Oracle Fusion Cloud Applications

Common Features Reference

F30652-04
November 2022

Oracle Fusion Cloud Applications Common Features Reference,

F30652-04

Primary Author: Oracle Corporation

Contents

Preface

1 Reference Topics

Preface

This document lists some of the common feature topics that other Oracle Fusion Cloud Applications guides link or refer to.

Audience

This document is intended for the users of Oracle Fusion Middleware and Oracle Fusion Applications who are looking for sections that link or refer to Middleware or Fusion applications documentation.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

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.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/us/corporate/accessibility/index.html`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/us/corporate/`

accessibility/support/index.html#info or visit http://www.oracle.com/us/corporate/accessibility/support/index.html#trs if you are hearing impaired.

1
Reference Topics

This chapter provides content from Oracle Fusion Middleware that supplements Oracle Fusion Cloud Applications guides. This collection of diverse topics is just for reference and not meant to be read in any particular order.

Auditing Web Services

Auditing describes the process of collecting and storing information about security events and the outcome of those events. An audit provides an electronic trail of selected system activity.

An audit *policy* defines the type and scope of events to be captured at run time. Although a very large array of system and user events can occur during an operation, the events that are actually audited depend on the audit policies in effect at run time. You can define component- or application-specific policies, or audit individual users.

You configure auditing for system components, including web services, and applications at the domain level using the Audit Policy page. You can audit SOA and ADF services.

The following table summarizes the events that you can audit for web services and the relevant component.

Table 1-1 Auditing Events for Web Services

Enable auditing for the following web service events. . .	Using this system component. . .
• User authentication. • User authorization. • Policy enforcement, including message confidentiality, message integrity, and security policy.	OWSM—Agent
• Web service requests sent and responses received. • SOAP faults incurred. **Note**: In this case, events are logged for both security and non-security web service invocations.	Oracle web services
• OWSM assertion template creation, deletion, or modification. • OWSM policy intent creation, deletion, or modification. • OWSM policy creation, deletion, or modification. • OWSM policy set authoring creation, deletion, or modification.	OWSM—Policy Manager **Note:** The Policy Manager audits both local policy attachments and global policy attachments for policy sets.
• OWSM policy attachment.	OWSM—Policy Attachment **Note:** The Policy Attachment audits only local policy attachments.

You can also audit the events for a specific user, for example, you can audit all events by an administrator.

For more information about configuring audit policies, see Managing Audit.

The following sections describe how to define audit policies and view audit data:

Configuring Audit Policies

Follow the steps in this section to configure audit policies. For more information, see *Manage Audit Policies for Java Components with Fusion Middleware Control* in *Securing Applications with Oracle Platform Security Services*.

1. From the WebLogic Domain menu, select **Security > Audit Policy**.

   The Audit Policy Settings page is displayed.

   The audit policies table, at the center of the page, displays the audits that are currently in effect.

2. Select the component that you want to audit from the Audit Component Name menu.

3. Select an audit level from the Audit Level menu.

   Valid audit levels include:

   - None—Disables auditing.
   - Low, Medium, High—Audits subsets of event categories representing pre-defined levels of auditing.
   - Custom—Enables you to provide a custom auditing policy.

   You can view the components and applications that are selected for audit at each level in the audit policies list. For all audit levels other than Custom, the information in the audit policies list is greyed out, as you cannot customize other audit level settings.

4. To customize the audit policy, select the Custom option and perform one of the following steps:

   - Select the information that you want to audit by clicking the associated checkbox in the Select for Audit column.

     You can audit at the following levels of granularity: All events for a component, all events within a component event category, an individual event, or a specific outcome of an individual event (such as, success or failure).

     Click **Select All** to select all categories, **None** to deselect all categories, or **Audit All Events** to audit all events, including specific outcome of individual events (such as, successes and failures).

     At the event outcome level, you can specify an edit filter. Filters are rules-based expressions that you can define to control the events that are returned. For example, you might specify an Initiator as a filter for policy management operations to track when policies were created, modified, or deleted by a specific user. To define a filter for an outcome level, click the **Edit Filter** icon in the appropriate column, specify the filter attributes, and click **OK**. The filter definition appears in the Filter column.

Deselect the checkbox for a component at a higher level to customize auditing for its subcomponents. You can select all components and applications by checking the checkbox adjacent to the column name.

- At the event outcome level, you can specify an edit filter. Filters are rules-based expressions that you can define to control the events that are returned. For example, you might specify an Initiator as a filter for policy management operations to track when policies were created, modified, or deleted by a specific user. To define a filter for an outcome level, click the **Edit Filter** icon in the appropriate column, specify the filter attributes, and click **OK**. The filter definition appears in the Filter column.
- To audit only success or failures for all system components and applications, select **Select Successes Only** or **Select Failures Only** from the Select menu, respectively. To clear all selections, select **None**.

5. If required, enter a comma-separated list of users in the Users to Always Audit text box.

   Specified users will always be audited, regardless of whether auditing is enabled or disabled, and at what level auditing is set.

6. Click **Apply.**

   To revert all changes made during the current session, click **Revert**.

Managing Audit Data Collection and Storage

To manage the data collection and storage of audit information, you need to perform the following tasks:

- Set up and manage an audit data repository.

  You can store records using one of two repository modes: file and database. It is recommended that you use the database repository mode. The Oracle Business Intelligence Publisher-based audit reports only work in the database repository mode.
- Set up audit event collection.

For more information, see *Managing the Audit Data Store* in *Securing Applications with Oracle Platform Security Services*.

Viewing Audit Reports

For database repositories, data is exposed through pre-defined reports in Oracle Business Intelligence Publisher.

A number of predefined reports are available, such as: authentication and authorization history, OWSM policy enforcement and management, and so on. For details about generating and viewing audit reports using Oracle Business Intelligence Publisher, see *Using Audit Analysis and Reporting* in *Securing Applications with Oracle Platform Security Services*.

For file-based repositories, you can view the bus-stop files using a text editor and create your own custom queries.

Introduction to Oracle Fusion Middleware Audit Framework

The Oracle Fusion Middleware Audit Framework allows you to audit application events. Using this framework, you create events specific to your application, register the application at deployment, and generate audit reports.

What Are the Audit Objectives?

The objectives of audit are to comply with regulations, to monitor business activity, and to obtain data for risk analysis.

Compliance

To comply with regulations required in the enterprise and to allow the review of compliance policies, customers must audit identity information and user access events on applications and devices across the enterprise, including the following:

- User profile change
- Access rights change
- User access
- Operational activities, such as like application start and stop, upgrade, and backup

Monitoring

Audit data allows you to monitor activity, to create dashboards, and to build key performance indicators to observe the health of the various systems in the enterprise.

Analytics

Audit data analysis can be used to assess the efficacy of controls and risks. Based on historical data, a risk score is calculated and assigned to a user. Then, any runtime evaluation of a user access to systems can include risk scores as additional criteria to determine access permission.

Audit support across enterprises is not uniform. For example, there are no standards to generate audit records, format records, or define audit policies. As a result, audit solutions have a number of drawbacks:

- There is no centralized audit framework.
- Audit support is inconsistent from application to application.
- Audit data, audit policies, and configuration are scattered across the enterprise.
- Cross-component analysis of audit is complex and time-consuming.
- Scattered data, lack of consistency, and decentralization make the audit solutions fragile with idiosyncrasies.

Audit Terminology

This section introduces several audit terms used in this document.

Component

A *component* refers to an Oracle Fusion Middleware component.

Audit-Aware Components

An *audit-aware component* is a component that is integrated with Audit Framework, whose audit policies can be configured and whose events can be audited.

Audit Store

The *audit store* is a database that has a predefined audit schema and that stores audit events. After you configure the audit store, the audit loader periodically uploads data to this database. Audit data is cumulative and grows in size over time. Ideally, the audit store should be a database not used by other applications but used exclusively by audit. The audit store stores audit events generated by components as well as user applications integrated with Audit Framework.

Audit Definition File

An *audit definition file* is a file where an applications specify its specific audit rules (such as events and filters) that control audit.

Audit Events

An *audit event* is an event that is recorded by Audit Framework. This framework provides a set of generic events that you map to application audit common events, such as authentication or policy change. It also allows you to define specific application events and to update audit configuration with Fusion Middleware Control or with WebLogic Scripting Tool (WLST) commands.

Audit Loader

The *audit loader* is a module of Oracle WebLogic Server that supports audit activity in the server. After you configured the audit store, the audit loader collects audit records of all running components and loads them to the audit store. For Java components, the audit loader starts when the container starts up. To upload events with the audit loader, register the system component with audit (with the `registerAudit` WLST command) or use the standalone audit loader.

Audit Policy

An *audit policy* specifies the events that Audit Framework captures for a particular component. You define policies at the component level (so that it applies to a particular component), or at the domain level (so that it applies to all components in the domain).

Bus-Stop Files

A *bus-stop file* is a local file that contains audit data records. Bus-stop files are simple text files that can be queried easily to look up specific audit events. If audit is configured in the domain, then the data in these files is periodically uploaded to the audit store after a configurable time interval. If audit is not configured in the domain, then the data is kept in bus-stop files.

You correlate and combine audit data from multiple components in a report, for example, when you want to identify authentication failures in all middleware components and instances.

By default, the bus-stop files are located the following directory:

```
Weblogic Domain Home/servers/server_name/logs/auditlogs
```

with sub-directories for each component bus-stop files. For example, OPSS bus-stop files are kept in the following directory:

```
Weblogic Domain Home/servers/server_name/logs/auditlogs/JPS
```

Event Filters

An *event filter* is a filter that controls whether the event is logged. For example, a successful login event to a component is logged only for a certain subset of users.

Audit Configuration MBeans

Audit configuration MBeans are the MBeans that manage audit configuration. For Java components and applications, these MBeans are present in WebLogic Administration Server and the audit configuration is centrally managed. For system components, each component has its separate MBeans.

About Auditing with Oracle Fusion Middleware Audit Framework

This sections describes the Audit Framework support to audit components.

Overview of Oracle Fusion Middleware Audit Framework

The Audit Framework includes the following features:

- A uniform way to administer audits across Java components, system components, and applications.
- A Java component audit, including:
  - Support audit for applications that are not audit-aware.
  - The ability to search for audit data at any application level.
- Capturing authentication history and failures, authorization history, user management, and other common transaction data.
- Flexible policies including:
  - Previously seeded audit policies, which capture most common audit events, available for ease of configuration.
  - A tree-like policy structure.
- The ability to write your own reports based on the published audit schema.
- Keeping audit data and files in a common location (the audit store), which simplifies record maintenance.
- A common audit record format including:
  - Baseline attributes such as outcome (status), event date-time, and user.
  - Event-specific attributes such as the authentication method, source IP address, target user, and resource.
  - Contextual attributes such as the execution context ID (ECID), and session ID.
- A common and unified way to configure audit policies for the entire domain.
- Oracle Fusion Middleware support, so that audit:
  - Can be used across Oracle Fusion Middleware components and services.
  - Integrates with Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control).
  - Integrates with WLST.

- A dynamic metadata model that integrates with the Audit Framework and that allows applications to:
  - Register at any time.
  - Define and log specific audit events.
  - Upgrade definitions independent of release cycles by providing event definitions versions.

About Components and Applications

Oracle Fusion Middleware Audit Framework provides a centralized framework for all Oracle Fusion Middleware products. Specifically, it provides audit for the following applications and components:

- Middleware Platform - This includes Java components such as OPSS and Oracle Web Services Manager. All the deployed applications leveraging Java components benefit from audit, which happens at the platform level.
- Java EE applications - The framework provides audit for Java EE applications, including Oracle Java EE-based components, and applications and components can specify their own specific audit events.
- System Components - For system components, such as Oracle HTTP Server, the framework provides an end-to-end solution similar to that of Java components, including APIs for C and C++ applications.

> **See also:**
>
> *Oracle Fusion Middleware Components* in *Administering Oracle Fusion Middleware*.

Understanding Audit

This section explains fundamental audit concepts.

The Audit Model

The audit model provides a standards-based, integrated framework for Java EE and SE applications and components across Oracle Fusion Middleware.

Dynamic Model

The Oracle Fusion Middleware Audit Framework features a dynamic audit model that lets applications manage audit event definitions and make version changes independent of release cycles. Audit event definitions can be dynamically updated at redeployment.

Application Life Cycle Support

The model supports all aspects of the application life cycle from design to development to deployment.

Application Registration

A versatile registration lets you register applications with audit in different ways:

- Declaratively, by packaging the configuration in the `META-INF` directory of the application Enterprise ARchive (EAR) file.
- Programmatically, by calling the audit registration methods.
- At the command line, by calling WLST audit commands.
- When you create a domain, by specifying security artifacts in a product template.

Distributed Environments

Oracle Fusion Middleware Audit Framework supports distributed environments with multiple servers. It monitors the audit store so changes in audit policies introduced in one server are synchronized with all other servers in the domain.

Consider, for example, a distributed environment consisting of an Administration Server and three Managed Servers. A single security store (that includes audit data) supports all the servers in the domain. When you change an audit policy in the Administration Server with Fusion Middleware Control, then those changes are automatically propagated and synchronized with all other servers in the domain.

About the Audit Store

The audit store contains component event definitions, attribute table mappings, and audit policies.

The audit store includes:

- Audit registration that allows you:
  - Create, modify, and delete event definition entries.
  - Create attribute database mappings to store audit data.
- The service that retrieves event definitions and runtime policies.
- Audit MBean commands that allow you to look up and modify component audit definitions and runtime policies.

The Audit Framework requires a database to store audit data, and this database can be any of the supported ones.

When a new application registers with audit, the following artifacts are stored in the audit store:

- Audit event definitions including custom attribute group, categories, events, and filter preset definitions
- Localized translation entries
- Custom attribute-database column mapping tables
- Runtime audit policies

How Audit Data Is Stored

Audit data resides in intermediate or permanent storages.

- Intermediate storage, in bus-stop files. Each component instance writes to its own separate bus-stop file. Bus-stop files are text-based and easy to query.
- Permanent storage, in the audit store (if configured in the environment). Audit records generated by all components in the domain are written to the same store.

Advantages to Using a Database Store

Having the audit records stored in bus-stop files has some limitations:

- You cannot view domain-level audit data.
- You cannot obtain reports easily.

And there are advantages to using the audit store:

- It allows you to generate audit reports.
- The database store contains records from all components in the domain, whereas the bus-stop contains audit records for one component only.
- It improves performance.

About the Oracle Fusion Middleware Audit Framework

The Audit Framework provides a set of interfaces for any audit-aware components integrating with it. During runtime, applications may call these APIs to manage audit policies and to audit the necessary information about a particular event happening in the application code. These interfaces allow applications to specify event details and attributes needed to provide the context of the event they want to audit.

Audit Setup: Main Steps

The following list includes the major tasks that you carry out to you set up and maintain audit in your environment:

- Understanding the audit architecture, the essential elements of the framework, the flow of actions, and the Audit Framework. For information about these tasks, see Audit Administration Tasks.
- Integrating applications with the framework. For information about integration, see *Integrating Applications with the Oracle Fusion Middleware Audit Framework* in *Securing Applications with Oracle Platform Security Services.*
- Creating the audit definition file that specifies the application's audit events and how they map to the audit schema. For information about audit definition files, see *Creating Audit Definition Files* in *Securing Applications with Oracle Platform Security Services*.
- Registering the application with audit. For information about audit registration, see *Registering the Application with the Audit Service* in *Securing Applications with Oracle Platform Security Services*
- Migrating audit information. For information about audit data migration, see *Migrating Audit Data* in *Securing Applications with Oracle Platform Security Services*.
- Generating audit reports. For information about audit reporting, see *Using Audit Analysis and Reporting* in *Securing Applications with Oracle Platform Security Services*.

Understanding the Runtime Audit Event Flow

If the audit store is not configured in your environment, then the audit records are kept in bus-stop files. An application does not stop execution if it is unable to record an audit event.

The audit event flow is best understood by looking at the following sequence that takes place when an audit event occurs within an application running in an environment where you have configured audit:

1. During application deployment or service start-up, a client Java EE application registers with audit.
2. The service reads the application audit definition file and updates definitions in the audit store.
3. When a user accesses the component or application, an audit function is called to audit the event.
4. The Audit Framework checks whether to audit events with this type, status, and attributes. If they must be audited, then the audit function is called to create the event and collect information such as the status, initiator, resource, and ECID.
5. The event is stored in a bus-stop file. Each application or component has its own bus-stop file.
6. The audit loader pulls the events from bus-stop files, formats the data using the application's metadata, and moves it to the audit store.

About Audit Attributes, Events, and Event Categories

The Audit Framework supports a model that allows you to specify and define dynamically application audit attribute groups, categories, and events.

Audit Attribute Groups

Attribute groups provide broad classification of audit attributes and consist of three types: common, generic, and custom.

- The common attribute group contains system attributes common to all applications, such as the component type, system IP address, and host name. The `IAU_COMMON` database table contains attributes in this group.
- Generic attribute groups contain attributes for audit authentication and user provisioning.
- Custom attribute groups are those defined by an application to meet specific needs. The scope of attributes in a custom group is limited to a component. These attribute groups and attributes are stored in the `IAU_CUSTOMn` table, where n denotes an integer (1,2, and so on).

