrationtool.jcaps.libraries/lib` folder in the migration tool installation:
  - Swing Application Framework: http://download.java.net/maven/2/org/jdesktop/appframework/1.0.3/appframework-1.0.3.jar
  - Swing Worker API: http://download.java.net/maven/2/org/jdesktop/swing-worker/1.1/swing-worker-1.1.jar
  - Java Parser: https://docs.google.com/uc?id=0B6K9IqkgGbcgZTg3ZWMzMzYtNDQ4Zi00NzBmLWI1OTUtOTM2MTViMjYyMmEw&export=download&hl=en_US

    Note: This link requires a login to a Google account.

To set up the Oracle Java CAPS environment

Adding a new flag to the Oracle Java CAPS IDE preserves the Java code from any JCDs in the generated EAR files, which is required for the migration to complete successfully.

1. If you are migrating Oracle Java CAPS 6.x Repository projects that contain Java Collaboration Definitions (JCDs), do the following:
   a. In your Oracle Java CAPS home directory, navigate to `/netbeans/etc`.
   b. Open `netbeans.conf` in a text editor.
   c. Append the following flag to the end of the `netbeans_default_options` property before the closing double quote:

      ```
      -J-Drun.mode=debug
      ```

   d. Save and close the file.
2. If you are migrating Oracle Java CAPS 5.x projects, do the following:
   a. In your Oracle Java CAPS home directory, navigate to `/edesigner/bin`.
   b. Open `runed.bat` in a text editor.

      Do not double-click the file to open it; double-clicking the file launches Enterprise Designer.
   c. Append the following flag to the end of the `netbeans_default_options` property before the closing double quote:

      ```
      "-J-Drun.mode=debug"
      ```

   d. Save and close the file.

3 Migrating the Projects

This chapter provides instructions on how to perform an Oracle Java CAPS to Oracle SOA Suite migration using the provided migration tool.

This chapter includes the following topics:

- Section 3.1, "Overview of the Migration Tool Process"
- Section 3.2, "Migrating Oracle Java CAPS Projects"
- Section 3.3, "Converting OTD to XSD Format"
- Section 3.4, "Converting XSD to JAXB Format"

3.1 Overview of the Migration Tool Process

You can run the migration tool in either wizard mode or command-line mode. Wizard mode provides more visibility into the migration process by letting you view the Oracle Java CAPS components and file content that will be migrated and the Oracle SOA Suite files and content that will be generated. The migration tool uses the generated project EAR or ZIP file from the Oracle Java CAPS project in order to determine how to generate the Oracle SOA Suite project.

The output of the migration tool includes the necessary project components along with the Oracle JDeveloper project file (a JPR file). These artifacts include the `composite.xml` file, and can also include Spring component files, BPEL business process files, WSDL files, schema definitions, JAR files, Java classes, and so on.

Certain projects will require manual changes to the migration tool output, such as projects with outbound adapters other than File or JMS that would require you to replace the generated placeholders with Oracle SOA Suite adapters. Changes that you might need to make to the migration tool output are described in Chapter 4, "Post-Migration Tasks."

3.2 Migrating Oracle Java CAPS Projects

This section describes running the migration tool in both wizard and command-line modes. Make sure to complete the steps listed in Section 3.2.1, "Before you Begin" prior to running the migration tool

3.2.1 Before you Begin

The Oracle Java CAPS files to be migrated need to be configured correctly and then rebuilt and redeployed with the new debug options described in the installation instructions. Perform the following before running the migration tool. The first two

steps are optional, depending on the type of project you are migration, but the final step must be performed in order for the migration to process correctly.

- Verify JBI Projects for Compliance
- Modify Business Processes
- Modify Projects with File Write Operations
- Rebuild and Redeploy the Projects to Migrate

3.2.1.1 Verify JBI Projects for Compliance

While most JBI BPEL projects can be successfully migrated, you should verify the project components for compliance with Oracle SOA Suite supported standards. For information about the differences between Oracle Java CAPS JBI and Oracle SOA Suite and any changes you might need to make, see Section 1.2.5, "Support for JBI Project Migration."

3.2.1.2 Modify Business Processes

If you are migrating a project that includes multiple business processes, the target namespace and business process name cannot be the same in order for the migration to process correctly. If you have multiple business processes with the same namespace, modify one of the names prior to the migration.

Oracle SOA Suite requires that the start activity be a message activity, while Oracle Java CAPS allows an assign activity to be the first activity. If you have any projects in which the start activity is not a message activity in a business process, modify the business process prior to beginning the migration.

If the prefixes in the assign activities have duplicate definitions with different namespaces, you need to resolve the ambiguity and assign the mapping with the correct prefix.

3.2.1.3 Modify Projects with File Write Operations

Some older Oracle Java CAPS projects have a File Write operation with no output container. These projects will not build in Oracle SOA Suite. If you have any projects with this condition, set the output variable before performing the migration.

3.2.1.4 Rebuild and Redeploy the Projects to Migrate

In order to perform the migration for Repository projects, you need to rebuild and redeploy the projects using the updated debug flags, as described in To set up the Oracle Java CAPS environment. You then need to make the generated EAR file from the project accessible from the location of the migration tool. For JBI projects, the generated Composite Application Service Assembly (CASA) ZIP file needs to be made available. Both Repository and JBI projects need to be rebuilt and redeployed with the updated flags.

3.2.2 Setting the Migration Logging Properties

You can configure how much of the migration processing information is logged in the migration log files and is written to the console where you run the migration tool. Logging levels are set on three levels: global, per handler, and for the migration tool itself. The handler settings override the global settings.

You can specify the following log levels, listed in order from least information given to most:

- SEVERE (failure messages only)
- WARNING
- INFO
- CONFIG
- FINE
- FINER
- FINEST (most detailed logging)

To configure the migration logging properties

1. Navigate to the location where you installed the Oracle Java CAPS Migration Tool.
2. Open `logging.properties` in a text editor.
3. Modify any of the properties listed in the following table, and then save and close the file.

Property	Description
handlers	A comma-delimited list of log handler classes to use for the migration. The default is `java.util.logging.FileHandler, java.util.logging.ConsoleHandler`. This sends log messages to both the log file in the migration tool folder and to the console from which you run the migration tool.
level (global)	An indicator of the granularity of log messages across all loggers. This value is overridden by the log levels you specify for each logger later in the file. By default, it is commented out and not used.
FileHandler Properties	
pattern	A pattern indicating the name given to each log file. The default value is `migrationtool_%u.log`, which creates log files name `migrationtool##.log` where ## indicates a number that is incremented for each log file.
level	An indicator of the granularity of log messages for the events logged in the log file.
limit	The maximum number of bytes logged in each file. If this value is 0 (zero), there is no limit.
count	The number of output files to cycle through.
formatter	The name of the Java formatter class to use to format the messages in the log file. The default is `java.util.logging.SimpleFormatter`.
ConsoleHandler Properties	
level	An indicator of the granularity of log messages for the events displayed on the console.
formatter	The name of the Java formatter class to use to format the messages on the console. The default is `java.util.logging.SimpleFormatter`.
Migration Tool Properties	
level	An indicator of the granularity of log messages for the migration tool in general.

3.2.3 Migrating a Project Using the Wizard

The migration tool wizard provides a graphical view of the project components from the original Oracle Java CAPS project and the components generated by the migration tool for the Oracle SOA Suite project.

To migrate a project using the wizard

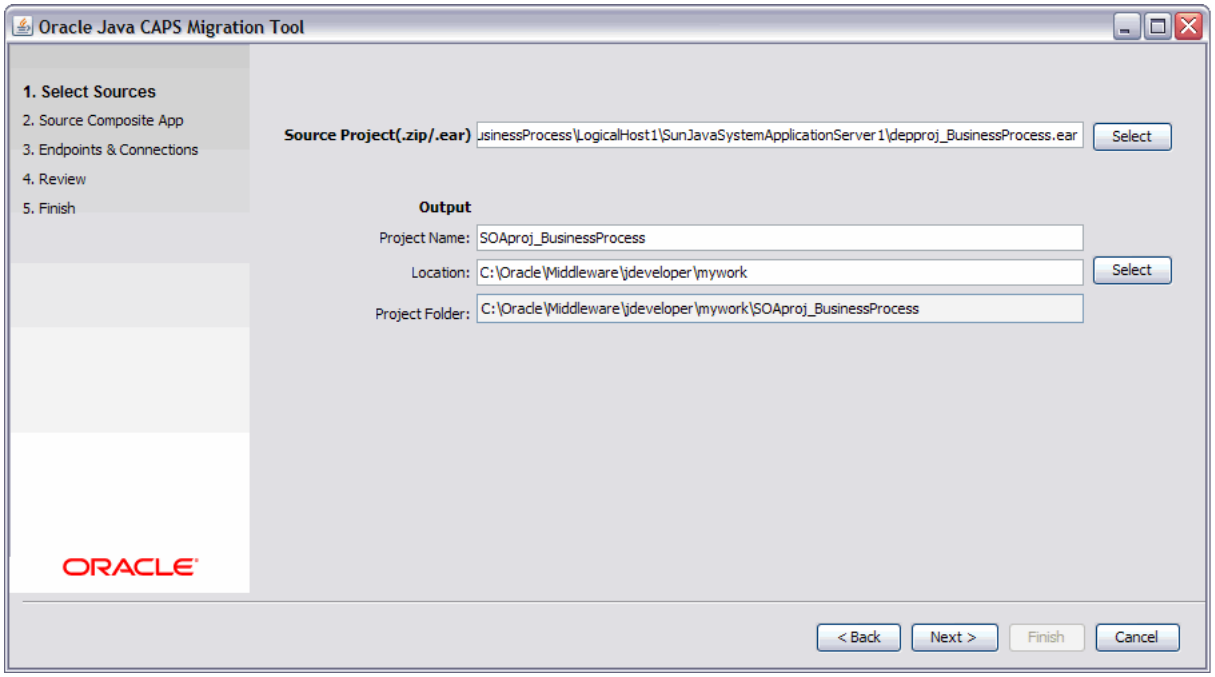
1. From a command line, run the following command:

```
java -jar MigrationTool.jar -soahome middleware_home -wizard
```

> **Note:** If the project being migrated includes JCDs that are either invoked by a business process or that are exposed as web services, add the `-usejaxb` option to the end of the command. The *middleware_home* directory is the path to `oracle_common` in your Oracle Fusion Middleware installation. For more information about these options, see Section 3.2.4.1, "Migration Tool Usage."