About Generic Attribute Groups

A generic attribute group refers to a namespace and a version number, and contains one or more attributes. The following example illustrates an attribute group with the `authorization` namespace and version 1.0:

```
<AuditConfig xmlns="http://xmlns.oracle.com/ias/audit/audit-2.0.xsd" >
    <Attributes ns="authorization" version="1.0">
        <Attribute displayName="CodeSource" maxLength="2048" name="CodeSource"
type="string"/>
        <Attribute displayName="Principals" maxLength="1024" name="Principals"
type="string"/>
        <Attribute displayName="InitiatorGUID" maxLength="1024"
name="InitiatorGUID" type="string"/>
```

```
        <Attribute displayName="Subject" maxLength="1024" name="Subject" type="string">
          <HelpText>Used for subject in authorization</HelpText>
        </Attribute>
    </Attributes>
    ……
```

You refer to the `CodeSource` attribute like this:

```
<Attribute name="CodeSource" ns="authorization" version="1.0" />
```

Each generic attribute group is stored in a dedicated database table. The naming conventions are:

- `IAU_`*`GENERIC_ATTRIBUTE_GROUP_NAME`* for table names
- `IAU_`*`ATTRIBUTE_NAME`* for table columns

For example, the `authorization` attribute group is stored in the `IAU_AUTHORIZATION` table with these columns:

- `IAU_CODESOURCE` as string
- `IAU_PRINCIPALS` as string
- `IAU_INITIATORGUID` as string

About Custom Attribute Groups

A custom attribute group refers to a namespace, a version number, and one or more attributes. Each custom attribute incudes:

- Attribute name
- Data type
- Attribute-database column mapping order - This property specifies the order in which an attribute is mapped to a database column of a specific data type in the custom attribute table.
- Help text (optional)
- Maximum length
- Display name
- Size - This property denotes how many values of the same data type the attribute can have. The default size value is 1. A size greater than 1 denotes an attribute that can have two or more values of the same data type. These attributes support all data types except for binary types.

The following example illustrates the definition of the `Accounting` attribute group with the `accounting` namespace and version 1.0:

```
<Attributes ns="accounting" version="1.0">
   <Attribute name="TransactionType" displayName="Transaction Type" type="string"
order="1"/>
   <Attribute name="AccountNumber" displayName="Account Number" type="int" order="2">
      <HelpText>Account number.</HelpText>
   </Attribute>
            ……
 </Attributes>
```

The following example defines the `AccountBalance` attribute with multiple values:

```
<Attribute order="3" displayName="AccountBalance" type="double" name="Balance"
size="2" sinceVersion="1.1">
    <MultiValues>
       <MultiValueName displayName="Previous Balance" index="0">
           <HelpText>the previous account balance</HelpText>
       </MultiValueName>
       <MultiValueName displayName="Current Balance" index="1"/>
    </MultiValues>
</Attribute>
```

About Audit Attribute Data Types

Table 1-2 shows the attribute data types supported and the corresponding Java object types:

Table 1-2 Audit Attribute Data Types

Attribute Data Type	Java Object Type	Notes
Integer	Integer	NA
Long	Long	NA
Float	Float	NA
Double	Double	NA
Boolean	Boolean	NA
DateTime	java.util.Date	NA
String	String	Maximum length 2048 bytes
LongString	String	Unlimited length
Binary	byte[]	NA

Audit Events and Event Categories

An event category contains audit events in a functional area. For example, a session category may contain login and logout events significant to the life cycle of a user session. An event category does not itself define attributes. Instead, it references attributes in component and system attribute groups.

There are two types of event categories:

About System Categories and Events

A system category references common and generic attribute groups and includes audit events. System categories are the base set of component event categories and events. Applications can refer to system categories and use the events in them to log audit events and set filter preset definitions.

The following example illustrates the definition of attributes, events, and the `UserSession` system category with an attribute referencing the common `AuthenticationMethod` attribute:

```
<SystemComponent major="1" minor="0">
 <Attributes ns="common" version ="1.0"></Attributes>
 <Attributes ns="identity" version ="1.0"></Attributes>
 <Attributes ns="authorization" version ="1.0"></Attributes>
 <Events>
```

```
  <Category name="UserSession" displayName="User Sessions">
   <Attributes>
     <Attribute name="AuthenticationMethod" ns="common" version ="1.0" />
   </Attributes>
   <HelpText></HelpText>
   <Event name="UserLogin" displayName="User Logins" shortName="uLogin"></Event>
   <Event name="UserLogout" displayName="User Logouts" shortName="uLogout"
    xdasName="terminateSession"></Event>
   <Event name="Authentication" displayName="Authentication"></Event>
   <Event name="InternalLogin" displayName="Internal Login" shortName="iLogin"
    xdasName="CreateSession"></Event>
   <Event name="InternalLogout" displayName="Internal Logout" shortName="iLogout"
    xdasName="terminateSession"></Event>
   <Event name="QuerySession" displayName="Query Session"     shortName="qSession"></
Event>
   <Event name="ModifySession" displayName="Modify Session"     shortName="mSession"></
Event>
  </Category>
  <Category displayName="Authorization" name="Authorization"></Category>
  <Category displayName="ServiceManagement" name="ServiceManagement"></Category>
 </Events>
</SystemComponent>
```

About Component and Application Categories

A component or application can extend system categories or define new component event categories.

The following example illustrates the definition of a category with the `AccountNumber`, `Date`, and `Amount` attributes from the `accounting` attribute group, and it includes the `purchase` and `deposit` events:

```
 <Category displayName="Transaction" name="Transaction">
    <Attributes>
       <Attribute name="AccountNumber" ns="accounting" version="1.0"/>
       <Attribute name="Date" ns="accounting" version="1.0" />
       <Attribute name="Amount" ns="accounting" version="1.0" />
 </Attributes>
    <Event displayName="purchase" name="purchase"/>
    <Event displayName="deposit" name="deposit">
       <HelpText>depositing funds.</HelpText>
    </Event>
……
 </Category>
```

Extend system categories by creating category references in your application audit definitions, listing the system events that the category includes, and adding attribute references and events to the category reference.

The following example illustrates the definition of the `ServiceManagement` system category reference with the `ServiceTime` attribute, and the `restartService` event:

```
<CategoryRef name="ServiceManagement" componentType="SystemComponent">
    <Attributes>
        <Attribute name="ServiceTime" ns="accounting" version="1.0" />
    </Attributes>
    <EventRef name="startService"/>
    <EventRef name="stopService"/>
    <Event displayName="restartService" name="restartService">
        <HelpText>restart service</HelpText>
```

```
    </Event>
</CategoryRef>
```

Audit Artifact Naming Requirements

The name of a category, an event, or an attribute must:

- Be an English word
- Be less than 26 characters
- Contain characters a-z, A-Z, and numbers 0-9 only
- Start with a letter

About Audit Definition Files

An audit definition file specifies the application's specific audit rules (such as events and filters). Audit definition files provide a way to translate event definitions to foreign languages.

There are two types of audit definition files:

About the component_events.xml File

The `component_events.xml` file specifies the properties audit uses to log audit events, including the following:

- Basic properties
  - The component type, which applications use to register with audit and obtain a runtime auditor instance
  - Major and minor version of the application
- A custom attribute group
- Event categories with attribute references and events
- Component level filter definitions
- Runtime policies

The following example illustrates the definition of this file:

```
<?xml version="1.0"?>
<AuditConfig xmlns="http://xmlns.oracle.com/ias/audit/audit-2.0.xsd">
    <AuditComponent componentType="ApplicationAudit" major="1" minor="0">
        <Attributes ns="accounting" version="1.0">
            <Attribute name="TransactionType" displayName="Transaction Type"
type="string" order="1">
                <HelpText>Transaction type.</HelpText>
           </Attribute>
            <Attribute name="AccountNumber" displayName="Account Number"
type="int" order="2">
                 <HelpText>Account number.</HelpText>
            </Attribute>
            <Attribute name="Date" displayName="Date" type="dateTime" order="3"/>
            <Attribute name="Amount" displayName="Amount" type="float" order="4">
                <HelpText>Transaction amount.</HelpText>
            </Attribute>
            <Attribute name="Status" displayName="Account Status" type="string"
```

```
order="5">
               <HelpText>Account status.</HelpText>
            </Attribute>
        </Attributes>
        <Events>
            <Category displayName="Transaction" name="Transaction">
                <Attributes>
                  <Attribute name="AccountNumber" ns="accounting" version="1.0" />
                  <Attribute name="Date" ns="accounting" version="1.0" />
                  <Attribute name="Amount" ns="accounting" version="1.0" />
                </Attributes>
                <Event displayName="purchase" name="purchase">
                   <HelpText>direct purchase.</HelpText>
                </Event>
                <Event displayName="deposit" name="deposit">
                   <HelpText>depositing funds.</HelpText>
                </Event>
                <Event displayName="withdrawing" name="withdrawing">
                   <HelpText>withdrawing funds.</HelpText>
                </Event>
                <Event displayName="payment" name="payment">
                   <HelpText>paying bills.</HelpText>
                </Event>
            </Category>
            <Category displayName="Account" name="Account">
                <Attributes>
                  <Attribute name="AccountNumber" ns="accounting" version="1.0" />
                  <Attribute name="Status" ns="accounting" version="1.0" />
                </Attributes>
                <Event displayName="open" name="open">
                   <HelpText>Open a new account.</HelpText>
                </Event>
                <Event displayName="close" name="close">
                   <HelpText>Close an account.</HelpText>
                </Event>
                <Event displayName="suspend" name="suspend">
                   <HelpText>Suspend an account.</HelpText>
                </Event>
            </Category>
        </Events>
        <FilterPresetDefinitions>
            <FilterPresetDefinition displayName="Low" helpText="" name="Low">
              <FilterCategory enabled="partial" name="Transaction">
deposit.SUCCESSESONLY(HostId -eq &quot;NorthEast&quot;),withdrawing </FilterCategory>
              <FilterCategory enabled="partial"
name="Account">open.SUCCESSESONLY,close.FAILURESONLY</FilterCategory>
            </FilterPresetDefinition>
            <FilterPresetDefinition displayName="Medium" helpText="" name="Medium">
              <FilterCategory enabled="partial"
name="Transaction">deposit,withdrawing</FilterCategory>
              <FilterCategory enabled="partial" name="Account">open,close</
FilterCategory>
            </FilterPresetDefinition>
            <FilterPresetDefinition displayName="High" helpText="" name="High">
              <FilterCategory enabled="partial"
name="Transaction">deposit,withdrawing,payment</FilterCategory>
              <FilterCategory enabled="true" name="Account"/>
            </FilterPresetDefinition>
        </FilterPresetDefinitions>

        <Policy filterPreset="Low">
```

```
          <CustomFilters>
             <FilterCategory enabled="partial" name="Transaction"> purchase </
FilterCategory>
            </CustomFilters>
        </Policy>
    </AuditComponent>
</AuditConfig>
```

About Runtime Properties

In addition, there are runtime properties you create with Fusion Middleware Control, WLST commands, or during registration. They include the following properties:

- `filterPreset`, to specify the audit filter level
- `specialUsers`, to specify the users to audit always
- `maxBusstopFileSize`, to specify the size of a bus-stop file

Translation Files

The following procedure explains how to generate the XLIFF (XML Localization Interchange File Format) translations files and pack them in the `component_events_xlf.jar` file. At deployment and during registration, this information is stored in the audit store along with the component event definition.

1. Run a command like the following to generate XLIFF files:

```
java -cp $MW_HOME/oracle_common/modules/oracle.jps_12.2.1/
jpsaudit.jar:
 $MW_HOME/oracle_common/modules/oracle.jps_12.2.1/jps-api.jar
oracle.security.audit.tools.NewXlfGenerator
-s
/tmp/comp_events.xml
 -t /tmp/comp_events.xlf
```

2. Translate the generated `xlf` file for the supported languages. This `xlf` file contains translation units as well as help texts for all categories, events, and attributes. The prefixes for these are `Category_, Event_ and Attribute_`.
3. Package the translated files in a JAR file.

About Mapping and Version Rules

Audit registration applies certain rules to create the audit data for the application, and this data is used to maintain different versions of the audit definition and to generate reports.

The following sections explain how the registration works:

What Are Version Numbers?

An audit definition file has a major and a minor version number. Any change introduced to an audit event definition requires updating the version number. These numbers are used by audit registration to determine the compatibility of event definitions and attribute mappings between versions. These version numbers have no relation to Oracle Fusion Middleware version numbers.

Component Version

When you register a component, audit registration checks if this is a first-time registration or an upgrade.

In case of a new registration, the service:

1. Retrieves the component audit and translation information.
2. Parses and validates the definition, and stores it in the audit store.
3. Generates the attribute-column mapping table and saves it in the audit store.

In case of an upgrade, the current version number for the component in the audit store is compared with the new version number to determine whether to proceed with the upgrade.

Java EE Application Version

To reset the version number after you modified an application audit definition, Oracle recommends that you:

- Increase the minor version number only when making changes in an audit definition that will work with the audit data created by the previous attribute database mapping table.

  For example, suppose the current definition version 2.1. When adding a new event that does not affect the attribute database mapping table, you change the version to 2.2, and leave the major version unchanged (major=2). Similarly, increase the minor version after adding a new attribute.
- Increase major version number when making changes where the new mapping table is incompatible with the previous table.

Changes becomes effective after you restart the server.

About Custom Attribute to Database Column Mappings

When you register a new component or application, audit registration creates an attribute-to-database column mapping table from the custom attributes, and then saves this table to the audit store.

If the number of custom attributes is greater than 100, then you must create additional tables manually. OPSS ships with the tables `IAU_CUSTOM` and `IAU_CUSTOM_01` only.

Attribute-database mapping tables are required to ensure unique mappings between your application's attribute definitions and database columns. The audit loader uses mapping tables to load data into the audit store. These tables are also used to generate audit reports from custom `IAU_CUSTOM` database table.

Use the `createAuditDBView` WLST command to generate a SQL file that creates a database view of the audit definitions for your component.

Understanding the Mapping Table for your Component

A custom attribute-database column mapping has the following properties: name, database column name, and data type.

Each custom attribute must have a mapping order number in its definition. Attributes with the same data type are mapped to the database column in the sequence of attribute mapping order.

For example, the following definition file:

```
<Attributes ns="accounting" version="1.1">
   <Attribute name="TransactionType" type="string" maxLength="0"
    displayName="Transaction Type" order="1"/>
   <Attribute name="AccountNumber" type="int" displayName="Account Number"
    order="2">
   <Attribute name="Date" type="dateTime" displayName="Date" order="3"/>
   <Attribute name="Amount" type="float" displayName="Amount" order="4"/>
   <Attribute name="Status" type="string" maxLength="0" displayName="Account
    Status" order="5"/>
   <Attribute name="Balance" type="float" displayName="Account Balance"
    order="6"/>
</Attributes>
```

maps to:

```
<AttributesMapping ns="accounting" tableName="IAU_CUSTOM" version="1.1">
   <AttributeColumn attribute="TransactionType" column="IAU_STRING_001"
    datatype="string"/>
   <AttributeColumn attribute="AccountNumber" column="IAU_INT_001"
    datatype="int"/>
   <AttributeColumn attribute="Date" column="IAU_DATETIME_001"
    datatype="dateTime"/>
   <AttributeColumn attribute="Amount" column="IAU_FLOAT_001" datatype="float"/>
   <AttributeColumn attribute="Status" column="IAU_STRING_002"
datatype="string"/>
   <AttributeColumn attribute="Balance" column="IAU_FLOAT_002" datatype="float"/>
</AttributesMapping>
```

The version number of the attribute-database column mapping table matches the version number of the custom attribute group. This allows your application to maintain a backward compatibility of attribute mappings across audit definition versions.

Managing Audit

This section explains the main administration tasks and tools you use to manage the audit store, audit policies, and bus-stop files.
This section includes the following topics:

Audit Administration Tasks

Setting up audit in your environment involves the following major tasks:

- Planning the type of store to use for audit records and the store configuration details. For information about audit store management, see *Managing the Audit Store* in *Fusion Middleware Securing Applications with Oracle Platform Security Services* .
- Configuring and maintaining audit policies so that audit events are generated. For information about audit policies, see *Managing Audit Policies* in *Fusion Middleware Securing Applications with Oracle Platform Security Services* .
- Configuring audit reports and queries. For information about reporting, see *Using Audit Analysis and Reporting* in *Fusion Middleware Securing Applications with Oracle Platform Security Services*.

- Registering applications. For information about application registration, see *Registering the Application with the Service* in *Fusion Middleware Securing Applications with Oracle Platform Security Services*.
- Migrating audit information. For information about audit data migration, see *Migrating Audit Data* in in *Fusion Middleware Securing Applications with Oracle Platform Security Services*.
- Administering the audit database, including increasing the database size that stores the generated audit data, and backing up and purging that data. For information about audit administration, see *Audit Database Administration* in *Fusion Middleware Securing Applications with Oracle Platform Security Services*.

About Audit Data Sources

When you create a domain, the process generates the audit schema, a data structure required to store audit records in the database. It also sets up an audit data source in the server that uses the audit schema. If your environment is not set up with a database to store records, then audit records are kept in bus-stop files.

For more information, see Bus-Stop Files.

Managing Bus-Stop Files

After the bus-stop file reaches a certain size and all the data was uploaded to the database, the audit loader deletes the file from the file system. Specify the location and maximum size of bus-stop files, so that bus-stop files are automatically deleted. Deleting audit files manually is not recommended.

Bus-Stop File Locations

Bus-stop files for Java components are located in the following directory:

```
$DOMAIN_HOME/servers/$SERVER_NAME/logs/auditlogs/Component_Type
```

Bus-stop files for system components are located in the following directory:

```
$ORACLE_INSTANCE/auditlogs/Component_Type/Component_Name
```

Bus-Stop File Size

In Java components, the maximum size of a bus-stop file is set with the `audit.maxFileSize` property.

In system components, the maximum size of a bus-stop file is set in the `auditconfig.xml` file:

```
<serviceInstance name="audit" provider="audit.provider">
  <property name="audit.maxFileSize" value="10240" />
  <property name=" audit.loader.repositoryType " value="Db" />
</serviceInstance>
```

When you switch from a file to a database store for audit data, all the events collected in the files are moved to the database tables and the audit files are deleted.

Configuring Standalone Audit Loader

The standalone audit loader moves records from bus-stop files to the audit store periodically. The mechanism driving the audit loader depends on the application environment:

- Java EE components and applications use the audit loader functionality provided by OPSS runtime. The standalone audit loader is not needed in these environments.
- System components and non-Java applications use the audit loader functionality provided by the `StandAloneAuditLoader` command.
- Java SE applications also use the standalone audit loader depending on where the bus-stop files are written. For information about audit for Java SE applications, see *Common Audit Scenarios in Java SE Applications* in *Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services* .

The following sections explain how to set up and run the standalone audit loader:

Configuring the Environment

The following settings apply only to non-Java applications and system components.

Before you run the standalone audit loader, set the following audit loader parameters:

- `ORACLE_HOME`, the full path to the home directory
- `COMMON_COMPONENTS_HOME`, the full path to the Java Required Files (JRF) directory
- `ORACLE_INSTANCE`, the full path of an Oracle instance directory
- `auditloader.jdbcString`, the Java Database Connectivity (JDBC) connection string for the database where the audit data is stored
- `auditloader.username`, the name of the user who runs the audit loader

In addition, make sure that the password for the database schema user is available and stored. This password is specified once.

To specify the database schema user password, use the java `StandAloneAuditLoader` command with the `-Dstore.password=true` property:

```
$JDK_HOME/bin/java
    -classpath $COMMON_COMPONENTS_HOME/modules/oracle.jps_12.2.1/jps-manifest.jar
    -Doracle.home=$ORACLE_INSTANCE -Doracle.instance=$ORACLE_INSTANCE
    -Dauditloader.jdbcString=jdbc:oracle:thin:@host:port:sid
    -Dauditloader.username=username
    -Dstore.password=true
    oracle.security.audit.ajl.loader.StandaloneAuditLoader
```

which will prompt you to enter a password. The command generates the `cwallet.sso` file containing the password you entered.

Running Standalone Audit Loader

To run the loader, use the `StandAloneAuditLoader` command:

```
$JDK_HOME/bin/java
    -classpath $COMMON_COMPONENTS_HOME/modules/oracle.jps_12.2.1/jps-manifest.jar
    -Doracle.home=$ORACLE_INSTANCE -Doracle.instance=$ORACLE_INSTANCE
    -Dauditloader.jdbcString=jdbc:oracle:thin:@host:port:sid
    -Dauditloader.username=username
    oracle.security.audit.ajl.loader.StandaloneAuditLoader
```

This command is typically scheduled to run automatically so that audit records are periodically uploaded to the audit store.

Keyboard Shortcuts

This section describes the keyboard shortcuts that can be used instead of pointing devices. Following are the topics covered in this section:

About Keyboard Shortcuts

Keyboard shortcuts are helpful to users as they act as an alternative to mouse. Using keyboard shortcuts for ADF Faces applications can greatly increase your productivity, reduce repetitive strain, and help keep you focused.

Keyboard shortcuts provide an alternative to pointing devices for navigating the page. There are five types of keyboard shortcuts that can be provided in ADF Faces applications:

- Tab traversal, using Tab and Shift+Tab keys: Moves the focus through UI elements on a screen.
- Accelerator keys (*hot keys*): bypasses menu and page navigation, and performs an action directly, for example, Ctrl+C for Copy.
- Access keys: Moves the focus to a specific UI element, for example, Alt+F for the File menu.
- Default cursor/focus placement: Puts the initial focus on a component so that keyboard users can start interacting with the page without excessive navigation.
- Enter key: Triggers an action when the cursor is in certain fields or when the focus is on a link or button.

Keyboard shortcuts are not required for accessibility. Users should be able to navigate to all parts and functions of the application using the Tab and arrow keys, without using any keyboard shortcuts. Keyboard shortcuts merely provide an additional way to access a function quickly.

It is the application developer's responsibility to provide user assistance that identifies the application's available keyboard shortcuts. If the application includes an ADF component that provides keyboard shortcuts, the developer should also provide a popup or other help that details the keyboard shortcuts available for the ADF component.

Tab Traversal

Tab Traversal can be defined as an order in which the elements of the ADF Faces user interface receive keyboard focus on successive passes of the Tab key.

Tab traversal allows the user to move the focus through different UI elements on a page.

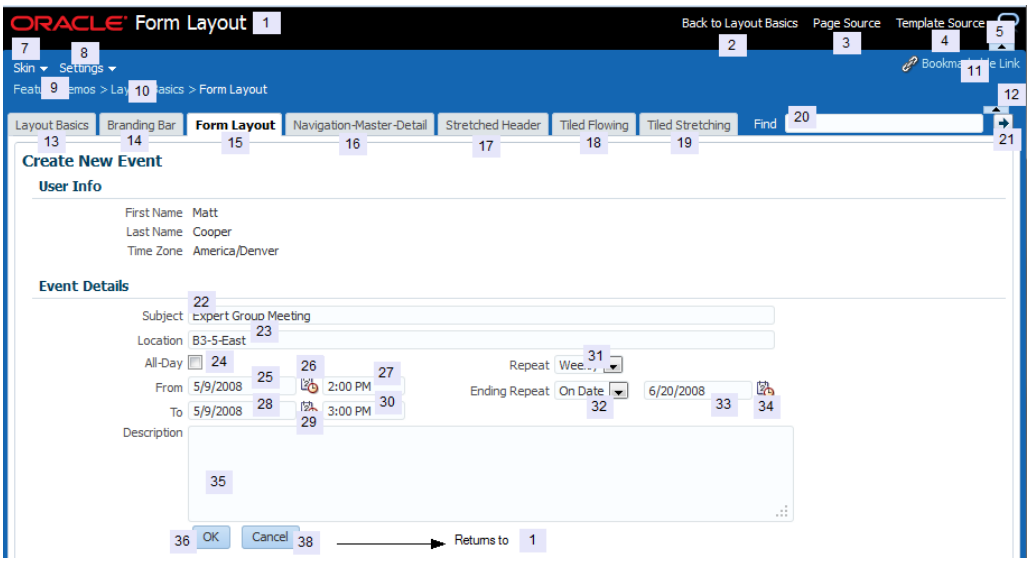
All active elements of the page are accessible by Tab traversal, that is, by using the Tab key to move to the next control and Shift+Tab to move to the previous control. In most cases, when a control has focus, the action can then be initiated by pressing Enter.

Some complex components use arrow keys to navigate after the component receives focus using the Tab key.

Tab Traversal Sequence on a Page

Default Tab traversal order for a page is from left to right and from top to bottom, as shown in Figure 1-1. Tab traversal in a two-column form layout does not follow this pattern, but rather follows a columnar pattern. On reaching the bottom, the tab sequence repeats again from the top.

Figure 1-1 Tab Traversal Sequence on a Page



Avoid using custom code to control the tab traversal sequence within a page, as the resulting pages would be too difficult to manage and would create an inconsistent user experience across pages in an application and across applications.

To improve keyboard navigation efficiency for users, you should set the `initialFocusId` attribute on the document. For accessibility purposes, you should also define a `skipLinkTarget` and include a `skip navigation` link at the top of the page, which should navigate directly to the first content-related tab stop.

Tab Traversal Sequence in a Table

The Tab traversals in a table establishes a unique row-wise navigation pattern when the user presses the Tab key to navigate sequentially from one cell to another. One of the following actions can occur when user presses the Enter key in a table cell:

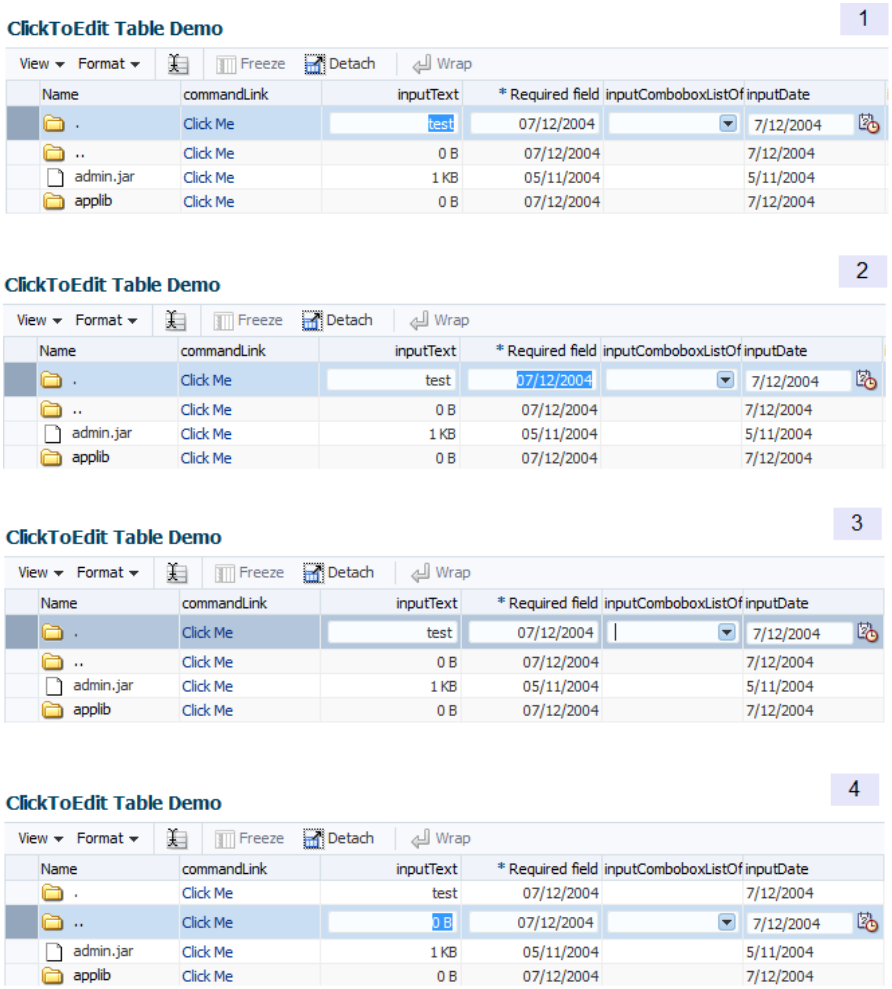
- Clicks on an editable cell and presses Enter key—the focus moves to the editable cell below in the same column in the next row

- Clicks on an editable cell, edits the contents of the cell, and presses Enter key—the focus completes the action in the current cell and moves to the editable cell below in the same column in the next row
- Clicks on an editable cell and presses Tab key without editing the cell, once or more than once, and then presses Enter key— the focus moves to the editable cell below in the next row, in the same column where the user started pressing the tab key.
- Clicks on an editable cell, edits the contents of the cell, and presses Tab key, once or more than once, then presses Enter key—the focus completes the action in the current cell where the user started pressing the Tab key. Focus traverses through the cells sequentially per the number of times the Tab key is pressed. Then, the focus moves to the editable cell below in the next row, in the same column where the user started pressing the tab key
- Clicks on a non-editable cell and presses Enter key—the focus moves to the first editable cell in the next row.
- Clicks on a non-editable cell and presses Tab key, once or more than once, then presses Enter key—the focus traverses through the cells sequentially per the number of times the Tab key is pressed and moves to the first editable cell in the next row.
- Clicks on a cell that contains any command, such as menu, link, or a dialog box, then presses Enter key—the default action for that command is executed

> **Note:**
>
> When user uses the Tab key to traverse through the cells sequentially and presses Enter key to move to the next row , a navigation pattern is formed based on the first set of Tab keys, which is followed in subsequent rows. The navigational pattern is not recognized if arrow keys are used to navigate from one cell to another.

Figure 1-2 shows an example of a Tab and Enter keys traversal sequence in a table.

Figure 1-2 Tab and Enter Keys Traversal Sequence in a Table



In Figure 1-2, the user has navigated the rows without editing any cell in the following way:

1. The user clicks a cell in the **inputText** column, giving it focus and making it editable.

   Because the Tab key is used to navigate, the **inputText** column is recognized as the starting column for the navigation pattern.

2. The user presses the Tab key and moves the focus in the same row to the cell of the *** Required field** column.

3. The user presses the Tab key and moves the focus in the same row to the cell of the **inputComboListOf** column.

4. The user presses the Enter key and the focus shifts to the **inputText** column in the next row.

Shortcut Keys

While it is possible to use the tab key to move from one control to the next in an ADF Faces application, keyboard shortcuts like the accelerator and access keys are more convenient and efficient. They help users to navigate around the web application easily and access a menu or function quickly.

There are various keyboard shortcuts provided by ADF Faces itself, as well as component attributes that enable you to create specific keyboard shortcuts for your specific applications. ADF Faces categorizes shortcut keys for components into two types, accelerator keys and access keys.

> **Note:**
>
> It is the application developer's responsibility to provide user assistance to identify the application's available keyboard shortcuts. Because an ADF component's terminology can conflict with an application's terminology, the application should also provide a popup or other help that details the keyboard shortcuts available for that component.

Accelerator Keys

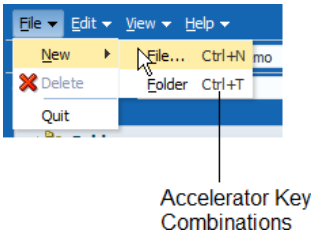
Accelerator keys bypass menu and page navigation and perform actions directly. Accelerator keys are sometimes also called **hot keys**. Common accelerator keys in a Windows application, such as Internet Explorer, are Ctrl+O for Open and Ctrl+P for Print.

Accelerator keys are single key presses (for example, Enter and Esc) or key combinations (for example, Ctrl+A) that initiate actions immediately when activated. A key combination consists of a meta key and an execution key. The meta key may be Ctrl (Command on a Macintosh keyboard), Alt (Option on a Macintosh keyboard), or Shift. The execution key is the key that is pressed in conjunction with the meta key.

Some ADF Faces components have their own built-in accelerator keys. For example, Ctrl+Alt+M is the accelerator key to open the context menu. For more information about ADF Faces components with their own built-in accelerator keys, see the component tag documentation.

ADF Faces also enable you to provide custom accelerator keys to specific menu items, as shown in Figure 1-3. All assigned menu accelerator keys are visible when you open the menu.

Figure 1-3 Accelerator Keys in a Menu



When defining accelerator keys, you must follow these guidelines:

- Because accelerator keys perform actions directly, if a user presses an accelerator key unintentionally, data may be lost or incorrect data may be entered. To reduce the likelihood of user error, accelerator keys should be used sparingly, and only for frequently and repetitively used functions across applications. As a general rule, less than 25% of available functions should have accelerator keys.
- Custom accelerator keys must not override accelerator keys that are used in the menus of ADF Faces-supported browsers (see the browser and system requirements for supported operating systems and browsers in ADF Faces), and must not override accelerator keys that are used in assistive technologies such as screen readers.
- Custom menu accelerator keys must always be key combinations. The meta key may be Ctrl, Ctrl+Shift, or Ctrl+Alt. Ctrl+Alt is the most used metakey because Ctrl and Ctrl+Shift are commonly used by browsers. The execution key must be a printable character (ASCII code range 33-126).
- Custom menu accelerator keys must be unique. If a page were to have different components that used the same accelerator, it would be difficult for the browser to predict which actions would be executed by the accelerator at any given time.

> **Note:**
>
> In Windows, users have the ability to assign a Ctrl+Alt+*character key* sequence to an application desktop shortcut. In this case, the key assignment overrides browser-level key assignments. However, this feature is rarely used, so it can generally be ignored.

Certain ADF Faces components have built-in accelerator keys that apply when the component has focus. Of these, some are reserved for page-level components, whereas others may be assigned to menus when the component is not used on a page. Table 1-3 lists the accelerator keys that are already built into page-level ADF Faces components. You must not use these accelerator keys at all.

Table 1-3 Accelerator Keys Reserved for Page-Level Components

Accelerator Key	Used In	Function
Ctrl+Alt+W Ctrl+Shift+W	Pop-up Messaging Secondary Windows	Toggle focus between open popups.
Ctrl+Alt+P	Splitter	Give focus to splitter bar.

The menu commands take precedence if they are on the same page as page-level components, and have the same accelerator keys. For this reason, you must not use the accelerator keys listed in Table 1-5 and Table 1-9 in menus when the related component also appears on the same page.

Access Keys

Access keys move the focus to a specific UI element, and is defined by the `accessKey` property of the ADF Faces component.

Access keys relocate cursor or selection focus to specific interface components. Every component on the page with definable focus is accessible by tab traversal (using Tab and Shift+Tab); however, access keys provide quick focus to frequently used components. Access keys must be unique within a page.

The result of triggering an access key depends on the associated element and the browser:

- Buttons: In both Firefox and Internet Explorer, access keys give focus to the component and directly execute the action. Note that in Internet Explorer 7 access key gives focus to the component, but does not execute the action.
- Links: In Firefox, access keys give focus to the component and directly navigate the link; in Internet Explorer, access keys give focus only to the link.
- Other Elements: In both browsers, access keys give focus only to the element. For checkbox components, the access key toggles the checkbox selection. For option buttons, the access key performs selection of the option button.

Note that the access key could be different for different browsers on different operating systems. You must refer to your browser's documentation for information about access keys and their behavior. Table 1-4 lists access key combinations for button and anchor components in some common browsers.

Table 1-4 Access Key For Various Browsers

Browser	Operating System	Key Combination	Action
Google Chrome	Linux	Alt + mnemonic	Click
Google Chrome	Mac OS X	Control + Option + mnemonic	Click
Google Chrome	Windows	Alt +mnemonic	Click
Mozilla Firefox	Linux	Alt + Shift + mnemonic	Click
Mozilla Firefox	Mac OS X	Control + mnemonic	Click
Mozilla Firefox	Windows	Alt + Shift + mnemonic	Click
Microsoft Internet Explorer 7	Windows	Alt + mnemonic	Set focus
Microsoft Internet Explorer 8	Windows	Alt + mnemonic	Click or set focus
Apple Safari	Windows	Alt + mnemonic	Click
Apple Safari	Mac OS X	Control + Option + mnemonic	Click

> **Note:**
>
> - Different versions of a browser might behave differently for the same access key. For example, using Alt + mnemonic for a button component in Internet Explorer 7 sets focus on the component, but it triggers the click action in Internet Explorer 8.
> - In Firefox, to change the default behavior of the component when access key combination is used, change the configuration setting for the `accessibility.accesskeycausesactivation` user preference.
> - Some ADF Faces components that are named as Button do not use HTML button elements. For example, `af:button` uses an anchor HTML element.