2. On the migration wizard, enter the following information:
   - **Source Folder**: The path and filename of the Oracle Java CAPS EAR (Repository project) or ZIP (JBI project) file to migrate.
   - **Project Name**: This is automatically populated with the name of the EAR file or composite application.
   - **Location**: The directory where the migration tool will place the generated Oracle SOA Suite project. If not supplied, the current working directory is used.
   - **Project Folder**: This is automatically populated with the full path where the migrated Oracle SOA Suite project will be placed.

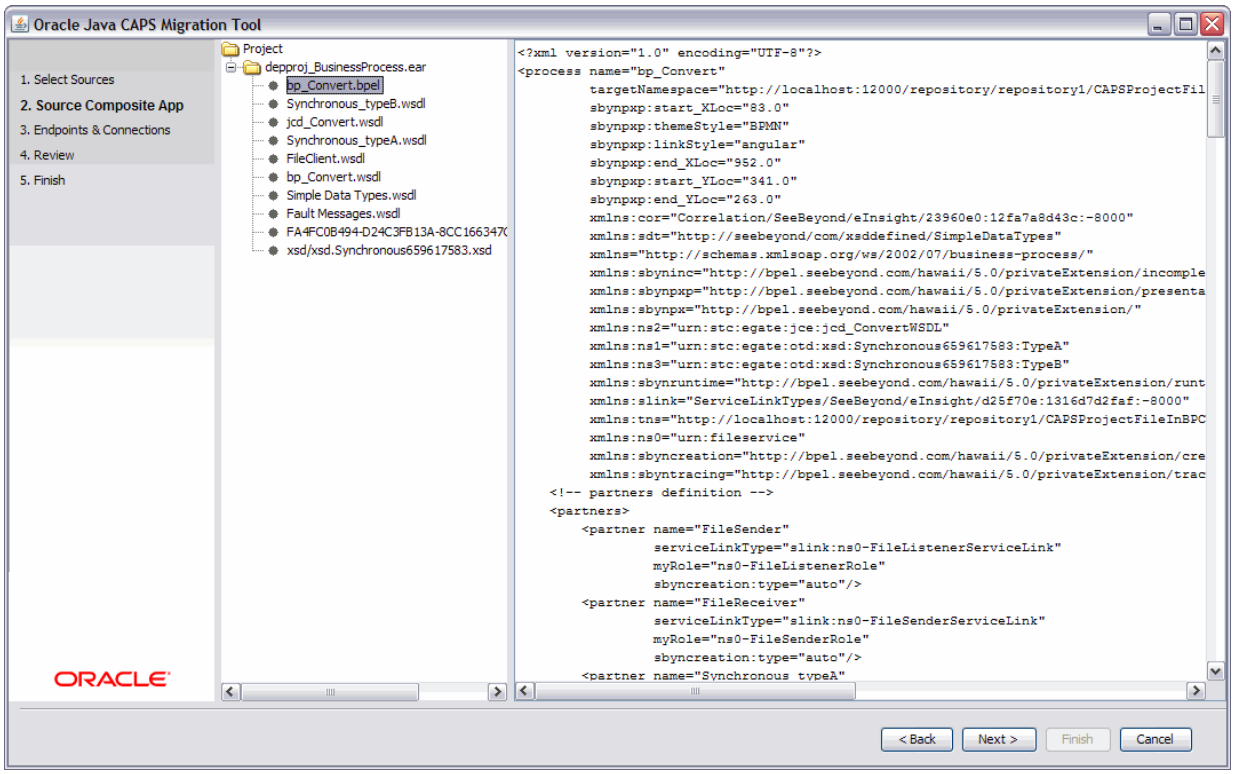
Figure 3–1 Oracle Java CAPS Migration Tool - Select Sources Page



3. Click **Next**.

   The migration tool parses through the project's ZIP or EAR file, extracts the required artifacts, and, if there are any business processes in the project, displays the artifacts in the wizard.

Figure 3–2 Oracle Java CAPS Migration Tool - Source Composite App Page



4. Do one of the following:

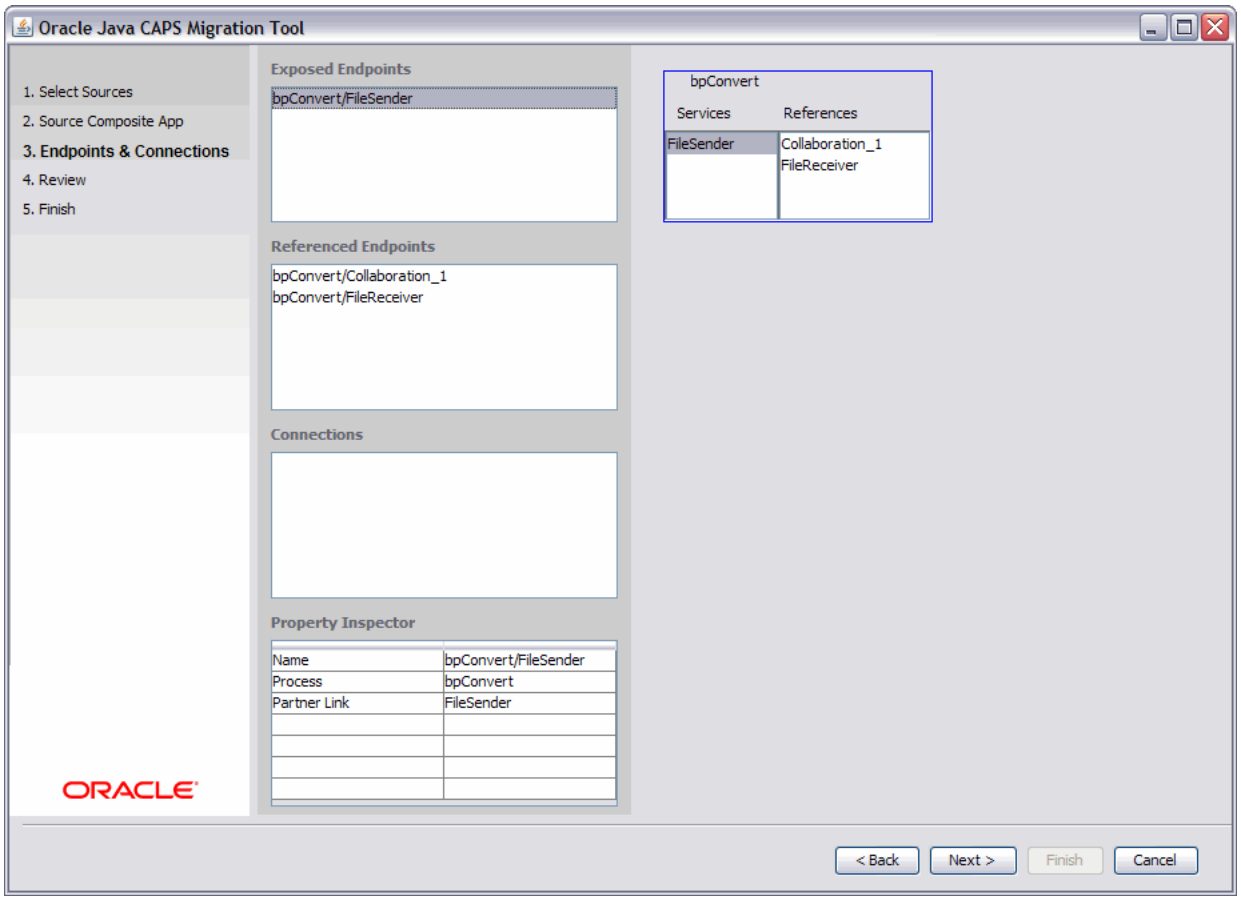
   a. If there are no business processes in the project, skip to step 8.

   b. If there are business processes in the project, verify the contents of the artifacts by clicking on the name in the canvas on the right.project explorer.

5. Click **Next**.

   The wizard renders the parsing results to display the services, references, connections, and properties for the project (again for business processes only).

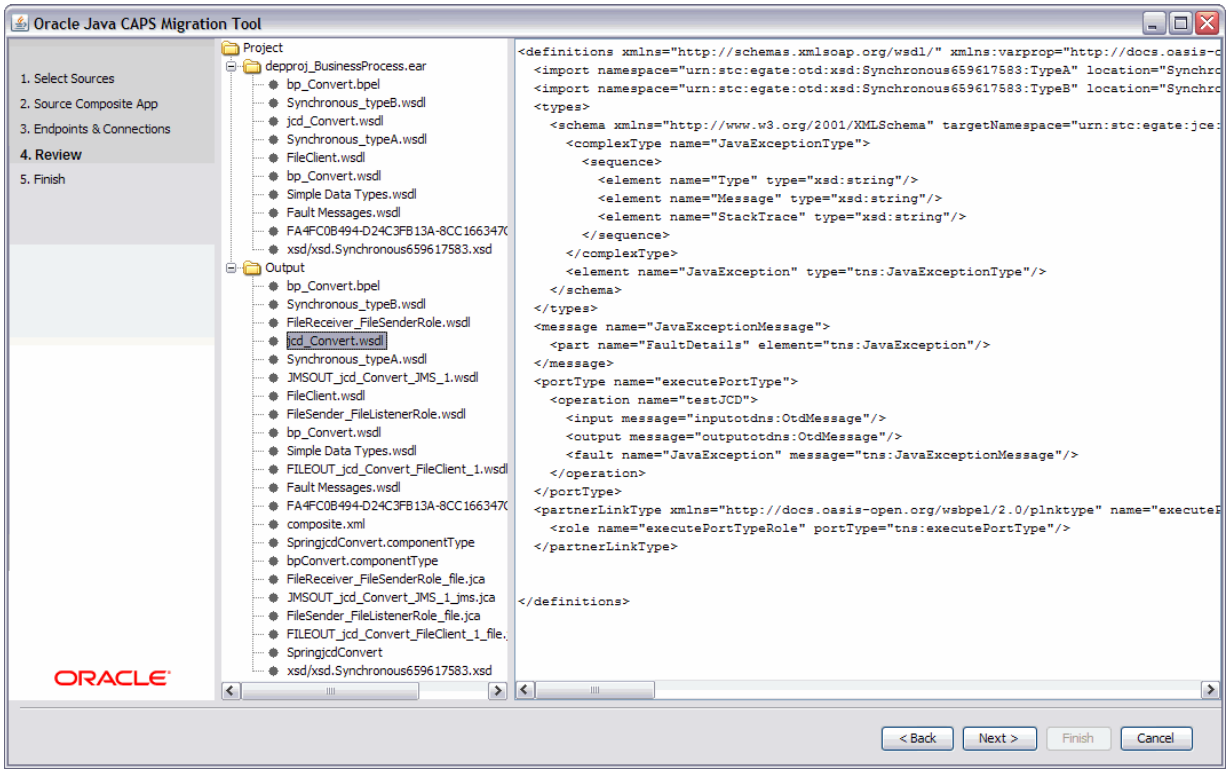
Figure 3–3 Oracle Java CAPS MIgration Tool - Endpoints & Connections



6. Click on any of the exposed or referenced endpoints to view their associated partner links. Click on a connection to highlight the two partner links for the connected components (this represents a "wire" in the Oracle SOA Suite composite).
7. Click **Next**.