If the mnemonic is present in the text of the component label or prompt (for example, a menu name, button label, or text box prompt), it is visible in the interface as an underlined character, as shown in Figure 1-4. If the character is not part of the text of the label or prompt, it is not displayed in the interface.

Figure 1-4 Access Key

When defining access keys, you must follow these guidelines:

- Access keys may be provided for buttons and other components with a high frequency of use. You may provide standard cross-application key assignments for common actions, such as Save and Cancel. Each of these buttons is assigned a standard mnemonic letter in each language, such as S for Save or C for Cancel.
- A single letter or symbol can be assigned only to a single instance of an action on a page. If a page had more than one instance of a button with the same mnemonic, users would have no way of knowing which button the access key would invoke.
- Focus change initiated through access keys must have alternative interactions, such as direct manipulation with the mouse (for example, clicking a button).
- The mnemonic must be an alphanumeric character — not a punctuation mark or symbol — and it must always be case-insensitive. Letters are preferred over numbers for mnemonics.
- In Internet Explorer, application access keys override any browser-specific menu access keys (such as Alt+F for the File menu), and this can be a usability issue for users who habitually use browser access keys. Thus, you must not use access keys that conflict with the top-level menu access keys in ADF Faces-supported browsers (for example, Alt+F, E, V, A, T, or H in the English version of Internet Explorer for Windows XP).
- You are responsible for assigning access keys to specific components. When choosing a letter for the access key, there are a few important considerations:

- Ease of learning: Although the underlined letter in the label clearly indicates to the user which letter is the access key, you should still pick a letter that is easy for users to remember even without scanning the label. For example, the first letter of the label, like Y in Yes, or a letter that has a strong sound when the label is read aloud, such as x in Next.
- Consistency: It is good practice to use the same access key for the same command on multiple pages. However, this may not always be possible if the same command label appears multiple times on a page, or if another, more frequently used command on the page uses the same access key.
- Translation: When a label is translated, the same letter that is used for the access key in English might not be present in the translation. Developers should work with their localization department to ensure that alternative access keys are present in component labels after translation. For example, in English, the button **Next** may be assigned the mnemonic letter x, but that letter does not appear when the label is translated to **Suivantes** in French. Depending on the pool of available letters, an alternative letter, such as S or v (or any other unassigned letter in the term Suivantes), should be assigned to the translated term.

> **Note:**
>
> For translation reasons, you should specify access keys as part of the label. For example, to render the label **Cancel** with the C access key, you should use `&Cancel` in the `textAndAccessKey` property (where the ampersand denotes the mnemonic) rather than C in the `accessKey` property. Product suites must ensure that access keys are not duplicated within each supported language and do not override access keys within each supported browser unless explicitly intended.

Shortcut Keys for Common Components

Table 1-5 lists the shortcut keys assigned to common components such as Menu, Menu bar, Multi-Select Choice List, Multi-Select List Box, and so on.

Table 1-5 Shortcut Keys Assigned to Common Components

Shortcut Key	Components	Function
Enter Spacebar	All components	Activate the component, or the component element that has the focus.
Tab Shift+Tab	All components Flash components like ThematicMap, Graph, and Gauge	Move focus to next or previous editable component.
Ctrl+A	All components	Select all.
Alt+Arrow Down	Multi-Select Choice List Multi-Select List Box	Open the list. Use arrow keys to navigate, and press Enter or Spacebar to select.
Ctrl+Shift+Home Ctrl+Shift+End	Multi-Select Choice List Multi-Select List Box	Select all items from top to current selection, or select all items from current selection to bottom.

Table 1-5 (Cont.) Shortcut Keys Assigned to Common Components

Shortcut Key	Components	Function
Arrow Left Arrow Right	Menu Bar	Move focus to different menu on a menu bar.
	Splitter Input Number Slider Input Range Slider	Move splitter left or right when it is in focus.
	Input Number Spinbox	Move slider left or right when input number slider or input range slider is in focus.
		Increment or decrement the value when input number spinbox is in focus.
Arrow Up Arrow Down	Menu	Move focus to different menu items in a menu.
	Splitter Input Number Slider Input Range Slider	Move splitter up or down when it is in focus.
		Move slider up or down when input number slider or input range slider is in focus.

Shortcut Keys for Widgets

Table 1-6 lists the shortcut keys assigned to common widgets such as Disclosure control, Hierarchy control, and Dropdown lists.

Table 1-6 Shortcut Keys Assigned to Common Widgets

Shortcut Key	Components	Function
Enter Arrow Down/Arrow Up	Disclosure Control	Open a closed Disclosure control, or close a open Disclosure control. A disclosure control is an icon that indicates that more content is available to either be shown or hidden.
Ctrl+Alt+R	Active Data	Applicable only if the page contains active data.
Ctrl+Shift+^	Hierarchy Control	If in hierarchy viewer, open the hierarchy popup.
Alt+Down Arrow	Dropdown list	Open the dropdown list.
Enter	Dropdown list	Select the focussed option of dropdown list.
Ctrl+A	Multi-Select List Box	Select all options.
Ctrl+Shift+Home	Multi-Select List Box	Select all options from the first option to the current option.
Ctrl+Shift+End	Multi-Select List Box	Select all options from the current option to the last option.
Ctrl+Alt+M	Various components	Opens the context menu in components that support it, such as Calendar and Table.

Table 1-6 (Cont.) Shortcut Keys Assigned to Common Widgets

Shortcut Key	Components	Function
Ctrl+Shift+W Ctrl+Alt+W	Various components	Toggle between open detachable menus.
Ctrl+Alt+P	Splitter	Move focus to next Splitter component.
Enter	Splitter	If the Splitter is in focus, toggles the split section from closed to open state.
Ctrl+Alt+F4	Tab	Remove the tab, if it is removable.

Shortcut Keys for Rich Text Editor Component

Table 1-7 lists shortcut keys assigned to the Rich Text Editor component. In regular mode, all toolbar controls appear on top of the Rich Text Editor area.

Table 1-7 Shortcut Keys Assigned to Rich Text Editor Component

Shortcut Key	Components	Function
Ctrl+B	Rich Text Editor	Boldface
Ctrl+I	Rich Text Editor	Italics
Ctrl+U	Rich Text Editor	Underline
Ctrl+5	Rich Text Editor	Strikethrough
Ctrl+E	Rich Text Editor	Center alignment
Ctrl+J	Rich Text Editor	Full-justified alignment
Ctrl+L	Rich Text Editor	Left alignment
Ctrl+R	Rich Text Editor	Right alignment
Ctrl+H	Rich Text Editor	Create hyperlink
Ctrl+M	Rich Text Editor	Increase indentation
Ctrl+Shift+M	Rich Text Editor	Decrease indentation
Ctrl+Shift+H	Rich Text Editor	Remove hyperlink
Ctrl+Shift+L	Rich Text Editor	Bulleted list
Ctrl+Alt+L	Rich Text Editor	Numbered list
Ctrl+Shift+S	Rich Text Editor	Clear text styles
Ctrl+Alt+-	Rich Text Editor	Subscript
Ctrl+Alt++	Rich Text Editor	Superscript
Ctrl+Alt+R	Rich Text Editor	Enable rich text editing mode
Ctrl+Alt+C	Rich Text Editor	Enable source code editing mode
Ctrl+Y	Rich Text Editor	Redo
Ctrl+Z	Rich Text Editor	Undo

Shortcut Keys for Table, Tree, and Tree Table Components

Table 1-8 lists shortcut keys assigned to Table, Tree, and Tree Table.

Table 1-8 Shortcut Keys Assigned to Table, Tree, and Tree Table components

Shortcut Key	Components	Function
Tab Shift+Tab	Table Tree Table	Move focus to next or previous cell or editable component. In a table, navigate to the next or previous editable content in cells in left-to-right direction. If the focus is on the last cell of a row in the table, the Tab key moves focus to the first editable cell in the next row. Similarly, Shift + Tab moves focus to the previous row.
Ctrl+A	Table Tree Table	Select all components, including column headers, row headers, and data area.
Ctrl+Alt+M	Table Tree Tree Table	Launch context menu. You can also launch context menu by pressing Ctrl+Alt+B.
Ctrl+Shift+^	Tree Tree Table	Go up one level.
Ctrl+Arrow Right	Table Tree Tree Table	In a table, expand row. In a tree or tree table, expand nodes or `detailStamp` facets.
Ctrl+Arrow Left	Table Tree Tree Table	In a table, collapse row. In a tree or tree table, collapse nodes or `detailStamp` facets.
Enter Shift+Enter	Table Tree Tree Table	Navigate to the next editable cell or previous editable cell of the column. In a table, navigate to the next or previous editable content in cells in top-to-bottom direction. If focus is on the column header, sort table data in ascending order. Pressing Enter again sorts the column in descending order. If the focus is on the filter cell, perform table filtering. In a table, if the user presses Tab key to navigate from one cell to another and presses Enter, move focus to the next row to follow same navigational pattern. See Tab Traversal Sequence in a Table.

Table 1-8 (Cont.) Shortcut Keys Assigned to Table, Tree, and Tree Table components

Shortcut Key	Components	Function
Arrow Left Arrow Right	Table Tree Table	Move focus. In a table, when the focus is on an editable component, move the text cursor.
Arrow Up Arrow Down	Table Tree Table	Move focus. If a row is selected, move focus to the previous row or next row. If no row is selected, scroll the table one row up or down. In a table, when the focus is on an editable component that supports multiple options (such as `selectOneChoice` and `inputNumberSpinBox`), scroll the selected option. If the first row is selected, move focus to the column header. In an editable table, if the user clicks a cell with an editable component (such as a text box, or a checkbox), a button or a link component, focus is set to the component in the cell. To use Up and Down arrow keys for navigation, focus should be moved from the editable component to the cell. The user would need to click on the background of the same cell (or any cell of the same row) again to move the focus. **Note**: If `selectionEventDelay` is `enabled`, row selection during keyboard navigation is delayed by 300ms to allow table keyboard navigation without causing unwanted row selection.
Ctrl+Arrow Up Ctrl+Arrow Down	Table	Move focus. If in edit mode, submit the changes made in the current row and navigate to the previous row or next row. In the click-to-edit table, when the focus is on an editable component that supports multiple options (such as `selectOneChoice` and `inputNumberSpinBox`), scroll the selected option.
Ctrl+Arrow Left Ctrl+Arrow Right	Table	Move focus. If in edit mode, when the focus is on an editable component, move the text cursor.

Table 1-8 (Cont.) Shortcut Keys Assigned to Table, Tree, and Tree Table components

Shortcut Key	Components	Function
Shift+Arrow Left Shift+Arrow Right	Table Tree Table	Move focus and add to selection.
Ctrl+Shift+Arrow Left Ctrl+Shift+Arrow Right	Table Tree Table	Move the selected column to the left or right.
Shift+Arrow Up Shift+Arrow Down	Table Tree Table Tree	Select multiple rows.
Page Up Page Down	Table Tree Table	If a row is selected, scroll and select the same row of the next or previous page. If no row is selected, scroll by one page.
Alt+Page Up Alt+Page Down	Table Tree Table	Horizontally scroll the table to the right or left.
Space Bar	Table	Select the node.
Ctrl+Space Bar	Tree Tree Table	To select or remove multiple nodes, press Ctrl+Space Bar.
Shift+Space Bar	Table Tree Table	Select multiple rows.
Esc	Table Tree Table	Remove selection. If the focus is on the cell, exit click-to-edit mode, revert the cell value to original value, and return focus to the cell. Press Esc key again to move focus to the row header.
F2	Table Tree Table	Activate click-to-edit mode for the row. Press F2 again to disable cell navigation mode.

Shortcut Keys for ADF Data Visualization Components

Table 1-9 lists shortcut keys assigned to ADF Data Visualization Components including charts, diagram, Gantt chart, hierarchy viewer components, geographic and thematic maps, NBox, pivot table, and pivot filter bar.

Table 1-9 Shortcut Keys Assigned to ADF Data Visualization Components

Shortcut Key	Components	Function
Arrow Left Arrow Right	Charts: Area, Bar, Bubble, Combination, Funnel, Line, Pie, Scatter, Spark Chart legend with horizontal orientation List region of all Gantt chart types Project Gantt chart region Scheduling Gantt chart region Resource Utilization Gantt chart region Geographic and Thematic Map Hierarchy Viewer - nodes Pivot table Pivot filter bar NBox Diagram	Move focus. If the focus is on the bars in bar charts, move focus and selection to bar on left or bar on right. If the focus is on a pie slice in a pie chart, move focus and selection to previous series in a counterclockwise direction or next series in a clockwise direction. If the focus is on a dot, bubble, or bar in an area, bubble, combination, funnel, line, scatter, or spark chart, move focus and selection to the nearest bar, dot, or bubble on left or right. If the focus is on a series in a chart legend, move focus to series on left or series on right. If the focus is on the chart region of scheduling Gantt, the arrow key navigation selects the previous or next taskbar of the current row. If the focus is on the time bucket of resource utilization Gantt, the arrow key navigation selects the previous or next time bucket in the current row. If the focus is on the ADF geographic map, the arrow key navigation pans left or right by a small increment. Press Home or End key to pan by a large increment. If the focus is on the node component of ADF hierarchy viewer, press Ctrl+Arrow to move the focus left or right without selecting the component. If you are using arrow keys to navigate cells of an editable pivot table, each focused cell is activated for editing before allowing you to navigate to the next cell, making the navigation slower. Press the Esc key to deactivate the edit mode of the focused cell, and navigate faster. To edit a cell, press the F2 or Enter key. If the focus is on the pivot table data cell, press Ctrl+Arrow Left to jump to the corresponding row header cell. If the locale is bidirectional (such as Arabic), press Ctrl+Arrow Right to jump to the corresponding row header cell. If the focus is on an NBox cell or node, move focus and selection to the next or previous cell, parent node (sorted by size in descending order) or individual

Table 1-9 (Cont.) Shortcut Keys Assigned to ADF Data Visualization Components

Shortcut Key	Components	Function
		node. Node navigation is based on list navigation; down or right moves to the next element and up or left moves to the previous element.

Table 1-9 (Cont.) Shortcut Keys Assigned to ADF Data Visualization Components

Shortcut Key	Components	Function
Arrow Up Arrow Down	Charts: Area, Bar (Stacked), Bubble, Combination, Funnel, Horizontal Bar, Line, Scatter, Spark Chart legend with vertical orientation List region of all Gantt chart types Project Gantt chart region Scheduling Gantt chart region Resource Utilization Gantt chart region Geographic and Thematic Map Hierarchy Viewer - nodes Pivot table Pivot filter bar NBox	Move focus. If the focus is on the bars in horizontal bar charts, move focus and selection up or down to next or previous bar. If the focus is on a stacked bar chart, move focus and selection up or down to next or previous series on the same bar. If the focus is on a dot, bubble, or bar in an area, bubble, combination, funnel, line, scatter, or spark chart, move focus and selection up or down to the nearest bar, dot, or bubble. If the focus is on a series in a chart legend, move focus up or down to next or previous series. If the focus is on the chart region of project Gantt, the arrow key navigation selects previous or next row. If the focus is on the chart region taskbar of scheduling Gantt, the arrow key navigation selects the first taskbar of the previous row or the next row. If the focus is on the time bucket of resource utilization Gantt, the arrow key navigation selects the time bucket of the previous row or next row. If the focus is on the ADF geographic map component, the arrow key navigation pans up or down by a small increment. If the focus is on the node component of ADF hierarchy viewer, press Ctrl+Arrow keys to move the focus up or down without selecting the component. If you are using arrow keys to navigate cells of an editable pivot table, each focused cell is activated for editing before allowing you to navigate to the next cell, making the navigation slower. Press the Esc key to deactivate the edit mode of the focused cell, and navigate faster. To edit a cell, press the F2 or Enter key. If the focus is on the pivot table data cell, press Ctrl+Arrow Up to jump to the corresponding column header cell. If the focus is on an NBox cell or node, move focus and selection up or down to the nearest cell, parent node (sorted by size in descending order) or individual node. Node navigation is based on list navigation; down or right moves to the

Table 1-9 (Cont.) Shortcut Keys Assigned to ADF Data Visualization Components

Shortcut Key	Components	Function
		next element and up or left moves to the previous element.
Page Up Page Down	Chart legend with vertical orientation Chart plot area Geographic and Thematic Map Hierarchy Viewer - diagram	If the focus is on a chart legend, scroll up or down. If the focus is on a chart plot area, pan up or down. If the focus is on the geographic map component, the page key navigation pans up or down by a large increment. If the focus is on the diagram of a hierarchy viewer, press and hold to Page Up or Page Down keys to pan up or down. Press Shift+Page Up or Shift+Page Down to pan left or right. Press and hold Shift+Page Down to pan continuously.
+	Geographic and Thematic Map Hierarchy Viewer - diagram	Increase zoom level. If the focus is on the diagram of a hierarchy viewer, press number keys 1 through 5 to zoom from 10% through 100%. Press 0 to zoom the diagram to fit within available space. Press and hold to continuously increase zoom.
-	Geographic and Thematic Map Hierarchy Viewer - diagram	Decrease zoom level. If the focus is on the diagram of a hierarchy viewer, press number keys 1 through 5 to zoom from 10% through 100%. Press 0 to zoom the diagram to fit within available space. Press and hold to continuously decrease zoom.
Ctrl+Alt+M	All Gantt chart types Pivot table Pivot filer bar	Launch context menu.
Ctrl+Left Arrow Ctrl+Right Arrow	Charts: Area, Bar, Bar (Stacked), Bubble, Funnel, Horizontal Bar, Line, Pie, Scatter, Spark NBox	Move focus to nearest bar, dot, or bubble to the left or right of the current selection, but do not select. If the focus is on a pie slice in a pie chart, move focus to previous series in a counterclockwise direction or next series in a clockwise direction, but do not select. If the focus is on a series in a stacked bar chart, move focus to nearest series to the left or right of the selected series, but do not select. If the focus is on an NBox node, move focus without selection.

Table 1-9 (Cont.) Shortcut Keys Assigned to ADF Data Visualization Components

Shortcut Key	Components	Function
Ctrl+Up Arrow Ctrl+Down Arrow	Charts: Area, Bar, Bar (Stacked), Bubble, Combination, Funnel, Horizontal Bar, Line, Scatter, Spark NBox	Move focus and to nearest bar, dot, or bubble above or below the current selection, but do not select. If the focus is on a series in a stacked bar chart, move focus to nearest series above or below the selected series, but do not select. If the focus is on an NBox node, move focus without selection.
Ctrl+Spacebar	Charts: Area, Bar, Bar (Stacked), Bubble, Combination, Funnel, Horizontal Bar, Line, Pie, Scatter, Spark NBox	Move focus and to nearest bar, dot, or bubble above or below the current selection, but do not select. If the focus is on a series in a stacked bar chart, move focus to nearest series above or below the selected series, but do not select. If the focus is on an NBox node, select or multi-select.
Shift+Left Arrow Shift+Right Arrow	Charts: Area, Bar, Bubble, Combination, Funnel, Horizontal Bar, Line, Pie, Scatter, Spark NBox	Move focus and multi-select nearest bar, dot, or bubble to the left or right of the current selection. If the focus is on a pie slice in a pie chart, move focus and multi-select previous series in a counterclockwise direction or next series in a clockwise direction. If the focus is on a series in a stacked bar chart, move focus and multi-select the nearest series to the left or right of the selected series. Move focus and multi-select nearest NBox node left or right.
Shift+Up Arrow Shift+Down Arrow	Charts: Area, Bar (Stacked), Bubble, Combination, Funnel, Horizontal Bar, Line, Scatter, Spark NBox	Move focus and multi-select nearest bar, dot, or bubble above or below the current selection. Move focus and multi-select the nearest NBox node up or down.
Home	Hierarchy Viewer - nodes	Move focus to first node in the current level.
End	Hierarchy Viewer - nodes	Move focus to last node in the current level.
Ctrl + Home	Hierarchy Viewer - nodes	Move focus and select the root node.
<	Hierarchy Viewer - nodes	Switches to the active node's previous panel.
>	Hierarchy Viewer - nodes	Switches to the active node's next panel.
Ctrl + Enter	Hierarchy Viewer - nodes	Toggle the display of the children of the active node.

Table 1-9 (Cont.) Shortcut Keys Assigned to ADF Data Visualization Components

Shortcut Key	Components	Function
Ctrl + /	Hierarchy Viewer - nodes	Synchronize all nodes to display the active node's panel.
Ctrl+Shift+^	Hierarchy Viewer - nodes	Go up one level.
Ctrl+/	Hierarchy Viewer - nodes	Switch content panel.
Ctrl+Alt+0	Hierarchy Viewer - diagrams	Center the active node and zoom the diagram to 100%.
Tab	Hierarchy Viewer - nodes Pivot table Pivot filter bar NBox	Move focus through elements. From a component outside an NBox, move focus from NBox, to legend, and then to next component. Use Shift+Tab to move focus to legend, to NBox, and then to previous component.
Esc	Hierarchy Viewer - nodes NBox	Return focus to the containing node. If the focus is on search panel, close the panel. Close the Detail window, if it appears while hovering over a node. Drill up NBox cell or category node.
Spacebar	Hierarchy Viewer - nodes Pivot table Pivot filter bar	Select the active node. Press Ctrl+Spacebar to toggle selection of the active node, and for selecting multiple nodes.
Enter	Hierarchy Viewer - nodes Pivot table Pivot filter bar NBox	Isolate and select active node. Press Shift+Enter to toggle the state of the node. Drill down NBox category node.
/	Hierarchy Viewer - nodes	Toggle control panel state.
[	NBox	Move focus and selection to the first node in the cell or container.
]	NBox	Move focus and selection from the node to the parent container.
Ctrl+F	Hierarchy Viewer - nodes	If the ADF hierarchy viewer component is configured to support search functionality, open the search panel.
Ctrl+Alt+1 through Ctrl+Alt+5	Hierarchy Viewer - nodes	Switch diagram layout.

Table 1-9 (Cont.) Shortcut Keys Assigned to ADF Data Visualization Components

Shortcut Key	Components	Function
Shift+Alt+Arrow keys	Pivot table Pivot filter bar	Change the layout by pivoting a row, column, or filter layer to a new location. Use Shift+Alt+Arrow keys to perform the following: • Provide visual feedback, showing potential destination of the pivot operation, if the header layer is selected • Select different destination locations. • Moving or swapping the selected header layer to the specified destination.

Some ADF Data Visualization Components provide some common functions to the end user through menu bar, toolbar, context menu, or a built-in Task Properties dialog box. You may choose to show, hide, or replace these functionality. If you hide or replace any functionality, you must provide alternate keyboard accessibility to those functions.

Shortcut Keys for Calendar Component

The Calendar component has several views: Day view, Week view, Moth view, and List view.

Table 1-10 lists shortcut keys assigned to the Calendar component.

Table 1-10 Shortcut Keys Assigned to Calendar Component

Shortcut Key	Components	Function
Tab Shift+Tab	Calendar	Move focus. If the focus is on the calendar toolbar, move focus through Day, Week, Month, List, Forward button, Backward button, and Today button. In the day view, move focus through activities of the day. In the week view and month view, move focus through the Month Day header labels only. Use Arrow keys to navigate through activities, "+n more links", and Month Day header labels. In the month view, if the focus is on a Month Day header label at the end of the week, move focus to the Month Day header label of the following week. In the list view, move focus to the day, and then through the activities of the day.

Table 1-10 (Cont.) Shortcut Keys Assigned to Calendar Component

Shortcut Key	Components	Function
Arrow Left Arrow Right	Calendar	Move focus. In the day view, Right and Left arrows do not move focus. In the week view, if the focus is on an activity, move focus to the first activity of the previous or next day. If the previous or next days contain no activities, move focus to the day header. In the month view, the following interaction occurs: • If the focus is on a Month Day header label, move focus to the previous or next day label. If the focus is on the label of the last day of the week in the first week of the month, Right Arrow moves focus to the label of the first day of the week in the second week of the month. If the focus is on the label of the last day of the month, the Right Arrow does nothing. • If the focus is on an activity, move focus to the next activity of the previous or next day. If the previous or next day does not contain any activities, move focus to the Month Day label. If focus is on an activity in the last day of a week, the Right Arrow does nothing. • If the focus is on a "+n more" link, move focus to the next "+n more" links, if they exist. If adjacent "+n more" links do not exist, move focus to the last activity of the day. If the "+n more" link resides in a day at the beginning or end of the week, the Left or Right Arrow do nothing.

Table 1-10 (Cont.) Shortcut Keys Assigned to Calendar Component

Shortcut Key	Components	Function
Arrow Up Arrow Down	Calendar	Move focus. In the day view, move focus through activities. When activities conflict and appear within the same time slot, the Down Arrow moves focus right and the Up Arrow moves focus left. In the week view, move focus through activities of the day. If the focus is on the first activity of a day, the Up Arrow moves focus to the day header. If the focus is on the day header, the Down Arrow moves focus to the first activity of that day. If the day has no activities, the Down Arrow does nothing. In the month view, move focus through activities in a day. • If the focus is on the first activity in a day, the Up Arrow moves focus to the Month day header label. • If the focus is on the Month Day header label, the Up Arrow moves focus to the last activity of the day above it. • If the focus is on the last activity on a day in the last week of the month, the Down Arrow does nothing. • If the focus is on the month header day label in the first week of the month, the Up Arrow does nothing.
Ctrl+Alt+M	Calendar	Launch context menu. You can also launch context menu by pressing Ctrl+Alt+B.

> **Note:**
>
> When using arrows to navigate through activities of a month or week, all-day activities get focus only when the user is navigating within a day, which an all-day activity starts on. Otherwise, all-day activities are skipped.

Default Cursor or Focus Placement

When a user opens an input form (built with ADF Faces) to enter some data, the default cursor is placed on the most suitable component so that the user can proceed with the help of a keyboard instead of using a mouse first. After the initial focus is set, the user can take control on cursor position.

The default cursor puts the initial focus on a component so that keyboard users can start interacting with the page without excessive navigation.

Focus refers to a type of selection outline that moves through the page when users press the tab key or access keys. When the focus moves to a field where data can be entered, a cursor appears in the field. If the field already contains data, the data is highlighted. In addition, after

using certain controls (such as a list of values (LOV) or date-time picker), the cursor or focus placement moves to specific locations predefined by the component.

During the loading of a standard ADF Faces page, focus appears on the first focusable component on the page — either an editable widget or a navigation component. If there is no focusable element on the page, focus appears on the browser address field.

When defining default cursor and focus placement, you should follow these guidelines:

- ADF Faces applications should provide default cursor or focus placement on most pages so that keyboard users have direct access to content areas, rather than having to tab through UI elements at the top of the page.
- You can set focus on a different component than the default when the page is loaded. If your page has a common starting point for data entry, you may change default focus or cursor location so that users can start entering data without excessive keyboard or mouse navigation. Otherwise, do not do this because it makes it more difficult for keyboard users (particularly screen reader users) to orient themselves after the page is loaded.

The Enter Key

The Enter key in an ADF Faces application causes a command line, form, or dialog box to operate its default function. It is typically used to finish an input form and begin the desired process.

The Enter key triggers an action when the cursor is in certain fields or when focus is on a link or button. You should use the Enter key to activate a common commit button, such as in a Login form or in a dialog.

Many components have built-in actions for the Enter key. Some examples include:

- When focus is on a link or button, the Enter key navigates the link or triggers the action.
- When the cursor is in a query search region, quick query search, or Query-By-Example (QBE) field, the Enter key triggers the search.
- When in a table, pressing the Enter key triggers one of the following actions:
  - Clicks on an editable cell and presses Enter key—the focus moves to the editable cell below in the same column in the next row
  - Clicks on an editable cell, edits the contents of the cell, and presses Enter key —the focus completes the action in the current cell and moves to the editable cell below in the same column in the next row
  - Clicks on an editable cell and presses Tab key without editing the cell, once or more than once, and then presses Enter key— the focus moves to the editable cell below in the next row, in the same column where the user started pressing the tab key.
  - Clicks on an editable cell, edits the contents of the cell, and presses Tab key, once or more than once, then presses Enter key—the focus completes the action in the current cell where the user started pressing the Tab key. Focus traverses through the cells sequentially per the number of times the Tab key is pressed. Then, the focus moves to the editable cell below in the next row, in the same column where the user started pressing the tab key

- Clicks on a non-editable cell and presses Enter key—the focus moves to the first editable cell in the next row.
- Clicks on a non-editable cell and presses Tab key, once or more than once, then presses Enter key—the focus traverses through the cells sequentially per the number of times the Tab key is pressed and moves to the first editable cell in the next row.
- Clicks on a cell that contains any command, such as menu, link, or a dialog box, then presses Enter key—the default action for that command is executed

> **Note:**
>
> When user uses the Tab key to traverse through the cells sequentially and presses Enter key to move to the next row , a navigation pattern is formed based on the first set of Tab keys, which is followed in subsequent rows. The navigational pattern is not recognized if arrow keys are used to navigate from one cell to another.

Configuring WebCenter Content Web Services for Integration

This chapter describes how to use Oracle WebCenter Content web services and Oracle WebLogic Server web services to integrate a client application with Content Server.

This section includes the following sections:

- About Configuring WebCenter Content Web Services for Integration
- Configuring Web Service Security Through Web Service Policies
- Configuring SAML Support

For general information about web services that you can use with Content Server, see *Overview of Web Services* in *Developing with Oracle WebCenter Content*.

The way to use web services described in this chapter was introduced in Oracle Universal Content Management 11g. If you want to use the way introduced in Oracle Universal Content Management 10g, with Web Services Definition Language (WSDL) and SOAP (Simple Object Access Protocol) files and the WSDL generator, see *Configuring Web Services with WSDL, SOAP, and the WSDL Generator* in *Developing with Oracle WebCenter Content*.

About Configuring WebCenter Content Web Services for Integration

WebCenter Content web services work with Oracle WebLogic Server web services to perform management functions for Content Server. Oracle WebLogic Server web services provide SOAP capabilities, and WebCenter Content web services include several built-in SOAP requests. WebCenter Content web services are automatically installed with Content Server, but they require additional configuration to set up security.

Technologies for Web Services

The core enabling technologies for WebCenter Content web services follow:

- SOAP (Simple Object Access Protocol) is a lightweight XML-based messaging protocol used to encode the information in request and response messages before sending them over a network. SOAP requests are sent from WebCenter Content web services to

Oracle WebLogic Server web services for implementation. For more information about SOAP, see *Simple Object Access Protocol (SOAP)* at http://www.w3.org/TR/soap12.

- Web Services Security (WS-Security) is a standard set of SOAP extensions for securing web services for confidentiality, integrity, and authentication. For WebCenter Content web services, WS-Security is used for authentication, either for a client to connect to the server as a particular user or for one server to talk to another as a user. For more information, see the OASIS Web Service Security page at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.
- Web Service Policy (WS-Policy) is a standard for attaching policies to web services. For WebCenter Content web services, policies are used for applying WS-Security to web services. The two supported policies are `username-token` security and SAML security.

  Historically, Oracle used Oracle Web Services Manager (Oracle WSM) to secure its web services, and Oracle WebLogic Server used Web Services Security Policy (WS-SecurityPolicy) to secure its web services. Because web services security is partially standardized, some Oracle WSM and WS-SecurityPolicy policies can work with each other.

  > **Note:**
  >
  > Use Oracle WSM policies over Oracle WebLogic Server web services whenever possible. You cannot mix your use of Oracle WSM and Oracle WebLogic Server web services policies in the same web service.

  WebCenter Content web services (`idcws/` as context root) are SOAP based, while WebCenter Content native web services (`idcnativews/` as context root) are JAX_WS based. Both kinds of web services can be assigned Oracle WSM policies through the Oracle WebLogic Server Administration Console.

  The generic WebCenter Content web services are JAX-WS based and can be assigned Oracle WSM policies and managed by Oracle WSM. The native WebCenter Content web Services are SOAP based and can only support WS-Policy policies managed through the Oracle WebLogic Server Administration Console.

  For more information about Oracle WSM, see the *Overview of Web Services Administration* in *Administering Web Services*.

  A subset of Oracle WebLogic Server web services policies interoperate with Oracle WSM policies. See *Overview of OWSM Interoperability* in *Interoperability Solutions Guide for Oracle Web Services Manager*.

  Web Services Security Policy (`WS-SecurityPolicy`) is a set of security policy assertions for use with the WS-Policy framework. For more information, see the Web Services Security Policy specification at http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html.
- SAML is an XML standard for exchanging authentication and authorization between different security domains. For more information, see the Security Assertion Markup Language (SAML) specification at http://docs.oasis-open.org/security/saml/v2.0/.

- WebLogic Scripting Tool (WLST) is a command-line tool for managing Oracle WebLogic Server. For more information, see *WebCenter Portal Custom WLST Commands* in *WebCenter WLST Command Reference*.

WebCenter Content Web Services

WebCenter Content provides two types of web services: a general (generic) JAX-WS based web service, and a native SOAP based web service. The two types of web services reside in two different context roots. The *context root* is the primary identifier in the URL for accessing the web services.

The context roots follow:

- `idcws`

  Use this context root for general access to Content Server through any regular web services client.

- `idcnativews`

  The Remote Intradoc Client (RIDC) uses the native web services. Oracle recommends that you *do not* develop a custom client against these services. For more information about RIDC, see *Using RIDC to Access Content Server* in *Developing with Oracle WebCenter Content*.

The following table describes the WebCenter Content web service in the `idcws` context root.

WebCenter Content Web Service	Descriptions
`GenericSoapService`	This service uses a generic format similar to HDA for its SOAP format. It is almost identical to the generic SOAP calls that you can make to Content Server when you set `IsSoap=1`. For details of the format, see the published WSDL at `idcws/GenericSoapPort?WSDL`. You can apply WS-Security to `GenericSoapService` through WS-Policy. Content Server supports Oracle WSM policies for SAML and `username-token`. As a result of allowing WS-Security policies to be applied to this service, streaming Message Transmission Optimization Mechanism (MTOM) is not available for use with this service. Very large files (greater than the memory of the client or the server) cannot be uploaded or downloaded. `GenericSoapService` automatically has `oracle/wsmtom_policy` applied to it. Content Server cannot accept SOAP requests that have this policy applied. For `GenericSoapService` to work, the following policy must be applied to it: `oracle/no_mtom_policy`

The following table describes the WebCenter Content web services in the `idcnativews` context root.

WebCenter Content Web Services	Descriptions
`IdcWebRequestService`	This is the general WebCenter Content service. Essentially, it is a normal socket request to Content Server, wrapped in a SOAP request. Requests are sent to Content Server using streaming Message Transmission Optimization Mechanism (MTOM) in order to support large files. Streaming MTOM and WS-Security do not mix. As a result, do not apply WS-Security to this service because it will break the streaming file support. In order to achieve security, you must first log in using the `IdcWebLoginService`, then use the same JSESSIONID received from that service in the next call to `IdcWebRequestService` as a cookie.
`IdcWebLoginService`	This service is solely for adding security to `IdcWebRequestService` calls. There are no parameters for this service; it simply creates a session. The important field to retrieve is the `JSESSIONID` value for future calls to `IdcWebRequestService`. If you want to use WS-Security with `IdcWebRequestService`, then apply it here. Content Server supports Oracle WSM policies for SAML and `username-token`.