   The migration tool generates the Oracle SOA Suite files and presents them to you for review. You can check the console window for processing messages and status. If an error or exception occurs, the migration process stops and the console displays the cause of the error.
8. On the Review page, review the content of the migrated artifacts and check the output project location for the files.

Figure 3–4 Oracle Java CAPS MIgration Tool - Review Page



9. When you are ready, click **Next**.
10. On the Finish page, click **Finish**.

    When the migration process is complete, the wizard returns to the Select Sources page so you can begin migrating another project if desired. You can check the log files in the migration tool directory to review the migration process.
11. Certain components require manual changes or manual migration. For more information and instructions, see the following:
    - Chapter 3.4, "Converting XSD to JAXB Format"
    - Chapter 3.3, "Converting OTD to XSD Format"
    - Chapter 4, "Post-Migration Tasks"

3.2.4 Migrating a Project Using the Command Line

When you migrate an Oracle Java CAPS project using the command line mode instead of the wizard mode, you specify all of the required information in the command, and do not see a preview of the components and files that will be migrated or the components and files that will be generated. The console displays the processing status of the migration along with any warnings or errors that occur.

3.2.4.1 Migration Tool Usage

Syntax

```
java -jar MigrationTool.jar <-soahome parent_directory_of_oracle_common>
 [-archive path_of_input_file] [-projectname output_project]
 [-output output_folder] [-wizard] [-usejaxb]
```

Table 3–1 Command Line Options and Flags

Option/Flag	Description
-soahome	The path to the Oracle Fusion Middleware Home directory. This is the parent directory to the `oracle_common` directory, which is required for JAX-WS interface generation. For example, `C:\Oracle\Middleware`. This option is required for both command-line and wizard modes.
-archive	The path and filename of the Oracle Java CAPS EAR or ZIP file to migrate.
-projectname	The name of the Oracle SOA Suite project that the migration tool will generate.
-output	The directory where the migration tool will place the generated Oracle SOA Suite project. If not supplied, the current working directory is used.
-wizard	A flag that starts the migration tool in wizard mode, launching a graphical interface where you can specify migration options and view project components. When the command is run in this mode, all other options except `-soahome` are optional. If you provide the other options, their values are automatically populated in the migration wizard.
-usejaxb	A flag that modifies the original JCD interfaces to use JAXB objects directly to avoid copying them to and from Oracle Java CAPS OTD objects in the migrated code. This flag is only valid for migrating projects with JCDs that are invoked by a BPEL business process, and with standalone JCDs that are exposed as web services. The tool automatically enables this option when the JCD uses First Class (FCX) OTDs and replaces the FCX OTDs (XMLBeans) with the JAXB objects. When migrating projects that have HL7 OTDs as input and output in a JCD, do not use the this option. **Caution:** When the migration tool is run using the `-usejaxb` option, the JCD Java code might fail to compile in the migrated Oracle SOA Suite project because some of the OTD helper methods are not available on the generated JAXB objects. Only the getter and setter methods for the XSD fields are available. When an OTD includes additional helper methods other than getter and setter methods for OTD fields, do not use this option when running the migration tool.

Example 3–1 Command Line Syntax

```
java -jar MigrationTool.jar -soahome C:\Oracle\Middleware\ -archive
 C:\javacaps\migration_projects\JCDOTD_CAPSProject_20110930.ear -projectname
 PurchaseOrder -output C:\SOA_Suite\mywork -usejaxb
```

3.3 Converting OTD to XSD Format

The migration tool converts most of the OTDs during the migration process. For any OTDs that are not automatically converted, the migration tool provides an option, `-otd2xsd`, that generates an XSD file based on the provided OTD class. The generated XSD file can then be used in a migrated BPEL business process or to generate JAXB objects using `-xsd2jaxb` option as describes in Section 3.4, "Converting XSD to JAXB

Format." Use this method when an OTD does not have a corresponding schema definition.

Running the following command generates XSD files for the specified OTD. The XSDs are made available to the JAR files supplied with the `-otdclasspath` option.

```
java -jar MigrationTool.jar <-soahome parent_directory_of_oracle_common>
 <-otd2xsd> <-otdclass OTD class name to search in the supplied jar files>
 <-otdclasspath comma separated list of OTD and dependent jar files>
```

Table 3–2 Command Line Options and Flags

Option/Flag	Description
-soahome	The path to the Oracle Fusion Middleware Home directory. This is the parent directory to the `oracle_common` directory, which is required for JAX-WS interface generation. For example, `C:\Oracle\Middleware`.
-otd2xsd	An indicator to the migration tool to run in OTD to XSD conversion mode.
-otdclass	The class name of the OTD to be converted to XSD.
-otdclasspath	A comma-separated list of the absolute paths and filenames of the OTD and dependent JAR files.

Example 3–2 Converting OTD to XSD

```
java -jar MigrationTool.jar -soahome C:\Oracle\Middleware\ -otd2xsd -otdclass
 xsd.xsdICP_ICPOutput.ICPOutput -otdclasspath
 C:/migration_projects/xsdICP_ICPOutput.jar,
 C:/migration_projects/xsdICP_ICPOutput2.jar
```

The above text is wrapped for readability; enter the command all in one line with no spaces between the JAR file names. After running the OTD to XSD conversion, you can either convert the XSD to JAXB objects, as described in Section 3.4, "Converting XSD to JAXB Format," or add the XSD files to the migrated projects in Oracle JDeveloper.

3.4 Converting XSD to JAXB Format

If you use the migration tool to convert a project that uses Java Collaboration Definitions with FCX OTDs used as other OTDs, the migrated project fails at runtime because the FCX OTDs generated by Oracle Java CAPS are not compatible with Oracle SOA Suite. You need to run the migration tool again in XSD to JAXB conversion mode, which converts the FCX OTDs to JAXB objects.

Running the following command generates a JAXB object using the existing FCX OTD XML schema:

```
java -jar MigrationTool.jar <-soahome parent_directory_of_oracle_common>
 <-xsd2jaxb> <-jaxboutput JAXB output folder> <-schema XSD schema location>
 [-jaxbpackage package for generated JAXB objects]
```

Table 3–3 Command Line Options and Flags

Option/Flag	Description
-soahome	The path to the Oracle Fusion Middleware Home directory. This is the parent directory to the `oracle_common` directory, which is required for JAX-WS interface generation. For example, `C:\Oracle\Middleware`.

Table 3–3 (Cont.) Command Line Options and Flags

Option/Flag	Description
-xsd2jaxb	An indicator to the migration tool to run in XSD to JAXB conversion mode.
-jaxboutput	The absolute path to the location where you want the generated JAXB objects to be stored.
-schema	The absolute path and file name of the XSD file to use for the conversion.
-jaxbpackage	The Java package for the generated JAXB objects. This parameter is optional.

Example 3–3 Converting FCX OTDs to JAXB

```
java -jar MigrationTool.jar -soahome C:\Oracle\Middleware\ -xsd2jaxb -jaxboutput
 C:\javacaps\migration_projects\FCX_JAXB -schema
 C:\javacaps\migration_projects\JavaCAPS_FCX.xsd -jaxbpackage
 com.oracle.caps.conversion
```

The above text is wrapped for readability. After running the XSD to JAXB conversion, you need to add the JAXB objects to the migrated Oracle SOA Suite project and then modify the Spring components that contain the migrated JCD code to use the JAXB objects in place of the FCX OTD.

4

Post-Migration Tasks

This chapter provides instructions for updates you need to make to a project that was migrated from Oracle Java CAPS to Oracle SOA Suite. After the migration tool converts the project, you need to open the project in Oracle JDeveloper and modify some of the generated components and in some cases, add new components. You can then deploy those projects to a WebLogic Server.

This chapter includes the following topics:

- Section 4.1, "Opening a Migrated Project in Oracle JDeveloper"
- Section 4.2, "Configuring Migrated JBI Projects"
- Section 4.3, "Configuring Migrated Adapters and OTDs"
- Section 4.4, "Configuring Converted Oracle SOA Suite Spring Components"
- Section 4.5, "Configuring Business Processes"
- Section 4.6, "Adding JAR Files to a Migrated Project"

4.1 Opening a Migrated Project in Oracle JDeveloper

Once the Oracle Java CAPS Migration Tool completes a project migration, you need to open the generated Oracle SOA Suite file in Oracle JDeveloper to verify the project, and in some cases modify or add project components.

To open a migrated project in Oracle JDeveloper

1. Launch Oracle JDeveloper.

   You can launch Oracle JDeveloper by navigating to *jdeveloper_home*`/jdev/bin` and running the `jdev` executable file.

2. In the Oracle JDeveloper main menu, click **File** and select **Open**.

3. In the dialog that appears, browse to the folder where the migrated project files are located, and select the project file.

   Tip: This is the file with the JPR extension.