Configuring Web Service Security Through Web Service Policies

The WebCenter Content web services are installed and ready to use by default with the WebCenter Content EAR. However, unless you configure web service security (WS-Security) on any of the WebCenter Content web services, all connections to Content Server will use the `anonymous` user. To configure security for WebCenter Content web services, you configure WS-Security through WS-Policy. Additional configuration is required to enable authentication.

WS-Security is set through the use of web service policies (WS-Policy). Security policies can be set for web services to define their security protocol. In particular, the WebCenter Content web services support Oracle WSM policies.

> **Note:**
>
> `GenericSoapService` automatically has `oracle/wsmtom_policy` applied to it. Content Server cannot accept SOAP requests that have this policy applied. For `GenericSoapService` to work, the following policy must be applied to it:
>
> `oracle/no_mtom_policy`

WebCenter Content supports the following Oracle WSM policies:

- `oracle/wss11_saml_token_with_message_protection_service_policy`
- `oracle/wss11_username_token_with_message_protection_service_policy`
- `oracle/wss_username_token_service_policy`

The 12c 12.2.1.3.0 uses Weblogic Server to apply Oracle WSM policies to web services. For more information, see *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

Configuring SAML Support

You can also provide SAML support for client-side certificate authentication.

See:

- *Securing Inbound SOAP Requests Using SAML Message Protection* in *Use Cases for Securing Web Services Using Oracle Web Services Manager*.
- *Configuring Message Protection for Web Services* in *Web Services and Managing Policies with Oracle Web Services Manager*

Using Approval Management

Get an overview of the approval management extensions that are available for the human workflow services of Oracle SOA Suite. The human workflow service handles all interactions with users or groups who participate in the business process by creating and tracking tasks for the appropriate users in the organization.

Users typically access tasks through a variety of clients, including Oracle BPM Worklist, email, portals, or custom applications. Approval management extensions enable you to define complex task routing slips for human workflow by taking into account business documents and associated rules to determine the approval hierarchy for a work item. Additionally, approval management extensions let you define multi-stage approvals with associated list builders based on supervisor or position hierarchies. You define the approval task in the Human Task Editor of Oracle JDeveloper, and associate the task with a BPEL process.

For more information about human tasks, see the chapters in *Using the Human Workflow Service Component* in *Developing SOA Applications with Oracle SOA Suite*.

Introduction to Approval Management

Approval Management extensions (AMX) extend human workflow services with complex approval patterns. It serves as a sophisticated “Assignment Manager" for human workflow.

Some of the key workflow features include:

- Declarative modeling of approval management processes.
- The ability to define complex multi-stage approval with static and dynamic approval list.
- A Workflow Editor to define task parameters, assignment and routing policies, escalation and expiration settings, and notification settings.
- Policy-based task assignment, which allows users to define approval rules based on business documents.
- The ability to design a task form to render contents of the approval task and associated task operations.
- The ability to define email and instant messaging (IM) notifications for various participants in the workflow.
- A web-based worklist application for task assignees, process owners, and administrators.

- The ability to look up users and roles in various user directories, including Oracle Internet Directory, LDAP, and third-party directories.

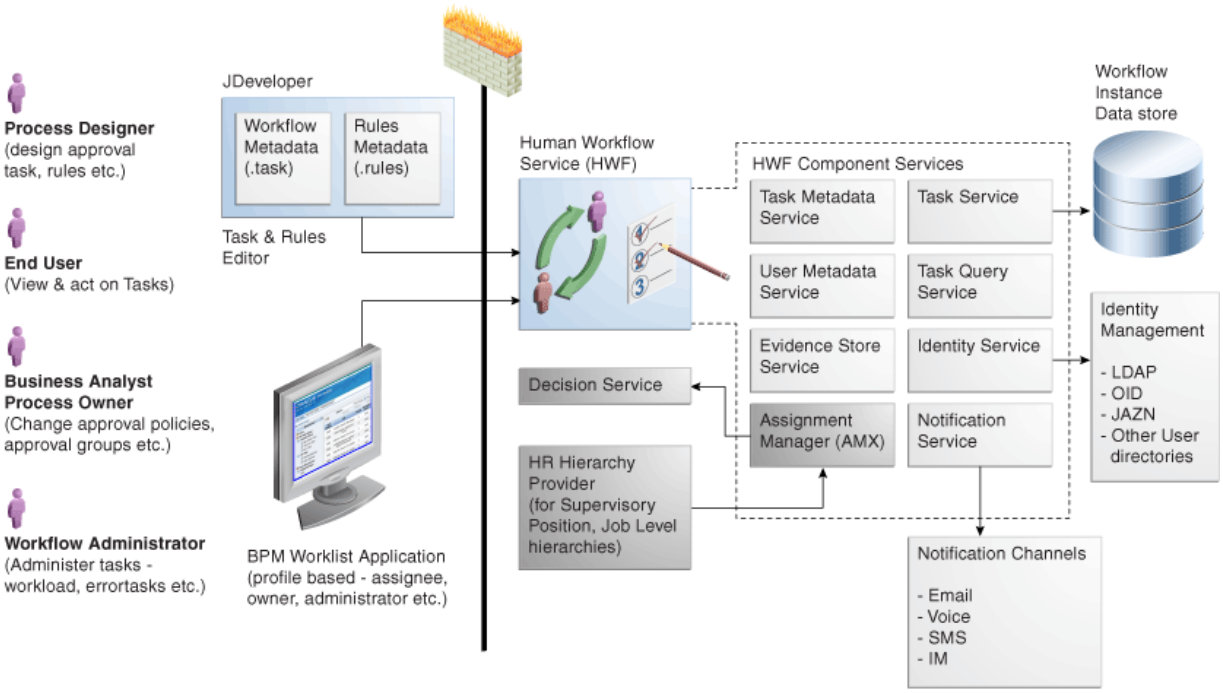
AMX provides the following additional features:

- Attributes derived from ADF view object in transactional applications.
- The ability to retrieve various job, position, and supervisory hierarchies from HR systems using hierarchy provider plug-ins.
- The ability to define rules for controlling approval lists and hierarchy configurations.

AMX Components

The following figure shows the key AMX and human task integration components. These components are described in subsequent sections of this chapter.

Figure 1-5 Overall Architecture



The human workflow service enables users to model human interactions as part of a business process. The human workflow service handles requests based on task and rules metadata. It consists of the following set of core services:

- Task service
- Task query service
- User metadata service
- Task metadata service
- Identity service

- Notification service
- Assignment manager

These services are described in detail in *Introduction to Human Workflow Services* in *Developing SOA Applications with Oracle SOA Suite*. AMX serves as a sophisticated assignment manager within human workflow allowing you to model complex approval patterns based on business rules.

The core components required for approval management include the following:

- **Human Task Editor in JDeveloper**

  This task editor is used to define the metadata for a human task and the routing slip. The task editor lets you define such things as task parameters, outcomes, expiration and escalation, and notification settings. Some of the components added by AMX include the ability to do the following:

  – Define multi-stage approvals and associated approval list builders in JDeveloper.

  – Determine the approval hierarchy based on business documents (ADF objects) and business rules. This is done through Rules Designer in JDeveloper

- **Human workflow services**

  Some of the key services that are required for handling complex approvals include the following:

  – Task Service - Responsible for creating and managing tasks in the dehydration store

  – Identity Service - Responsible for authentication and authorization of users and groups. The service can look up various user directories for authorization and contact information for users.

  – Task Query Service - Responsible for retrieving tasks for the web-based worklist application

  – Decision Service - Responsible for executing business rules related to approvals

- **Oracle BPM Worklist**

  Oracle BPM Worklist is a web-based application that lets users access tasks assigned to them and perform actions based on their roles in the approval process. Oracle BPM Worklist supports the following profiles:

  – Work assignee - An end user who is assigned a task. These users can view tasks assigned to them and perform actions, and also can define custom views and define routing rules for their tasks.

  – Process owner - Typically a business analyst responsible for managing certain types of approvals. These users can manage tasks for the processes they own, define approval groups, and change approval policies

  – Workflow administrator - Typically a system administrator responsible for managing errored tasks, and administering and monitoring work queues. This user also may use Oracle Enterprise Manager to monitor the health of the workflow services.

Understanding Approval Management Concepts

AMX extends human workflow services with additional functionality to handle complex approval patterns.

Some human workflow concepts with which you must be familiar are the following:

- Human Task Editor in JDeveloper
- Task metadata (task parameters, allowed operations, and patterns) and routing slip
- ADF task flow based on task forms
- Oracle BPM Worklist

These concepts are described in the chapters in *Using the Human Workflow Service Component* in *Developing SOA Applications with Oracle SOA Suite*.

Task

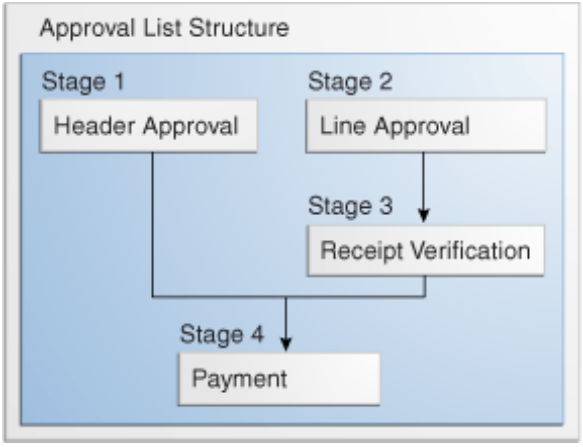
A task handles approvals. A different task is created for each approval requirement based on the business served by it. For example, an approve new expense report task or an approve new purchase order task.

Some of the standard metadata for a task include the following:

- Task attributes such as title, outcomes (approve, reject, and so on) priority, expiration and others
- Task parameters that may be based on simple primitive types, XML elements, or external entities such as ADF view objects
- A complex approval task that may include one or more stages to identify the key milestones within the approval sequence. For more information see Stages.
- Expiration and escalation policy
- Notification settings for notifying various participants
- *List builders* within stages, which are based on names and expression, management chain, supervisory, position, job-level hierarchy, or approval groups. For more information, see List Builders.
- *Approval task configurations*, including policies for substitution and modification of approvers, configuration of self-approval, and repeated approvers. For more information, see Task.

  The following figure shows the various stages in a sample approval pattern.

Figure 1-6 Approval List Structure



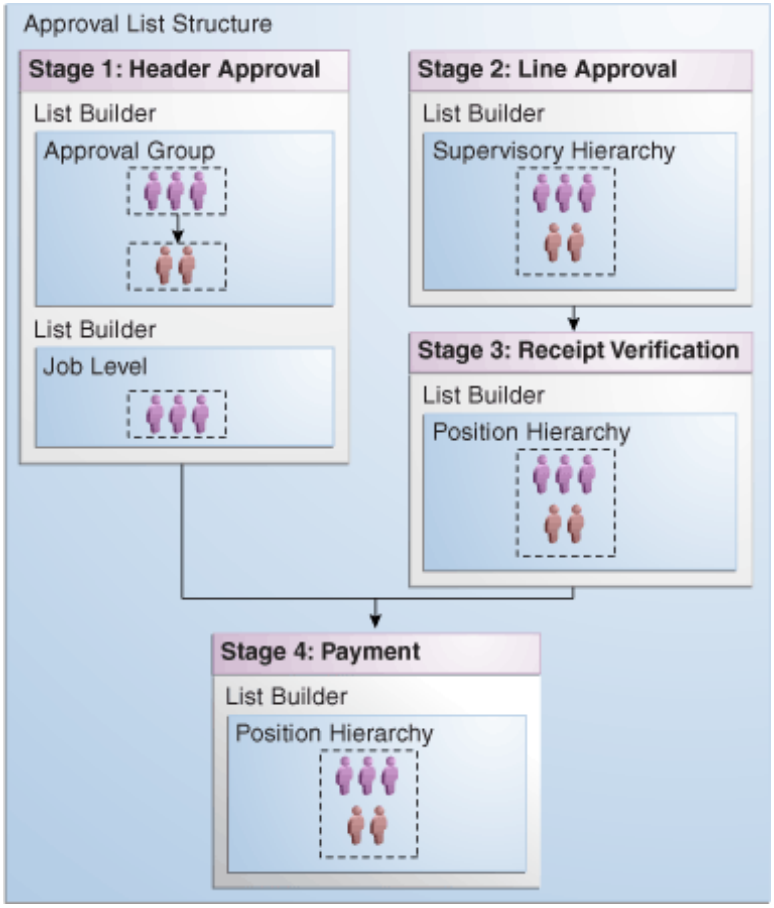
The approval pattern consists of four stages:

- Header approval
- Line approval
- Receipt verification
- Payment

Header approval runs in parallel with line approval and receipt verification. After these stages run, the payment stage runs.

Each of the four stages has list builders. Multiple list builders in a stage can run in serial or parallel to one another. One or more approvers can exist within each list builder. The following figure illustrates these concepts.

Figure 1-7 Stages and Their List Builders



These concepts are described in the sections that follow.

Service Data Objects

ADF Business Components objects can be exposed easily as Service Data Objects (SDOs) through the service interface. This provides a flexible way to accept business entities. Subsequently, supporting SDOs natively enables accepting multiple business entities. This also lays the foundation for future Flexfield SDO support. Since an SDO is a structured XML, you can pass it in as static XML through the task payload.

A collection is defined in an entity parameter for the task. It enables access to a portion of the business entity as an XML fragment retrieved by an XPATH expression. Keys allow us to identify the primary keys in this fragment.

An entity parameter is the definition that tells us how to access an SDO or a static XML. An entity parameter captures the following information for an SDO:

- Identity of a reference in the overall SCA process, including the Web service definition language (WSDL) for the SDO web service
- Method to invoke
- Input message to the web service
- Output message to the web service
- Collections

An entity parameter captures the following information for a static XML:

- XSD for the static XML
- Collections

For example, an expense voucher can have hierarchical groupings of header, lines, and cost centers. For approval policy purposes, you may only define a collection on header and lines if these are the only components required for determining the set approvers. It is not necessary to map as collections those parts of the business document that are not necessary to define rules.

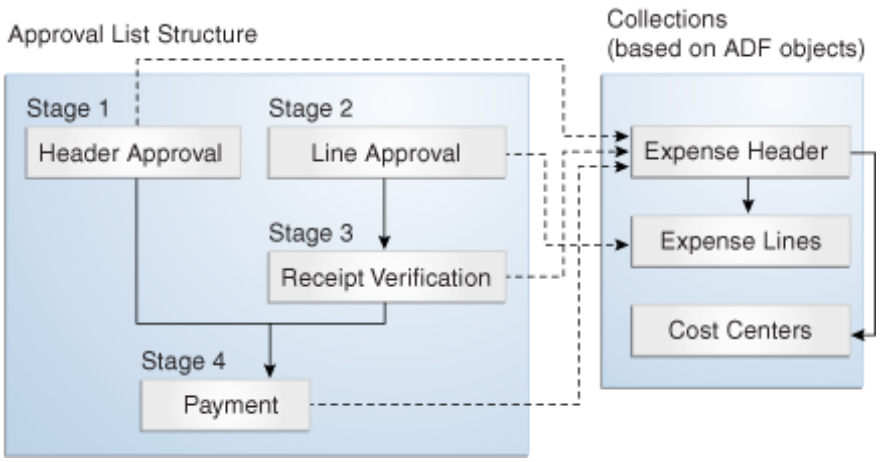
For more information, see *Implementing Business Services with Application Modules* and *Creating SOAP Web Services with Application Modules* in *Developing Fusion Web Applications with Oracle Application Development Framework*.

Stages

A stage is a set of approvals related to a collection. The same collection can be associated with multiple approval stages.

The following figure illustrates the mapping of stages and collections.

Figure 1-8 Mapping of Stages and Collections



Each approval stage is associated with a collection. In the figure, there are four stages in the approval.

- **Header Approval** is associated with the Expense Header collection.
- **Receipt Verification** is associated with the Expense Header collection.
- **Payment** is associated with the Expense Header collection.
- **Line Approval** is associated with the Expense Lines collection.

A compound approval may consist of multiple stages and then can be modeled in serial or parallel with each other. Each stage consists of list builders to determine the list of approvers.

Optionally, each list builder can be associated with an approval policy, that is, a set of rules. At runtime, the appropriate set of approvals are returned based on the list builders used within the stage and on the associated policies.

List Builders

As described in Stages, each approval stage consists of list builders to determine the actual list of approvers. The following list builders are supported.

- **Names and Expressions**

  Enables you to construct a list using static names, or names coming from XPath expressions.

- **Approval Groups**

  Includes predefined approver groups in the approver list. Approval groups can be static or dynamic.

- **Job Level**

  Ascends the supervisory hierarchy, starting at a given approver and continuing until an approver with a sufficient job level is found.

- **Position**

  Ascends the position hierarchy, starting at a given approver's position and continuing until a position with a sufficient job level is found.

- **Supervisory**

  Ascends the primary supervisory hierarchy, starting at the requester or at a given approver, and generates a chain that has a fixed number of approvers in it.

- **Management Chain**

  Enables you to construct a list based on management relationships in the corresponding user directory.

  The management chain participant type only supports parallel routing when the first assignee in the management chain is a single user. You cannot specify parallel participants such as a set of users or a group, as the initial assignees in the management chain.

- **Rule-based**

  Enables you to model rules that return different list-builder types based on different conditions. For example, if you model a supervisory list builder with rules, the rule can return only the supervisory list builder. If you model a rule-based list builder, the rule can return different list builder types.

> **Note:**
>
> The Approval Groups, Job Level, Position, and Supervisory list builders are specific to AMX, and are described in detail in How to Model and Configure List Builders.
>
> For information about the Names and expressions, Management Chain, and Rule-based list builders, see *Creating a Single Task Participant List* in *Developing SOA Applications with Oracle SOA Suite*.