4. Click **Open** on the dialog.

   The Create Application to Contain Project dialog appears.

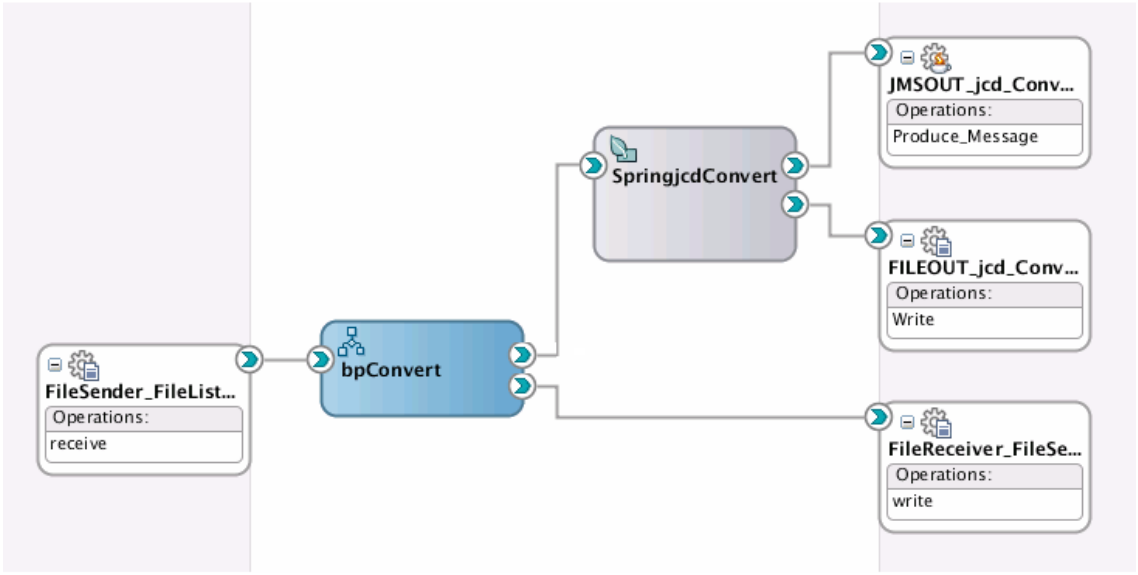
5. On the dialog, enter a name for the application and click **OK**.

   The project files appear in the Application navigation panel on the left.

6. In the navigation tree, expand **SOA Content** and open the `composite.xml` file.

Figure 4–1 shows an example of a converted project in Oracle JDeveloper. It includes a service, a business process, a Spring component, and several references.

Figure 4–1 Converted Oracle Java CAPS Project in Oracle JDeveloper



4.2 Configuring Migrated JBI Projects

After you process an Oracle Java CAPS JBI project using the migration tool, manual steps might be required to complete the migration and perform a successful deployment of the migrated project. The following are some updates you might need to make to your migrated JBI projects.

- Section 4.2.1, "Configuring Migrated Binding Components"
- Section 4.2.2, "Adding Service Elements"
- Section 4.2.3, "Configuring Quality of Service Properties"
- Section 4.2.4, "Verifying the BPEL Structure"

4.2.1 Configuring Migrated Binding Components

Not all Oracle Java CAPS binding component properties have a one-to-one correspondence with Oracle SOA Suite adapter properties, so the migration tool migrates only the essential set of properties. Before deploying your migrated project, you should run the configuration wizard for each service and reference in the composite to verify or update the configuration.

To configure a service or reference

1. Open the project and its composite in Oracle JDeveloper as described in Section 4.1, "Opening a Migrated Project in Oracle JDeveloper."
2. Double-click the service or reference you want to configure, or right-click the component and select **Edit**.

   The configuration wizard for that component appears.
3. Follow the steps on the wizard to complete the configuration.

4.2.1.1 Changes for JMS Adapters

On the JMS Provider page of the JMS Adapter Configuration Wizard, by default the selected option is **Third Party** for migrated projects. If the JMS provider is Oracle WebLogic JMS or Oracle Advanced Queueing, you need to change the selection to **Oracle Enterprise Messaging Service** and then select the provider to use.

For JMS adapters, you need to create a new connection factory and destinations in the Oracle WebLogic Server Administration Console, making sure to match the names specified in the JMS adapter configuration. As an alternative, you can reconfigure the adapter to use an existing connection factory and destination. For instructions on creating JMS resources in WebLogic Server, see Section 4.7, "Creating JMS Resources."

4.2.2 Adding Service Elements

WSDL supports multiple operations for a given port type. In Oracle Java CAPS, there can be one service definition for a port type, but Oracle SOA Suite requires that multiple service elements be defined for each operation. For references in the composite, there is no need for multiple reference elements for a port type with multiple operations. In Oracle SOA Suite project with binding types other than SOAP, the binding configuration in the JCA file defines multiple endpoint activation and endpoint interaction elements, one for each operation.

For Oracle Java CAPS WSDL documents that have port types with multiple operations, the migration tool populates multiple endpoint activation and endpoint interaction elements in the migrated WSDL and JCA files, but it does not define multiple service elements (one for each operation) in the generated`composite.xml` file.

For such project, you need to manually edit the generated `composite.xml` file by creating multiple service elements, one for each operation. You also need to qualify the `binding.jca` attribute of the `service` element with an `operation` attribute.

Example 4–1 Manual Changes for Port Types with Multiple Operations

For this example, the original Java CAPS binding component defined two operations for the inbound File endpoint. Below is the `service` element in the `composite.xml` file generated by the migration tool.

```
<service name="FileInboundService_FileInWSDL_InboundPort"
    ui:wsdlLocation="FileInWSDL.wsdl">
  <interface.wsdl interface="http://j2ee.netbeans.org/wsdl/FileInOut/
    FileInWSDL#wsdl.interface(FileInboundPortType)"/>
  <binding.jca
    config="FileInboundService_FileInWSDL_InboundPort_file.jca"/>
</service>
```

After making the necessary modifications, the `service` element in `composite.xml` looks like this:

```
<service name="FileInboundService_FileInWSDL_InboundPort"
    ui:wsdlLocation="FileInWSDL.wsdl">
  <interface.wsdl interface="http://j2ee.netbeans.org/wsdl/FileInOut/
    FileInWSDL#wsdl.interface(FileInboundPortType)"/>
  <binding.jca
    config="FileInboundService_FileInWSDL_InboundPort_file.jca"
    operation="pollMain"/>
</service>
<service name="FileInboundService_FileInWSDL_InboundPort"
    ui:wsdlLocation="FileInWSDL.wsdl">
   <interface.wsdl interface="http://j2ee.netbeans.org/wsdl/FileInOut/
    FileInWSDL#wsdl.interface(FileInboundPortType)"/>
```

```
  <binding.jca
    config="FileInboundService_FileInWSDL_InboundPort_file.jca"
    operation="pollBackupDir"/>
</service>
```

4.2.3 Configuring Quality of Service Properties

For Oracle Java CAPS JBI projects, you can configure certain Quality of Service properties. Not all of these properties transfer directly to Oracle SOA Suite features. Throttling and redelivery properties are not migrated by the Oracle Java CAPS Migration tool. Throttling allows you to set the maximum number of concurrent messages that are processed by a particular endpoint in order to maintain consistent performance. Redelivery settings handle message delivery when first-time delivery fails.

For throttling, you can limit the number of requests in memory for the BPEL process service engine. This affects in-only inbound requests. To set this limit, configure the `MaximumNumberOfInvokeMessagesInCache` property for the BPEL process service engine. For more information, see `"Configuring BPEL Process Service Components and Engines"` in *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

Rich redelivery can be configured using fault policies in the BPEL process service engine. For more information, see `"Using the Fault Management Framework"` in *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

4.2.4 Verifying the BPEL Structure

Since Oracle Java CAPS JBI supports some BPEL 2.0 that are not available in the Oracle SOA Suite, you need to verify the converted BPEL code for projects that use the following constructs. These are not supported in Oracle SOA Suite, but are included in the converted BPEL code.

- Compensation and termination handlers: When these exist in the original Java CAPS business process, partner links are included in the migrated BPEL as placeholders. After the migration, you need to recreate and configure the partner links.
- Standard faults: These include forcedTermination, repeatedCompensation, and invalidReply.
- Partners: A partner groups several partnerLink elements, but they were removed from BPEL 2.0 because the partner concept did not have any executable properties.

4.3 Configuring Migrated Adapters and OTDs

While the migration tool supports File, JMS, and Web Services Adapters as well as messageable OTDs, some manual steps are required to complete the migration of these components.

For JMS adapters, you need to create new destinations in the Oracle WebLogic Server Administration Console, making sure to match the names specified in the JMS adapter configuration. Make sure to verify the JMS adapter configuration using the adapter configuration wizard. For information about creating JMS destinations, see Section 4.7, "Creating JMS Resources."

The following are some updates you might need to make to your migrated JBI projects.

- Section 4.3.1, "Enabling File or JMS Message Types as Opaque"
- Section 4.3.2, "Configuring a Project with the Same XSD and FCX OTDs for Inbound and Outbound"
- Section 4.3.3, "Configuring FCX OTDs"
- Section 4.3.4, "Adding Adapters not Converted by the Migration Tool"

You should also verify the migrated adapter configuration, as described in Section 4.2.1, "Configuring Migrated Binding Components" and Section 4.2.1.1, "Changes for JMS Adapters". The information under Section 4.2.2, "Adding Service Elements" also applies to Oracle Java CAPS Repository projects.

4.3.1 Enabling File or JMS Message Types as Opaque

The Oracle Java CAPS Migration Tool converts all the File and JMS inbound and outbound adapters to use Oracle SOA Suite File and JMS adapters. Converted JMS and File adapters are configured to use the XML schema types `ewaytype:FileTextMessage` and `ewaytype:Message` in their corresponding WSDL documents. The WSDL documents are imported by the migrated adapter connections in the composite, so changes to these files affect all the connections of that adapter.

The defined `ewaytype` is an element type, which means that the input message should be wrapped inside an XML tag, as illustrated in Example 4–2. Oracle SOA Suite always requires an XML element as the input message for File and JMS endpoints that are not defined as opaque. In the converted WSDL documents, the message part is defined as a string element by default.

Example 4–2

```
<?xml version="1.0" encoding="UTF-8" ?>
<FileTextMessage xmlns=" http://xml.netbeans.org/schema/eWayTypes " >
    <<Actual messagse input>>
</FileTextMessage>
```

You can modify the WSDL document to convert the input message part from an `ewaytype` element type to an `opaque` element type in Oracle SOA Suite. If the message part is converted to opaque, Oracle SOA Suite adapters process the message as a Base64-encoded string. The input and output messages can be a simple string, similar to Oracle Java CAPS adapters, if the opaque message type is used.

To enable the message type as opaque

1. Open the converted project in Oracle JDeveloper, and then open the WSDL document you want to modify.
2. In the WSDL document, scroll to the `message` element.

   Below is an excerpt of the `types` and `message` elements. You can see the commented section describing how to define the element as opaque.

```
<types>
  <schema targetNamespace="http://xml.netbeans.org/schema/eWayTypes"
       xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="Message" type="string"/>
  </schema>
  <schema targetNamespace="http://xmlns.oracle.com/pcbpel/adapter/opaque/"
       xmlns="http://www.w3.org/2001/XMLSchema">
    <element name="Message" type="base64Binary"/>
  </schema>
</types>
```

```
<message name="Message">
  <part name="Message" element="ewaytype:Message"/>
<!-- This is an opaque message type, comment out Message part defined above
 and uncomment following Message part to use opaque message type
 <part name="Message" element="opaque:Message"/>
-->
</message>
```

3. To change the message type to opaque, comment out the line immediately above the instructional comments, and uncomment **<part name="Message" element="opaque:Message"/>**.
4. Do the following for the opaque message type before using the input string message or before writing or sending the output message:
   a. Decode Base64-encoded input using the `oracle.soa.common.util.Base64Decoder.decode()` method to get the string message.
   b. Convert the output message to Base64 string using the `oracle.soa.common.util.Base64Encoder.encode()` method.
5. For the opaque message type, do one of the following:
   - If you use the `OTDUtil.otdOperation` method to marshal or unmarshal opaque messages, set the last parameter to true as shown below:

     ```
     OTDUtil.otdOperation(otdClass,"marshalToString",inputelem,"text",true);
     ```

   - If you do not use the `OTDUtil.otdOperation` method, you need to add logic to call the encode or decode methods to convert the message before using it in the business process or Spring output.

Note: With the last parameter of `OTDUtil.otdOperation` set to true, the following occurs, depending on the OTD operation used:

- The output is converted to Base64 after a `marshalToString` operation.
- The input is decoded using `Base64Decoder` before an `unmarshalFromString` operation.

Oracle SOA Suite adapters might not work as expected if the message is not encoded to Base64 before invoking the outbound adapter.

4.3.2 Configuring a Project with the Same XSD and FCX OTDs for Inbound and Outbound

When you migrate a project that includes a JCD with the same XSD and FCX OTDs for inbound and outbound, you need to retrieve the value from the JAX-WS holder as shown below. The JAX-WS proxy generated for this type of implementation creates a JAX-WS holder encapsulating the JAXB object.

```
public void test(Holder<org.netbeans.xml.schema.synchronous.TypeA> input,
Holder<org.netbeans.xml.schema.synchronous.TypeA> output) throws Throwable {
//output.setTypeA(input.getTypeA()); ? Replace this line
output.value = input.value; ? Retrieve the value from the input Holder to output.
```

4.3.3 Configuring FCX OTDs

The migration tool automates the migration of Repository projects with FCX OTDs in a business process or in a JCD. The supported models include the following:

- A business process with FCX OTDs performing marshal and unmarshal activities.
- A business process calling a JCD with an FCX OTD as both inbound and outbound.
- A standalone JCD exposed as a web service with FCX OTDs as request and reply types.

In the above cases, the migration tool converts the FCX OTDs, which are based on XMLBeans, by replacing them with JAXB objects. The mappings in the Oracle Java CAPS project's FCX OTD are preserved in the JAXB objects.

For Oracle Java CAPS projects that include a JCD with FCX OTDs used as other OTDs, the migration tool does not convert the OTDs to JAXB objects. Use the `-xsd2jaxb` option to convert the FCX OTD used as other OTD to JAXB.

> **Note:** The converted Oracle SOA Suite project will build and deploy with no errors if you do not perform these steps; but an error will occur at runtime due to an XMLBean conflict.

To update an FCX OTD used as other OTDs

1. Run the migration tool on the project's EAR file as described in Section 3.2, "Migrating Oracle Java CAPS Projects."

   If you navigate to the converted Java code, you will see the FCX OTD used as other OTDs remains as an OTD and was not converted to JAXB.

2. Run the migration tool using the **-xsd2jaxb** option to convert the FCX OTDs to JAXB objects.

   This is described in Section 3.4, "Converting XSD to JAXB Format."

3. Replace the FCX OTD in the converted JCD code with the generated JAXB object in the converted Java code.

4.3.4 Adding Adapters not Converted by the Migration Tool

While the migration tool automatically generates code for JMS, File, and Web Service OTDs, certain adapters are not handled by the migration tool and need to be added manually to the project. The migration tool creates placeholders for these endpoints in business process projects, and the placeholders need to be replaced by the corresponding Oracle SOA Suite adapter.

> **Note:** These instructions are also published in the Spring Bean Java class file that you need to modify when adding an unconverted adapter to a migrated project in Oracle JDeveloper.

To add unconverted adapters in Spring

1. In Oracle JDeveloper, open the migrated Oracle SOA Suite project and then open its associated `composite.xml` file.
2. In the Component Palette, locate the Oracle SOA Suite equivalent adapter to the unmigrated adapter, and drag it to either the services or references swim lane.

3. Follow the steps in the configuration wizard to configure the adapter.
4. Connect the new adapter to the Spring component that was generated for the appropriate JCD (that is, the JCD that was connected to the unmigrated adapter in the original Oracle Java CAPS project).
5. In the Application Navigator, expand **Application Sources** and locate the Java class for the Spring component you just connected.

   You can find the package and class name for the Spring Bean Java class in the `bean` element of the Spring XML file. An example is shown below.

   ```
   <bean name="Consume_Message_ptt"
   class="CAPSJCDProj_JMSInFileJMSOut.Collaboration_JMSIn">
   ```

6. Open the Java class file and add a global field of a type of the adapter interface class that was generated when you connected the new adapter above. Create the accessor methods for the field.
7. Validate and then save and close the Java file.
8. In the `composite.xml`, right-click the Spring component and select **Edit**.

   The Spring XML file opens in the Oracle JDeveloper editor.
9. In the Spring XML file, locate the adapter reference that was generated when you connected the adapter and Spring component. Add the global field you created above to the `bean` element of the Spring class and refer to the adapter reference from the property.

   Note: The global adapter field created above is automatically instantiated at runtime. You can use it as if it is already instantiated.

Example 4–3 Adding an Unmigrated Adapter

As an example of the above, a database adapter is added to a migrated project with the following conditions:

- The global field added to the Java class in step 6 above is identified by the following statement:

  ```
  private xe.projectname.db.adapter.pcbpel.com.oracle.xmlns.XE_ptt database;
  ```

- Connecting the adapter to the Spring component generates the following reference in the Spring XML file:

  ```
  <reference type="xe.projectname.db.adapter.pcbpel.com.oracle.xmlns.XE_ptt"
   name="XE" xmlns="http://xmlns.oracle.com/weblogic/weblogic-sca" />
  ```

You would need to add the following property to the Spring XML file:

```
<property name="database" ref="XE" />
```

4.4 Configuring Converted Oracle SOA Suite Spring Components

The Spring Bean Java class generated by the migration tool from the original JCD code requires some modification for outbound File or JMS Adapters and for projects where a JCD calls a sub-collaboration. Perform the following steps to configure the Java class.

- Section 4.4.1, "Modifying the Spring Bean Java Class"

- Section 4.4.2, "Converting a Byte Array Input to String"
- Section 4.4.3, "Configuring the Spring Bean Class for File or JMS Outbound Adapters"
- If you need to access the JMS header properties, follow the steps in Section 4.4.4, "Accessing JMS Header Properties".
- If the original Java CAPS project has a JCD that calls a sub-collaboration, follow the steps in Section 4.4.5, "Configuring Sub-Collaborations Called from Java Collaboration Definitions".

4.4.1 Modifying the Spring Bean Java Class

Regardless of whether your project includes any of the special cases included in this section, it is important to review the migrated Java class that contains the original JCD logic. This class contains comments and notes that give you more information about changes you might need to make for your specific JCD implementation.

To modify the Spring Bean Java class

1. In Oracle JDeveloper, open the Spring component XML file.
2. Locate the `bean` element containing the name of the Bean class and note the package and class names. The element looks similar to the following:

```
<bean name="Consume_Message_ptt"
class="CAPSJCDProj_JMSInFileJMSOut.Collaboration_JMSIn"
```

3. In the left navigation panel of Oracle JDeveloper, expand **Application Sources**, and then expand the package specified by `bean` element.
4. Open the Java class file with the name specified by the `bean` element.
5. Review the code and look for comments generated by the migration tool. These are preceded by the text ***Migration Tool*** so they are easy to locate in the file.
6. When you are done making changes, save and close the file.

4.4.2 Converting a Byte Array Input to String

By default, Oracle SOA Suite receives input messages as a Byte message for inbound JMS and File Adapters. In the Oracle Java CAPS project, the input can be a String message. In this case, you need to convert the byte array to a string in the generated code.