Task Operations

Most of the standard human task operations also are available on AMX-based tasks. Some of the common operations include the following:

- **User-defined outcomes** - Business outcomes, such as "Approve" and Reject," that are associated with a task. When a user performs these types of actions, the task is removed from the user's "Inbox" and is marked as completed or moved to the next approver.
- **Delegate** - Allows a user to assign a task to another person or role to act on his or her behalf.
- **Escalate** - Allows a user or an administrator to escalate a task to the user's supervisor.
- **Reassign** - Allows users to transfer a task to another user. From that point on, the new user's hierarchy is used for supervisor or other organization-based approvals.
- **Withdraw** - Allows the task initiator or administrator to cancel or withdraw the task after the approval has started.
- **Request for Information** - Allows a task approver to request information from any prior participant or the task initiator.
- **Pushback** - Allows the task approver to push back the task to the previous approver to review it again.
- **Adhoc Insertions** - Allows any task assignee to insert approvers in the generated approval list.

> **Note:**
>
> The position list builder does not allow the approver to reassign, delegate, escalate or perform adhoc insertions.

For a complete list of actions, see *Acting on Tasks: The Task Details Page* in *Developing SOA Applications with Oracle SOA Suite*.

Business Rules for Approval

Approvers of a task can be defined either inline in a task definition or by using business rules to specify the list builders that identify the actual approvers of a task. In addition, you can use business rules to specify approver substitution and list

modifications. These rules are defined with the help of Oracle Business Rules and can vary between organizations. Typically, however, they are defined by the customer.

Business rules are a combination of conditions and actions. Optionally, priority and validity periods can be defined for these rules. In Human Workflow rules, rule conditions are defined using fact types that correspond to the task, and to the task message and entity attributes (which are XML representations of SDO objects). Rule actions consist of approver list builders and their parameters. Approver list builders move up a particular hierarchy and construct or modify the approver list according to the parameters defined. Approver list builders are implemented as XML (JAXB) fact types.

For more information about these concepts, see *Using the Business Rules Service Component* in *Developing SOA Applications with Oracle SOA Suite*.

List Creation

A list creation policy includes rule conditions and actions that create the list builders.

The following example rules illustrate the configuration of the Supervisory list-builder parameters that create an approver list based on an SDO-based fact type.

For more information, see How to Create Lists.

Example 1-1 Rule 1

```
IF
ExpenseItems.ReceiptAmount < 200
THEN
call CreateSupervisoryList( levels:1,
startingPoint:HierarchyBuilder.getPrinicipal("jstein",-1,"",""),
uptoApprover:HierarchyBuilder.getPrinicipal("wfaulk",-1,"",""),
autoActionEnabled:false,autoAction:null,
responseType:ResponseType.REQUIRED,ruleName:"Rule_1",lists:Lists)
```

Example 1-2 Rule 2

```
IF
xpenseItems.ReceiptAmount >= 200
THEN
call CreateSupervisoryList( levels:1,
startingPoint:HierarchyBuilder.getPrinicipal("wfaulk",-1,"",""),
uptoApprover:HierarchyBuilder.getPrinicipal("cdickens",-1,"",""),
autoActionEnabled:false,autoAction:null,
responseType:ResponseType.REQUIRED,ruleName:"Rule_2",lists:Lists)
```

Approver Substitution

Users, groups, and application roles appearing in a list can be substituted using list substitution. List substitution is available from Rules Designer and does not require any configuration in JDeveloper.

The following example rule illustrates approver-substitution usage.

This rule implies that if the expense item amount is less than 4000, then substitute approver "jcooper," if present in the approver list, with approver "jstein."

For more information, see How to Make Approver Substitutions.

Example 1-3 Approver-Substitution Usage

```
IF
ExpenseItems.ReceiptAmount < new BigDecimal(4000)
THEN
call Substitute(fromId:"jcooper", toId:"jstein", ruleName:"Substituted",
substitutionRules: SubstitutionRules)
```

List Modification

Job Level and Position lists can be extended or truncated from rules. List modification is applied after list creation.

The following example rule illustrates list-modification usage.

This rule implies that if the expense item amount is greater than 3000, and if the final approver in the approver list is of Job Level 3, then extend the approver list by at least two relative levels.

For more information, see How to Make List Modifications.

Example 1-4 List-Modification Usage

```
IF
ExpenseItems.ReceiptAmount > new BigDecimal(3000)
THEN
Call Extend(ifFinalApproverLevel:3, extendBy:2,ruleName:"Modified",lists:Lists)
```

Designing Approval Management Tasks in Oracle JDeveloper

You design approval management tasks by defining a human task that provides the ability to model multi-stage approvals and determine the appropriate approvers based on approval policies for a business object and the associated HR hierarchy provider.

This section describes the overall modeling process and the specifics of the process you use to model approval management tasks in JDeveloper.

Introduction to the Modeling Process

The modeling process for designing approval management tasks includes the following:

- Creating a human task definition
- Creating a task display form using the Human Task Editor

Creating a human task definition includes the following tasks:

- Specifying general information, such as task title and task-title globalization, outcomes, priority, owner, and category
- Specifying task parameters, including those with service data object (SDO) references
- Specifying mapped attributes
- Modeling task routing by specifying stages and list builders, and modeling any business rules that define the list builders
- Defining escalation and renewal policies
- Specifying notification settings

- Modeling any advanced settings like callbacks, security access rules, and restricted assignment

For more information, see the chapters in *Using the Human Workflow Service Component* in *Developing SOA Applications with Oracle SOA Suite*.

Before You Begin

Before designing approval management tasks, you must satisfy the following prerequisites:

- You must have deployed SDO services.
- You must have created a human task service component in which to design the approval task.

Specifying General Information

Some general information, including task title, outcomes, priority, owner, and category, is not specific to AMX.

For more information, see *How to Define the Human Task Activity Title, Initiator, Priority, and Parameter Variables* in *Developing SOA Applications with Oracle SOA Suite*.

Task-Title Globalization

The title attribute of the task object contains a user-friendly value that mainly is descriptive in nature. In AMX, the task title can be globalized so that it renders in the user's preferred language.

Title is defined in the `*.task` file for design time and in the `WorkflowTask.xsd` file for runtime. Currently, the definition of these elements in both of these files are simple xsd:string types. For globalization, the structure and usage of these elements change to accommodate a mechanism that provides translatable, formatted strings.

The design-time metadata for these elements is enhanced to contain a value element and an optional set of parameters. Messages defined as an XPath expression or static have their information stored in the value element and require no parameters. Messages defined that rely on information in a resource bundle have a key stored in the value element with some parameters also defined.

The Human Task Editor provides a mechanism in the Expression Builder to enable the user to specify the resource key and parameters and, at the same time, generate the appropriate design time XML in the taskDefinition.

The following figure shows the globalization icon in the Human Task Editor.

Figure 1-9 Title Globalization Icon

The following procedure explains how to add translatable strings. It assumes that a resource bundle has been specified.

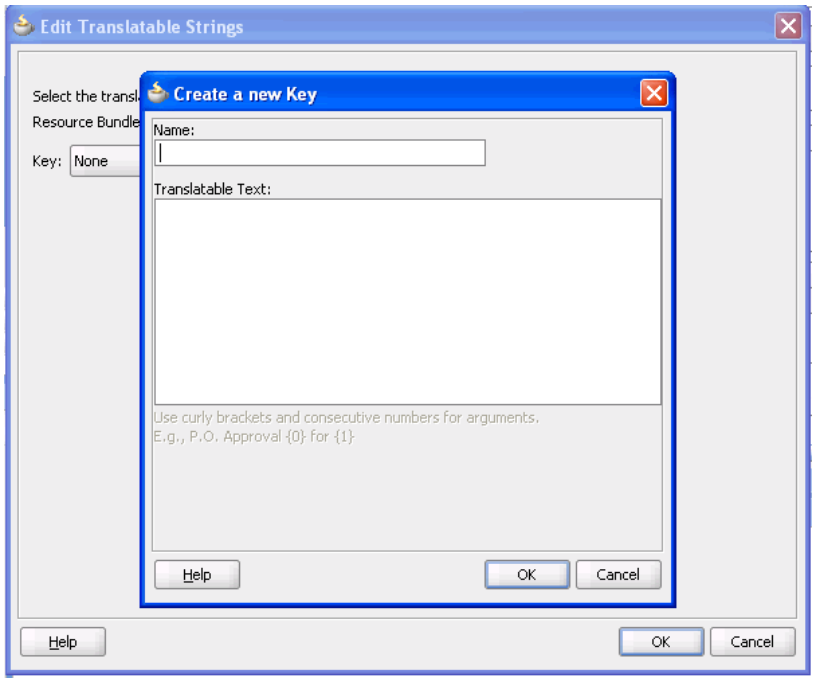
1. Select Translation from the drop-down list.

   The Global icon displays.

2. Click the icon to display the Edit Translatable Strings dialog box.
3. Select a key from the drop-down list or click the plus sign (+) to create one.

   The following Create a New Key dialog box, displays when you click the plus sign (+) on the Edit Translatable Strings dialog box.

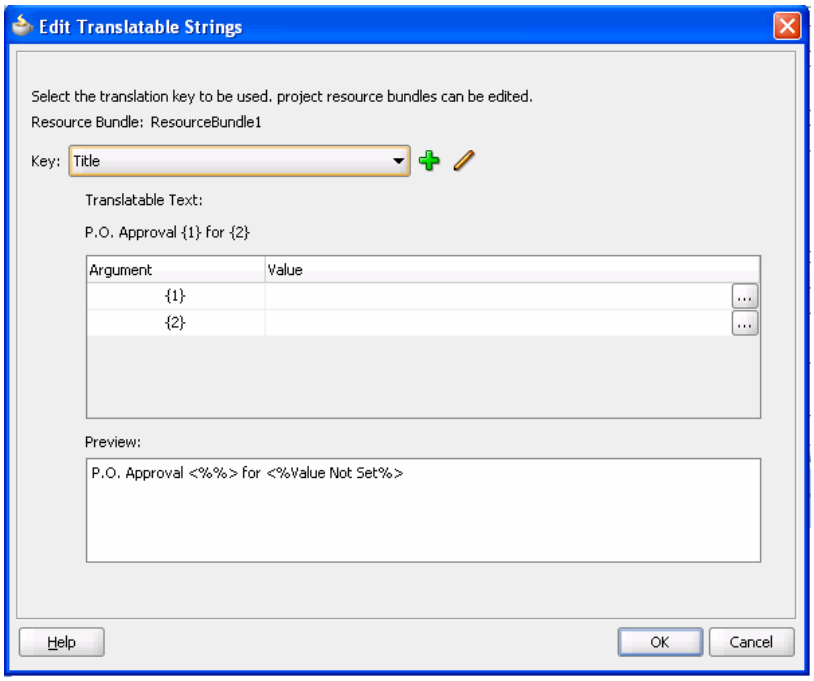
Figure 1-10 Create a New Key Dialog



4. Enter a name, the translatable text, and click **OK**.

   The New Key added dialog box shows the Edit Translatable Strings dialog box after a new key has been added.

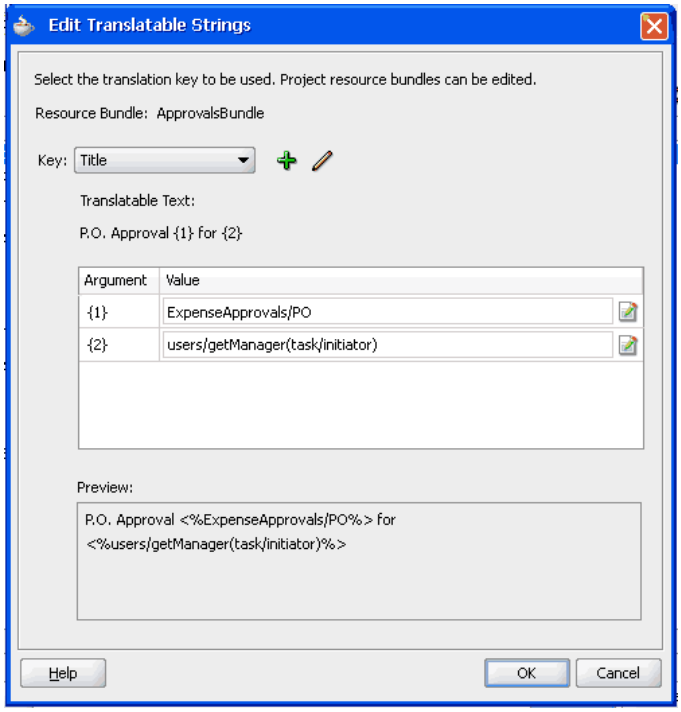
Figure 1-11 New Key Added



5. Use the Expression Builder to add values.

   The Translatable Text and Values dialog box shows the completed Edit Translatable Strings dialog box.

Figure 1-12 Translatable Text and Values



> **Note:**
>
> The title value, or a definition of the title value can be set in two places: in the TaskDefinition XML (`.task`) file, or in the bpel file. When set in the bpel file, this value takes precedence over the definition in the TaskDefinition. However, the value in the bpel file is not translatable.

6. Click **OK** to close the dialog box.

Specifying Task Parameters

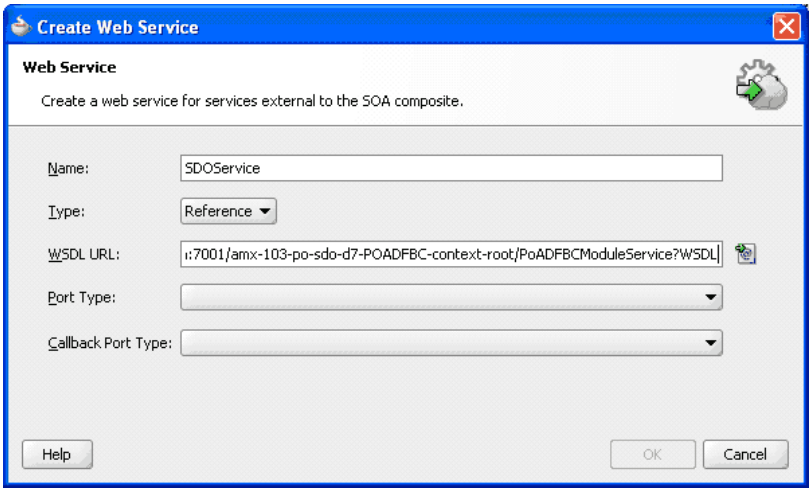
Specifying task parameters includes the following tasks:

How to Create Service Data Object (SDO) References

An SDO service can be invoked from workflow services to retrieve the SDO as XML. This invocation is in the form of a SOA web service call. When the SDO service WSDL URL is available, a web service reference should be added using the Create Web Service dialog box.

To create a reference, enter the WSDL URL and select the port type from the available port types, as shown in the following figure.

Figure 1-13 Web Service Reference



For information about creating SDOs, see topics in *Designing an SDO-Based Enterprise JavaBeans Application* in *Developing SOA Applications with Oracle SOA Suite*.

How to Define Entity Parameters

The following procedure enables you to accept a service data object (SDO).

1. Create a Service reference in the composite.

   This allows Fabric to create all the necessary wiring to a specific URL that points to a WSDL.

2. Define the task payload as external and specify which workflow retrieves the SDO object.

   This creates task parameters representing the input and output to the SDO web service.
3. Choose **Entity**.
4. Select a reference.
5. Set the collection for the stage.
6. Click **OK**.

The following procedure enables you to accept static XML.

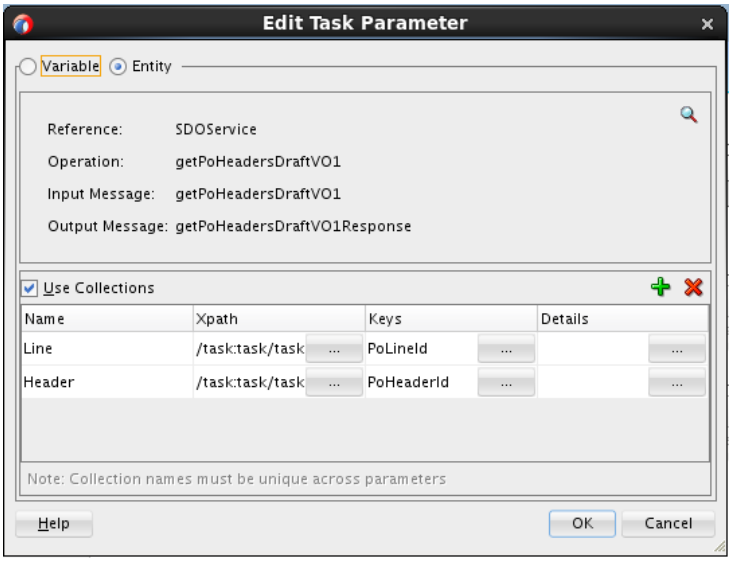
1. Provide the XSD where the schema is defined.
2. Define the task payload parameter as static XML.
3. Define the collection, its XPATH expression, and its keys.
4. Set the collection for the stage.
5. Click **OK**.

How to Define Collections

Collections are references to specific parts of a task message attribute, both static-XML based and entity attributes. After defined, collections can then be associated with stages to identify a stage as acting on a collection.

Defining a collection involves defining the name of the collection and the XPath to the collection element. If the collection is defined for an entity attribute, the keys for the collection element have to be specified as well. Each key has to be a direct child of the collection element. The following figure shows how collections are defined.

Figure 1-14 Defining Collections



When you define a collection, JDeveloper automatically determines if it should be repeating element or not. This information is used when collections are associated with a stage. A non-repeating collection can be associated with a singular stage. A repeating collection, when associated with a stage, repeats the stage in parallel for each element in the collection at

runtime. For information about how the collection information is used in a stage, see How to Model and Configure Stages.

Specifying Mapped Attributes

Human workflow provides task-message attributes that you can use for storing use-case-specific data, such as data extracted from a task's payload. These attributes are also known as *flexfield attributes* or *mapped flexfield attributes*.

Mapped flexfield attributes allow payload values to be displayed as columns in the task listing, rather than being hidden in the task details. These values are stored in the human workflow database schema, and you can use them in queries, view definitions, and assignment rule definitions.