To convert a byte input array to string

1. Locate and open the Spring Bean class as described in Section 4.4.1, "Modifying the Spring Bean Java Class."
2. In the Spring Bean Java class file, check whether the input is text. For example:

```
public void receive(com.stc.connectors.jms.Message input,
 com.stc.connector.appconn.file.FileApplication FileClient_1,
 com.stc.connectors.jms.JMS JMS_1)  throws Throwable {
        String in = input.getTextMessage();
        ...
```

3. If the original JCD receives a TextMessage as input for a File Adapter, convert the byte[] to String, as shown below:

```
String in =  new String(input.getBytesMessage());
```

4. If the original JCD receives a TextMessage as input for a JMS Adapter, set the TextMessage in the code for the `consumeMessage` method, as shown below:

```
public void consumeMessage(byte opaque[]) {
   com.stc.connectors.jms.Message input = new
        com.stc.connectors.jms.impl.MessageImpl();
   //input.setBytesMessage(opaque);
   input.setTextMessage(new String(input.getBytesMessage()));
```

5. Save and close the file.

4.4.3 Configuring the Spring Bean Class for File or JMS Outbound Adapters

For converted outbound File and JMS Adapters, locate any code in the generated Spring Bean class that uses the Oracle Java CAPS File or JMS outbound references and replace it with Oracle SOA Suite calls using the Spring Bean reference.

To configure the Spring Bean class for File or JMS outbound Adapters

1. Locate and open the Spring Bean class as described in Section 4.4.1, "Modifying the Spring Bean Java Class."
2. Open the Spring XML file and locate the Bean properties. For example:

```
<bean name="Consume_Message_ptt"
class="CAPSJCDProj_JMSInFileJMSOut.Collaboration_JMSIn">
    <property name="JMSOUT_Collaboration_JMSIn_JMS_1"
              ref="JMSOUT_Collaboration_JMSIn_JMS_1"/>
    <property name="FILEOUT_Collaboration_JMSIn_FileClient_1"
              ref="FILEOUT_Collaboration_JMSIn_FileClient_1"/>
    <property name="collabContext" ref="collabCtxBean"/>
    ....
```

Note: The above code has been wrapped for readability.

3. Make a note of the property names for the outbound adapters.
4. In the Spring Bean Java class file, locate any variables that represent the Java CAPS outbound File and JMS Adapters that are used as parameters to a method. For example:

```
public void receive(com.stc.connectors.jms.Message input,
 com.stc.connector.appconn.file.FileApplication FileClient_1,
 com.stc.connectors.jms.JMS JMS_1)  throws Throwable {
        String in = new String(input.getBytesMessage());
        System.out.println("@@ Input message " + in);
        System.out.println("@@ Sending text message to File " + in);
        FileClient_1.setText(in);
        FileClient_1.write();
        System.out.println("@@ Sending text message to JMS " + in);
        JMS_1.sendText(in);
    }
```

5. Replace the Java CAPS adapter references with the Oracle SOA Suite adapter Spring Bean properties from the Spring XML file. For example:

```
String in = new String(input.getBytesMessage());
System.out.println("@@ Input message " + in);
```

```
System.out.println("@@ Sending text message to File " + in);
this.FILEOUT_Collaboration_JMSIn_FileClient_1.write(in.getBytes());
System.out.println("@@ Sending text message to JMS " + in);
this.JMSOUT_Collaboration_JMSIn_JMS_1.produceMessage(in.getBytes());
```

6. Save and close the file.

4.4.4 Accessing JMS Header Properties

Oracle Java CAPS supports the JMS message, which encapsulates the JMS header along with the payload. In Oracle SOA Suite, the JMS header and the payload are not associated in one object, but you can define code to access the header properties.

Note: This is only supported in Oracle SOA Suite 11.1.1.6.0 and later.

To access JMS header properties from a Spring Component

1. In Oracle JDeveloper, open the Spring context that you need to modify.
2. Add the following property to the Spring Context in the appropriate `bean` element:

   ```
   <property name="headerHelper" ref="headerHelperBean"/>
   ```

3. In an editor, open the class file for the Spring Bean to which you added the property, and do the following:
   a. Add an import statement for the `IHeaderHelperBean` helper class:

      ```
      import oracle.soa.platform.component.spring.beans.IHeaderHelperBean;
      ```

   b. Declare a variable with the name **headerHelper** (the name used in the Spring context file).

      ```
      private IHeaderHelperBean headerHelper;
      ```

   c. Define the getter and setter methods for the `headerHelper` variable in the Bean class.

      Use the get and set properties from the header helper Bean. For all the custom properties use the prefix `jca.jms.JMSProperty.<PROPERTY_NAME>`:

      ```
      headerHelper.setHeaderProperty("jca.jms.JMSDestinationName",
      "queue_TMHException");
      ```

4.4.5 Configuring Sub-Collaborations Called from Java Collaboration Definitions

When an Oracle Java CAPS project contains a main Java Collaboration Definition (JCD) that calls another JCD (known as a sub-collaboration), the model is not fully migrated. The code for both JCDs is converted to the generated Oracle SOA Suite project, so you can manually update the project to complete the migration.

Only the main JCD is converted into a Spring Bean class. The sub-collaboration requires some manual conversions and wiring to the main JCD.

To migrate a sub-collaboration

1. Create a Java interface and add the method definition (that is, the main operation) of the sub-collaboration.
2. Update the sub-collaboration to implement the new interface.

3. Modify the sub-collaboration's Spring component XML file by adding the service and Bean nodes. Refer to the interface for the service node type, and refer to the sub-collaboration's Bean class for the Bean node class.
4. Modify the Spring component XML file for the main JCD by adding a reference node with the type as the Java interface you created earlier.
5. Add a Bean property and set the `ref` attribute to the reference node of the sub-collaboration.
6. Create a class variable named the same as the Bean property name you added to the main JCD's Spring component XML file.
7. In the main JCD's Bean class, define getter and setter methods for the Bean property defined above.

   In the Oracle SOA Suite project's `composite.xml` file, the Spring components for both the main JCD and the sub-collaboration have new endpoints exposed after you make the above changes.
8. Wire the main JCD's Spring component to the sub-collaboration's Spring component by linking the two endpoints.
9. Modify the main JCD's Spring component XML file to replace the reference to the JCD invoking the sub-collaboration with the Spring Bean reference for the sub-collaboration.
10. Save the project.

4.5 Configuring Business Processes

Following are some updates you might need to make for migration Repository projects with business processes:

- Section 4.5.1, "Migrating User Activities in Business Processes"
- Section 4.5.2, "Migrating Correlation Initialization in Marshal and Unmarshal Activities"

4.5.1 Migrating User Activities in Business Processes

The migration tool cannot migrate business processes with user activities (Worklist Manager). The original business process contains BPEL constructs specific to Oracle Java CAPS, which need to be manually replaced with Oracle SOA Suite human workflow services. For more information about using Oracle SOA Suite human workflow services, see `"Getting Started with Human Workflow"` in Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite.

4.5.2 Migrating Correlation Initialization in Marshal and Unmarshal Activities

When correlation is initialized in the marshal and unmarshal (Invoke) activities in a business process, the initialization is ignored in the migrated business process because the marshal and unmarshal activities are replaced with embedded Java activities.

To address this, move the correlation initialization logic outside the marshal and unmarshal activities before migrating a project with this condition. Replace the correlation property alias definition in the WSDL document with an element attribute instead of a messagetype attribute by doing the following:

- Change the `messageType` attribute to its corresponding element attribute

- Remove the part attribute in the `propertyAlias` definition

Below is an example of the code before and after the required changes.

```
<message name="ReserveVehicleIn">
<part name="itinerary" element="ota:TravelItinerary"/>
</message>

<bpws:propertyAlias
propertyName="tres:ItineraryRefId"
messageType="vres:ReserveVehicleIn"
part="itinerary">
<bpws:query>/ota:TravelItinerary/ota:ItineraryRef/ota:UniqueID</bpws:query>
</bpws:propertyAlias>
```

Change the propertyAlias definition as shown below:

```
<bpws:propertyAlias
propertyName="tres:ItineraryRefId"
element="ota:TravelItinerary">
<bpws:query>/ota:TravelItinerary/ota:ItineraryRef/ota:UniqueID</bpws:query>
</bpws:propertyAlias>
```

4.6 Adding JAR Files to a Migrated Project

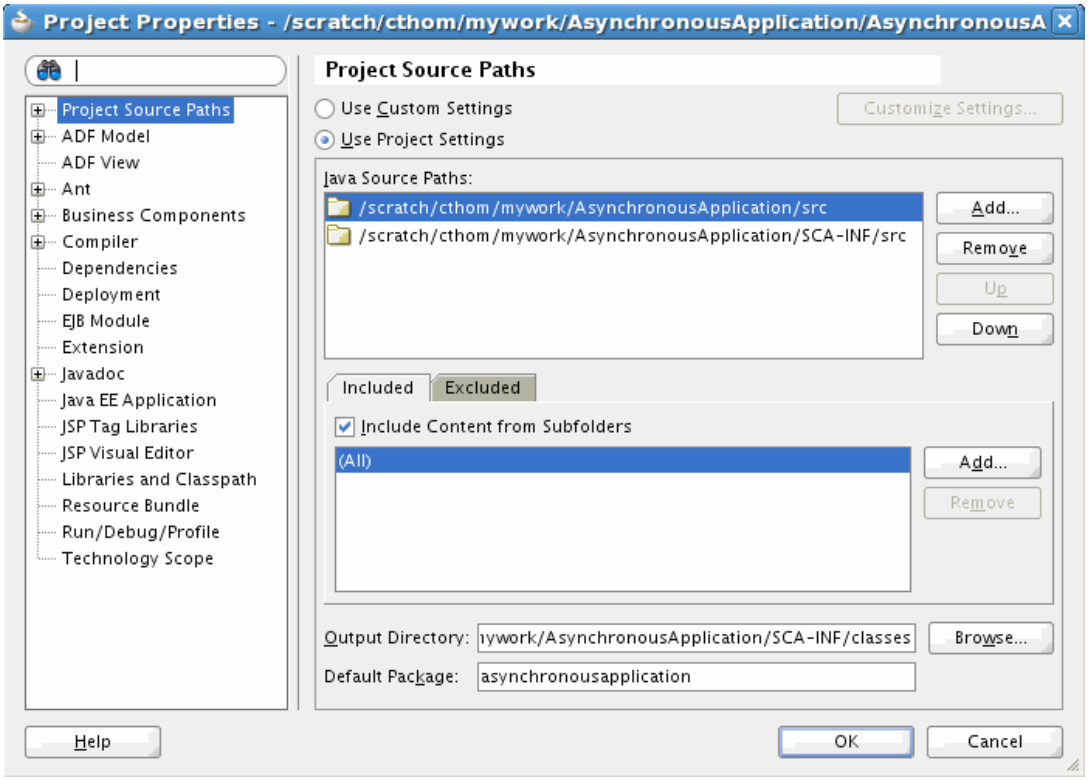
Some Repository projects that include JCDs use additional JAR files that are specific to Oracle Java CAPS. These files are also required in the migrated Oracle SOA Suite project, and are added to the generated project in the `/SCA-INF/lib` directory (since they are packaged in the source EAR file). Most of the JAR files are automatically added to the JDeveloper project file by the Migration Tool. If there are additional JAR files that need to be added to the project, you can load them into Oracle JDeveloper.

To add a JAR file to a migrated project

1. Open the migrated project in Oracle JDeveloper.
2. In the Application Navigator, right-click the project name, and select **Project Properties**.

   The Project Properties editor appears.

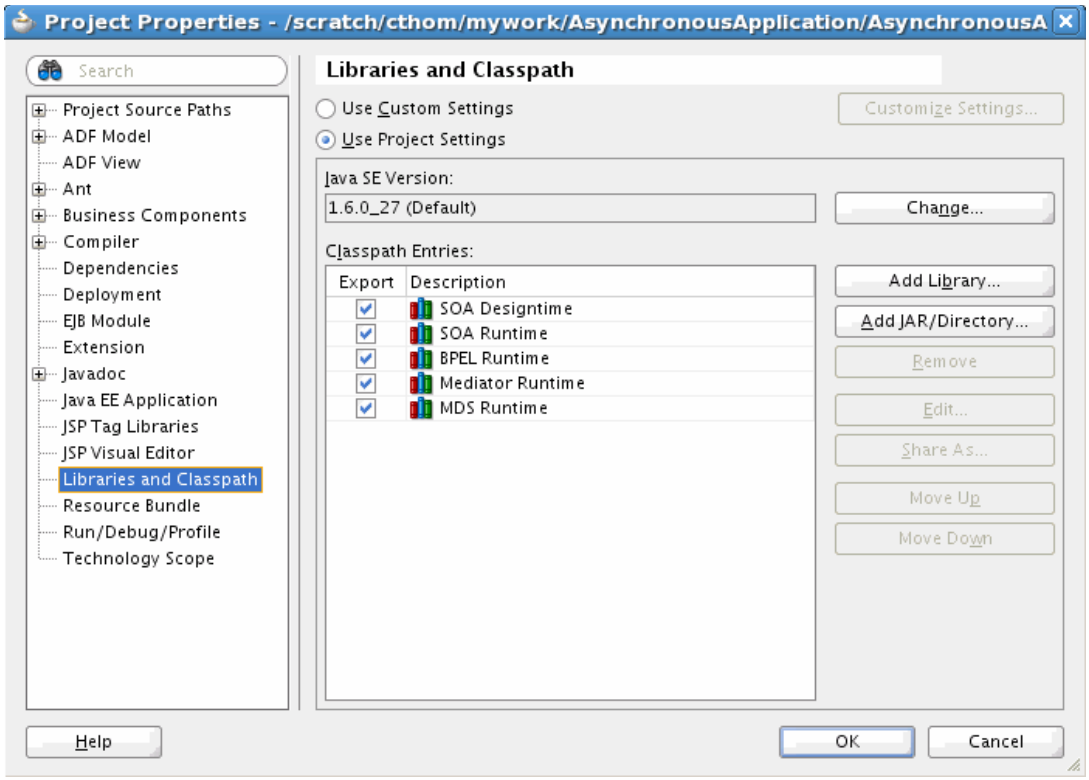
Figure 4–2 Project Properties - Project Source Paths



3. In the left navigation pane, select **Libraries and Classpath**.

   The Libraries and Classpath page appears.

Figure 4–3 Project Properties - Libraries and Classpath



4. Click **Add JAR/Directory**.
5. Browse to and select the JAR file to add, and click **Select**.

   The JAR file is added to the Classpath Entries list.
6. Repeat the previous two steps for any additional JAR files you need to add.

4.7 Creating JMS Resources

The steps required to create JMS resources vary depending on whether you are using Oracle WebLogic JMS as the message provider or third-party message providers. You can use existing JMS modules and servers, or add new ones if needed. If you are using a JMS server other than WebLogic, you may need to create and configure a foreign server, and add connection factories to the new server.

General instructions for adding a destination to a JMS module are provided below. For specific information and instructions on creating JMS resources using the Oracle WebLogic Server Administration Console, see *Oracle Fusion Middleware Configuring and Managing JMS for Oracle WebLogic Server*.

To add destinations to the JMS Module

1. In a Web browser, launch the Oracle WebLogic Server Administration Console:

   `http://hostname:port_number/console`
2. Under Services, select **JMS Modules**.

   The JMS Modules page appears.

3. In the JMS Modules table, select the module to which you want to add the destination.

   The Settings page for the module you selected appears.

4. In the Summary of Resources table, click **New**.

   The Create a New JMS System Module Resource page appears.

5. Select **Queue** or **Topic**, and then click **Next**.

6. In the **Name** field, enter a name for this topic or queue. This is a logical name that is referenced by Oracle WebLogic Server.

7. In the **JNDI Name** field, enter the local JNDI name that you will use in your application to look up this destination

8. Click **Next**.

9. To use a subdeployment for the destination, do the following:

   a. Select an existing subdeployment from the **Subdeployments** field or click **Create a New Subdeployment** to create a new one.

   b. Select the target server for the subdeployment.

10. Click **Finish**.

    The new destination is added to the Summary of Resources table.

11. To activate these changes, in the Change Center of the Administration Console, click **Activate Changes**.

12. Restart the server.

A

Examples of Java Collaboration Definition Conversions

This appendix provides code samples that illustrate how a stand-alone Java Collaboration Definition (JCD) is converted to SOA Suite and how a JCD called from a business process is converted.

This appendix includes the following topics:

- Section A.1, "Sample Code for Migrating a Stand-Alone JCD"
- Section A.2, "Sample Code for Migrating a JCD Called from a Business Process"

For more information about migrating JCDs, see Section 1.2.2.1, "Support for Java Collaboration Definition Migration."

A.1 Sample Code for Migrating a Stand-Alone JCD

The following example illustrates how a JCD with a File inbound connection and both File and JMS outbound connections is converted to a SOA Suite Spring Bean class.

Sample Source JCD Code for Stand-Alone JCD

```
package CAPSProject1;
public class Collaboration_1
{
   public com.stc.codegen.logger.Logger logger;
   public com.stc.codegen.alerter.Alerter alerter;
   public com.stc.codegen.util.CollaborationContext collabContext;
   public com.stc.codegen.util.TypeConverter typeConverter;

   public void receive( com.stc.connector.appconn.file.FileTextMessage input,
   com.stc.connector.appconn.file.FileApplication FileClient_1,
   com.stc.connectors.jms.JMS JMS_1,employees.EmployeesOTD Employees_1 )
   throws Throwable
   {
     logger.info( "inside receive." );
     com.stc.connectors.jms.Message msg =
               JMS_1.createTextMessage( input.getText() );
     employees.EMPLOYEES emp = Employees_1.getEMPLOYEES();
     logger.info( emp.getLAST_NAME() );
     JMS_1.send( msg );
     FileClient_1.setText( input.getText() );
     FileClient_1.write();
   }
}
```

Sample Output Generated for SOA Suite for a Stand-Alone JCD

```
package CAPSProject1;

public class Collaboration_1 implements
FILEIN_Collaboration_1_input.capsproject1.file.adapter.pcbpel.com.oracle.xmlns.Rea
d_ptt {

  private JMSOUT_Collaboration_1_JMS_
    1.capsproject1.jms.adapter.pcbpel.com.oracle.xmlns.Produce_Message_ptt
    JMSOUT_Collaboration_1_JMS_1;
  private FILEOUT_Collaboration_1_FileClient_
    1.capsproject1.file.adapter.pcbpel.com.oracle.xmlns.Write_ptt
    FILEOUT_Collaboration_1_FileClient_1;

  public void read(byte opaque[]) {
    com.stc.connector.appconn.file.FileTextMessage input = new
      com.stc.connector.appconn.file.FileTextMessageImpl();
    input.setByteArray(opaque);
    com.stc.connector.appconn.file.FileApplication FileClient_1 = new
      com.stc.connector.appconn.file.FileApplication();
    com.stc.connectors.jms.JMS JMS_1 = new com.stc.connectors.jms.JMS();
    employees.EmployeesOTD Employees_1 = new employees.EmployeesOTD();
    try {
       this.receive(input, FileClient_1, JMS_1, Employees_1);
    } catch (Throwable exp_0) {
       throw new java.lang.RuntimeException(exp_0);
    } finally {
    }

  //JMSOUT_Collaboration_1_JMS_1.produceMessage(opaque);
  //FILEOUT_Collaboration_1_FileClient_1.write(opaque);
  }

  public com.stc.codegen.logger.Logger logger;
  public com.stc.codegen.alerter.Alerter alerter;
  public com.stc.codegen.util.CollaborationContext collabContext;
  public com.stc.codegen.util.TypeConverter typeConverter;
  public void receive(com.stc.connector.appconn.file.FileTextMessage input,
      com.stc.connector.appconn.file.FileApplication
      FileClient_1, com.stc.connectors.jms.JMS JMS_1, employees.EmployeesOTD
      Employees_1) throws Throwable {
    logger.info("inside receive.");
    com.stc.connectors.jms.Message msg = JMS_1.createTextMessage(input.getText());
    employees.EMPLOYEES emp = Employees_1.getEMPLOYEES();
    logger.info(emp.getLAST_NAME());
    JMS_1.send(msg);
    FileClient_1.setText(input.getText());
    FileClient_1.write();
  }

  public final void setJMSOUT_Collaboration_1_JMS_1(JMSOUT_Collaboration_1_JMS_
     1.capsproject1.jms.adapter.pcbpel.com.oracle.xmlns.Produce_Message_ptt
     JMSOUT_Collaboration_1_JMS_1) {
    this.JMSOUT_Collaboration_1_JMS_1 = JMSOUT_Collaboration_1_JMS_1;
  }

  public final JMSOUT_Collaboration_1_JMS_1.capsproject1.jms.adapter.pcbpel.com.
     oracle.xmlns.Produce_Message_ptt getJMSOUT_Collaboration_1_JMS_1() {
     return this.JMSOUT_Collaboration_1_JMS_1;
  }
```

```
  public final void setFILEOUT_Collaboration_1_FileClient_1(FILEOUT_Collaboration_
     1_FileClient_1.capsproject1.file.adapter.pcbpel.com.
     oracle.xmlns.Write_ptt FILEOUT_Collaboration_1_FileClient_1) {
    this.FILEOUT_Collaboration_1_FileClient_1 = FILEOUT_Collaboration_1_
     FileClient_1;
  }

  public final FILEOUT_Collaboration_1_FileClient_1.capsproject1.file.adapter.
     pcbpel.com.oracle.xmlns.Write_ptt getFILEOUT_Collaboration_1_FileClient_1() {
    return this.FILEOUT_Collaboration_1_FileClient_1;
  }

  public final void setLogger(com.stc.codegen.logger.Logger logger) {
    this.logger = logger;
  }

  public final com.stc.codegen.logger.Logger getLogger() {
    return this.logger;
  }

  public final void setAlerter(com.stc.codegen.alerter.Alerter alerter) {
    this.alerter = alerter;
  }

  public final com.stc.codegen.alerter.Alerter getAlerter() {
    return this.alerter;
  }

  public final void setCollabContext(com.stc.codegen.util.CollaborationContext
     collabContext) {
    this.collabContext = collabContext;
  }

  public final com.stc.codegen.util.CollaborationContext getCollabContext() {
   return this.collabContext;
  }

  public final void setTypeConverter(com.stc.codegen.util.TypeConverter
     typeConverter) {
    this.typeConverter = typeConverter;
  }

  public final com.stc.codegen.util.TypeConverter getTypeConverter() {
    return this.typeConverter;
  }
}
```

You can see in the above samples that the migrated JCD class now implements the SOA Suite File Adapter interface, and the implemented interface method `public void read(byte opaque[])` replaces the following method from the original JCD:

```
public void receive(com.stc.connector.appconn.file.FileTextMessage input,
 com.stc.connector.appconn.file.FileApplication FileClient_1,
 com.stc.connectors.jms.JMS JMS_1, employees.EmployeesOTD Employees_1)
```

This is the triggering method when an input file is present in the input directory. The configuration of the SOA Suite File Adapter is defined in a JCA file generated by the migration tool. You can modify the configuration in Oracle JDeveloper.

The interface method invokes the original JCD method. The following two class fields represent the outbound SOA Suite File and JMS adapters:

```
private
JMSOUT_Collaboration_1_JMS_1.capsproject1.jms.adapter.pcbpel.com.oracle.xmlns.Prod
uce_Message_ptt JMSOUT_Collaboration_1_JMS_1;
private
FILEOUT_Collaboration_1_FileClient_1.capsproject1.file.adapter.pcbpel.com.oracle.x
mlns.Write_ptt FILEOUT_Collaboration_1_FileClient_1;
```

The above two class fields, as well as the following class fields in the original JCD, are not instantiated, but getter and setter methods are generated for them. This allows them to be instantiated using SOA Suite's context injection.

```
public com.stc.codegen.logger.Logger logger;
public com.stc.codegen.alerter.Alerter alerter;
public com.stc.codegen.util.CollaborationContext collabContext;
public com.stc.codegen.util.TypeConverter typeConverter;
```

A.2 Sample Code for Migrating a JCD Called from a Business Process

The following example illustrates how a JCD that is invoked from a business process is converted to a SOA Suite Spring Bean class.

Sample JCD Source Code for a JCD Invoked from a Business Process

```
package Jcd_Bpel_Project;

public class Collaboration_3
{
  public com.stc.codegen.logger.Logger logger;
  public com.stc.codegen.alerter.Alerter alerter;
  public com.stc.codegen.util.CollaborationContext collabContext;
  public com.stc.codegen.util.TypeConverter typeConverter;
  public void op1_vikas( dtd.otdInputDTD_2111422138.DBemployee input,
     dtd.otdOutputDTD_1854792262.DBemployee output )
    throws Throwable
  {
    output.setEmpNo( input.getEmpNo() );
    output.setLastname( input.getLastname() );
    output.setFirstname( input.getFirstname() );
    output.setRate( input.getRate() );
    output.setLastDate( input.getLastDate() );
    Thread.sleep( 120000 );
  }
}
```

Sample Output Code Generated for a JCD Invoked from a Business Process

```
package Jcd_Bpel_Project;

import oracle.migrationtool.migration.caps.runtime.OTDUtil;

public class Collaboration_3 implements
urn_stc_egate_jce_collaboration_1wsdl.ExecutePortType {
  private stc.egate.otd.dtd.otdoutputdtd_1854792262.DBemployee output;

  public stc.egate.otd.dtd.otdoutputdtd_1854792262.DBemployee
     op1Vikas(stc.egate.otd.dtd.otdinputdtd_2111422138.DBemployee DBemployee)
```

```
   throws urn_stc_egate_jce_collaboration_1wsdl.JavaExceptionMessage {
  this.output = new stc.egate.otd.dtd.otdoutputdtd_1854792262.DBemployee();
  try {
    this.op1_vikas(DBemployee, this.output);
  } catch (Throwable exp_0) {
    throw new urn_stc_egate_jce_collaboration_1wsdl.JavaExceptionMessage
      (exp_0.getMessage(), null, exp_0);
  } finally {
  }
  return this.output;
}

public com.stc.codegen.logger.Logger logger;
public com.stc.codegen.alerter.Alerter alerter;
public com.stc.codegen.util.CollaborationContext collabContext;
public com.stc.codegen.util.TypeConverter typeConverter;

public void op1_vikas(stc.egate.otd.dtd.otdinputdtd_2111422138.DBemployee input,
 stc.egate.otd.dtd.otdoutputdtd_1854792262.DBemployee output) throws Throwable {
  output.setEmpNo(input.getEmpNo());
  output.setLastname(input.getLastname());
  output.setFirstname(input.getFirstname());
  output.setRate(input.getRate());
  output.setLastDate(input.getLastDate());
  Thread.sleep(120000);
}

public final void setOutput(stc.egate.otd.dtd.otdoutputdtd_1854792262.DBemployee
   output) {
  this.output = output;
}

public final stc.egate.otd.dtd.otdoutputdtd_1854792262.DBemployee getOutput() {
  return this.output;
}

public final void setLogger(com.stc.codegen.logger.Logger logger) {
  this.logger = logger;
}

public final com.stc.codegen.logger.Logger getLogger() {
  return this.logger;
}

public final void setAlerter(com.stc.codegen.alerter.Alerter alerter) {
  this.alerter = alerter;
}

public final com.stc.codegen.alerter.Alerter getAlerter() {
  return this.alerter;
}

public final void setCollabContext(com.stc.codegen.util.CollaborationContext
   collabContext) {
  this.collabContext = collabContext;
}

public final com.stc.codegen.util.CollaborationContext getCollabContext() {
  return this.collabContext;
}
```

```
  public final void setTypeConverter(com.stc.codegen.util.TypeConverter
     typeConverter) {
    this.typeConverter = typeConverter;
  }

  public final com.stc.codegen.util.TypeConverter getTypeConverter() {
    return this.typeConverter;
  }
}
```

The converted class implements a JAXB proxy interface and the original OTD classes of the JCD are replaced by JAXB objects.Getters and setters are generated for the Spring context injection.

Most comments in the original JCD are preserved in the migrated SOA Spring Bean class. Comments without a following Java statement are not preserved during the migration. To ensure comments are preserved, add a semicolon (;) after comments without a following Java statement in the original JCD so that the comment can be preserved.

Glossary

Adapter (also *eWay Adapter*)

A link between a Java Collaboration Definition or business process and an external connection, including the message server connection (topic or queue) or external application. Adapters are specific to Repository projects, and provide a similar functionality as Oracle Java CAPS JBI Binding Components and Oracle SOA Suite Adapters.

Document Object Model (DOM)

A language-independent standard for accessing and manipulating objects in XML documents from the programming language being used.

Document Type Definition (DTD)

A set of declarations that define the structure for an XML document (a precursor to XML schema).

Enterprise Archive (EAR)

A file format for packaging Java CAPS Repository modules into a single archive that can be deployed to an application server.

First Class (FCX) OTD

An XMLBean based Object Type Definition (OTD).

Java Business Integration (JBI)

A specification for implementing a service-oriented architecture (SOA), which includes web services, a container for service producers and consumers, binding components to define connectivity, service engines, and a normalized message router.

Java Collaboration Definition (JCD)

Business rules and logic written in Java format in a Java CAPS Repository project. Typically, the encoding consists of operations on an Object Type Definition.

JAXB

A Java API for mapping Java classes to XML representations. Java Architecture for XML Binding (JAXB) is part of the Java EE platform.

JAX-WS

A Java API for creating web services. Java API for XML Web Services (JAX-WS) is part of the Java EE platform.

JBI Projects

Oracle Java CAPS projects based on JBI technologies. These projects use service engines and binding components to define business logic and connectivity. The source code is stored in a file system.

Marshal

An OTD operation that serializes an XML structure or object to String, Bytes, or Stream format.

Messageable OTD

An OTD with marshal and unmarshal methods (serializable objects). These OTDs are not associated with any adapters, but are used to define the data structure for a specific message format.

Object Type Definition (OTD)

The data structure and rules that define an object. OTDs define the API used to map data and external systems into canonical objects that can be used in mappings throughout Oracle Java CAPS.

Repository Projects

Oracle Java CAPS projects based on Oracle Java CAPS 5.1.3 technology. These projects use Object Type Definitions, Java Collaboration Definitions, BPEL 1.0 business processes, and eWay Adapters to define business logic and connectivity. The source code is stored in an encoded repository.

User-Defined OTD (UD OTD)

A custom OTD that you create and configure manually or from a flat file. You can add or delete nodes in a user-defined OTD, and edit their properties.

Unmarshal

An OTD operation that deserializes a message in String, Stream, or Bytes format to an OTD object.