There are two types of message attributes:

- *public* - attributes mapped to specific task components at runtime. These mappings can be changed at any time, and must be re-created when a task component is redeployed. For more information see *Using Mapped Attributes (Flex Fields)* in *Developing SOA Applications with Oracle SOA Suite*.
- *protected* - AMX-specific mappings between a task component and protected flexfield attributes defined at design time. They cannot be changed at runtime, and are deployed along with the task component.

  Table 1-11 summarizes the 60 available protected flexfield attributes.

Table 1-11 Protected Flexfield Attributes

Name	Description
ProtectedTextAttribute1 - ProtectedTextAttribute20	Stores text data, up to 2000 characters. The content in these fields is checked during keyword searches in the Oracle BPM Worklist and through the task-query service.
ProtectedFormAttribute1 - ProtectedFormAttribute10	Stores text data, up to 2000 characters. The content in these fields is **not** checked during keyword searches in the Oracle BPM Worklist.
ProtectedURLAttribute1 - ProtectedURLAttribute10	Stores text data, up to 200 characters. The content in these fields is **not** checked during keyword searches in the Oracle BPM Worklist.
ProtectedDateAttribute1 - ProtectedDateAttribute10	Stores date information.
ProtectedNumberAttribute1 - ProtectedNumberAttribute10	Stores number information.

About Attribute Labels and Attribute-Label Mappings

Attribute labels are user-defined properties that allow a meaningful string to be applied to a particular flexfield attribute. The label should reflect the data to store in the attribute. For example, “CustomerName" for “ProtectedTextAttribute1," “OrderNumber" for “ProtectedNumberAttribute2," or “OrderDate" for “ProtectedDateAttribute1."

A flexfield attribute can have multiple attribute labels defined for it. For example, the attribute “ProtectedTextAttribute1" could have the labels “CusomerName," “PartId" and “EmployeeDepartment".

Attribute-label mappings for protected attributes are defined at design time in the Human Task Editor. They define a mapping between a particular task component and an attribute label, and also specify how the value of the attribute should be populated. The same attribute label can be re-used in multiple mappings. This allows task components to map data having the same semantic meaning into a common attribute identified by a common label.

For example, PurchaseOrder, LoanRequest and ServiceRequest tasks all could define mappings to the “CustomerName" label. By sharing the same attribute labels across multiple task components, it is possible to construct worklist queries that query multiple task types and display or filter values from the common attribute labels. For example, it would be possible to construct a query that selected PurchaseOrder, LoanRequest, and ServiceRequest tasks, and then displayed the “CustomerName" as a column in the worklist task listing.

How to Define Attribute-Label Mappings

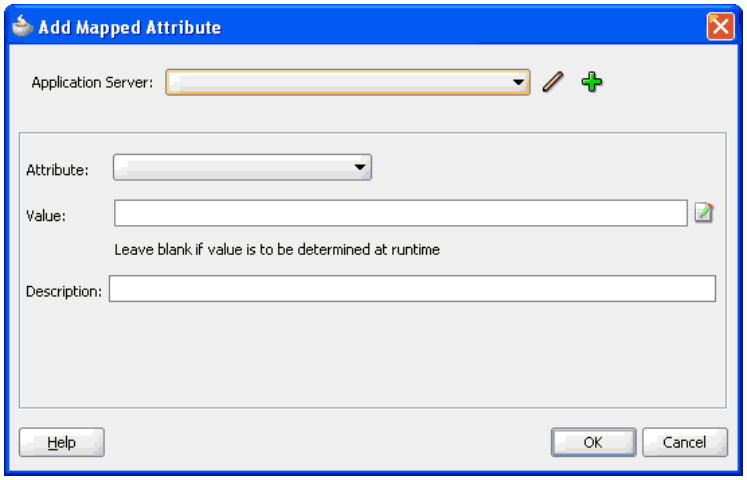
You define attribute-label mappings in the **Mapped Attributes** section of the Human Task Editor, as shown in the following figure.

Figure 1-15 Mapped Attributes Section

Use the following procedure to define attribute-label mappings:

1. Click the **Add** icon to display the Add Mapped Attribute dialog box.

Figure 1-16 Add Mapped Attribute Dialog



2. Perform one of these options:
   - From the drop-down list, select the application server that contains the protected-attribute labels.
   - Click the **Add** icon to create a connection.

- Click the **Edit** icon to edit an existing connection.

The **Attribute** drop-down list populates with the available attribute labels from the specified server.

3. From the drop-down list, select an attribute.

> **Note:**
>
> The list does not include any labels for flexfield attributes to which this task component is being mapped.

4. At the **Value** field, specify a value using one of these options:
   - Enter an XPath expression that determines the value to be stored in the attribute.
   - Click the icon to create a value in the Expression Builder.
   - Leave the field blank to allow the value to be determined at runtime.

   Usually, this XPath expression selects a value from the tasks's payload, but you can specify any valid expression that evaluates to a simple type, such as a string, a date, or a number.

   Be aware that specifying an XPath expression is not mandatory. You may prefer to set the value of the underlying flexfield-attribute value yourself. For example, you can add a custom assign activity to the BPEL process that initiates the task, or manipulate the Task object through the workflow service APIs.
5. Enter a description. This is optional.
6. Click **OK**.

Specifying Routing and Approval Policies

Specifying routing and approval policies includes the following tasks:

How to Model and Configure Stages

Based on functional needs, you can add and arrange multiple stages in a structure that can be a combination of sequential and parallel stages. This section describes how to create sequential and parallel stages.

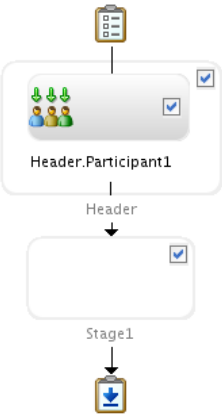
Use the following procedure to create a stage:

1. In the **Assignment and Routing** section of the Human Task Editor, select a stage.
2. Drag the stage from the palette on the right side to a specific location on the canvass.

Figure 1-17 Create Stage

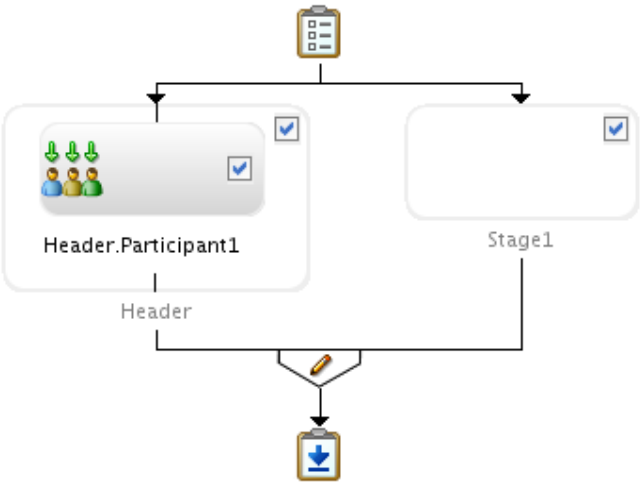
If you chose to create a sequential stage, the **Assignment and Routing** section looks like the following figure.

Figure 1-18 Add Sequential Stage



If you chose to create a parallel stage, the **Assignment and Routing** section looks like the following figure.

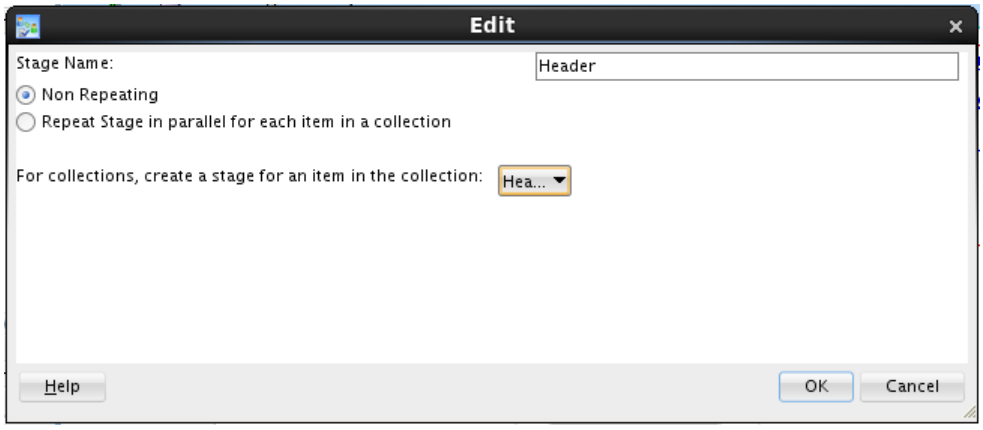
Figure 1-19 Add Parallel Stage



3. Double-click the stage you just created.

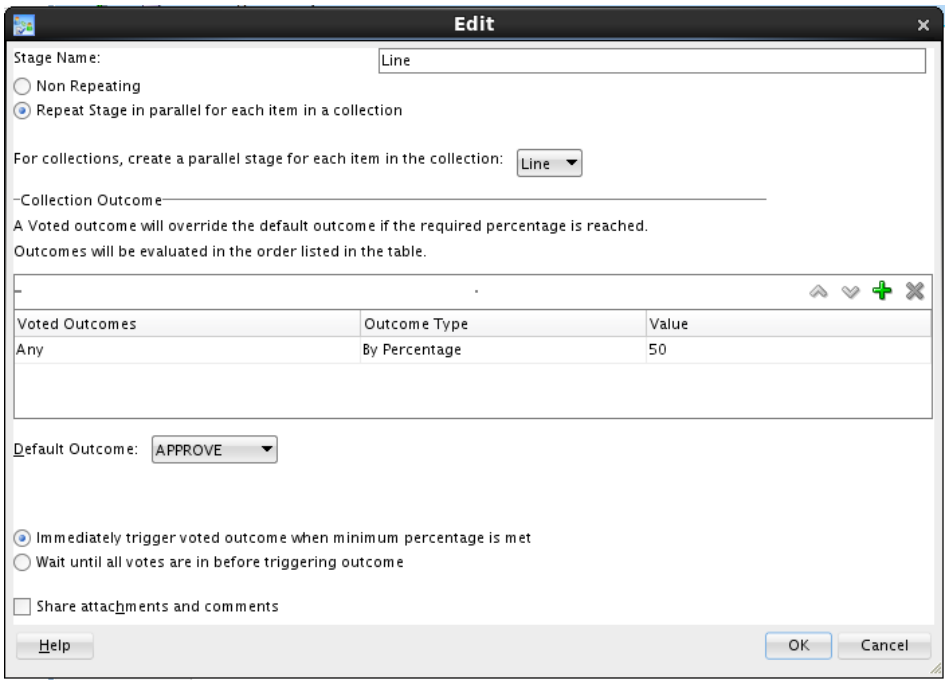
   The Edit dialog box displays, as shown in the following figure.

Figure 1-20 Edit Stage Dialog



4. Enter a name for the stage.
5. Choose one of these options:
   - **Non Repeating** - specifies that there is only one stage in parallel for each element in a collection
   - **Repeat Stage in parallel for each item in a collection** - specifies that the stage to repeat in parallel for each element in a collection. For example, if a purchase order contain 10 lines, the stage is repeated 10 times in parallel.
6. From the drop-down list, select a collection.
7. According to your selection, use one of these options:
   - If you selected **Non Repeating**, click **OK** to close the Edit dialog box.
   - If you selected **Repeat Stage in parallel for each item in a collection**, additional options display, as shown in the following figure.

Figure 1-21 Edit Stage Dialog: Repeat Stage



Do the following:

– Select a default outcome.

– Select a consensus percentage.

– Choose either to trigger the outcome immediately or wait until all the votes are in before triggering the outcome.

– Check the **Share attachments and comments** check box.

– Click **OK** to close the Edit dialog box.

How to Model Task Participants

Inside each stage you either can edit the default task participant or add new task participants. Task participants are assigned based on routing patterns, which can be any of the following:

- Single
- Parallel
- Serial
- FYI

After selecting a routing pattern, you also must select and model a list builder. This process is discussed in more detail in How to Model and Configure List Builders.

How to Model and Configure List Builders

Stages use a combination of list builders to generate the approver list. For more information, see Stages and List Builders. You can use each type of list builder only one time per stage. You can arrange these approver list builders in either sequential or parallel order. The order

you select governs the order in which those approvers included in approver lists that are generated by list builders are assigned an approval task.

The following list builders are specific to Approval Management extensions (AMX):

- Approval Groups (see How to Model an Approval Groups List Builder)
- Job Level (see How to Model a Job Level List Builder)
- Position (see How to Model a Position List Builder)
- Supervisory (see How to Model a Supervisory List Builder)

In the List Builder dialog, you can select to specify attributes in two ways: value-based (using the List Builder dialog) or rule-based (using the Rule Editor):

- **Value-based**: Specifies constraints to build the list of participants based on provided values in the List Builder dialog. Does not apply to a Position list builder.
- **Rule-based**: Specifies constraints to build the list of participants based on rules that are defined in the Rule Editor. Applies to all list builders.

Table 1-12 List Builder Options

Option Name	Description	List Builder
Name	The name of the approval group to use.	Approval Groups
Allow Empty Groups	When selected, allows the use of approval groups with no members. • Not selected: When an approval group has no members or is empty, the rules engine generates an error notification that the approval group is empty. • Selected: When an approval group has no members or is empty, the rules engine does not generate an error and continues to evaluate other rules and participants.	Approval Groups
Starting Participant	The first participant in a list, usually a manager.	Job Level Position (rule-based only) Supervisory
Top Participant	The last participant in the approval. Approval does not go beyond this participant in a hierarchy.	Job Level Position (rule-based only) Supervisory

Table 1-12 (Cont.) List Builder Options

Option Name	Description	List Builder
Number of Levels	A positive number specifying the lowest and highest job level (for Job Level), or the number of levels to traverse (for Supervisory). This number can be an absolute value, or a value relative to the starting point or creator. *Settings for Job Level:* • **At least**: Referred to as x1 here. – This assigns approvers as long as the job level '<' x1. As soon as x1 is '=' or '>' approver-job-level, it will stop assigning approvers. It checks the job level of the current user and then assigns if the condition matches. – The **At least** action is more stringent than the **At most** action. Therefore, the **At least** condition must be fulfilled first and then **At most** will continue from where **At least** ended. • **At most:** Referred to as x2 here. – This assigns approvers as long as the approver's manager's job level is '<' or '=' x2. It will not assign any approvers '>' x2. It checks the job level of the user to be assigned and if the condition matches the request, it goes to approver's manager. See Example Job Level Settings for Number of Levels Option following this table. *Settings for Supervisory:* In the context of the Supervisory list builder, the Number of Levels parameter is a way to limit the hierarchy traversal. The other parameter that governs the hierarchy traversal is Top Participant. If either of the conditions set by Number of Levels or Top Participant is reached or met, the hierarchy traversal is stopped. • **XPath**: An expression that evaluates to a positive integer. For example, `Task.payload.noOfLevels`. • **By Number**: A positive number specifying the number of levels to traverse for Supervisory.	Job Level Position (rule-based only) Supervisory
Relative to	A positive number specifying the number of levels to traverse for Supervisory, or the number of job level for Job Level and Position. Possible values are: starting point, creator and absolute.	Job Level Position (rule-based only)
Include all managers at last level	If the job level equals that of the previously calculated last participant in the list then it includes the next manager in the list.	Job Level
Utilized Participants	Utilizes only the participants specified in this option from the calculated list of participants. Available options are: Everyone, First and Last manager, Last manager.	Job Level Position (rule-based only)
Auto Action Enabled	Specifies if the list builder automatically acts on task based on the next option.	Job Level Supervisory (rule-based only) Position (rule-based only)

Table 1-12 (Cont.) List Builder Options

Option Name	Description	List Builder
Auto Action	Specifies the outcome to be set. It can be null if auto action is not enabled.	Job Level Supervisory (rule-based only) Position (rule-based only)

Example Job Level Settings for Number of Levels Option

Example 1: **At least** < **At most**

Settings:

- **At least** = 3 (absolute)
- **At most** = 5 (absolute)
- **Creator** = JL1 (level = 1)
- **Starting Participant** = manager
- **Include all managers at last level** = no
- **Top Participant** = JL9 (level=9)

Results:

- Starting point is always considered in the approval flow: JL2
- Evaluation of **At least** begins first and per **At least** = 3 condition: JL2 and JL3 are eligible, but JL2 has already been evaluated as part of the starting point condition, therefore the only **At least** condition match is JL3.
- **At most** evaluation begins and per **At most** = 5 condition: JL2, JL3, JL4, and JL5 are eligible, but JL2 and JL3 are already evaluated as part of the starting point and **At least** condition, therefore the only **At most** condition match are JL4 and JL5.
- **All Approvers:** Starting point (JL2) + **At least** (JL3) + **At most** (JL4, JL5) = JL2, JL3, JL4, JL5

Example 2: **At least** = **At most**

Settings:

- **At least** = 4 (absolute)
- **At most** = 4 (absolute)
- **Creator** = JL1 (level = 1)
- **Starting Participant** = manager
- **Include all managers at last level** = no
- **Top Participant** = JL9 (level=9)

Results:

- Starting point is always considered in the approval flow: JL2
- Evaluation of **At least** begins first and per **At least** = 4 condition: JL2, JL3, and JL4 are eligible, but JL2 has already been evaluated as part of the starting point condition, therefore the only **At least** condition match is JL3 and JL4.

- **At most** evaluation begins and per **At most** = 4 condition: JL2, JL3, and JL4 are eligible, but JL2, JL3, and JL4 are already evaluated as part of the starting point and **At least** condition, therefore there is no match for the **At most** condition.
- **All Approvers:** Starting point (JL2) + **At least** (JL3, JL4) + **At most** (no match) = JL2, JL3, JL4

Configuring the Hierarchy Provider Plug-In

If you do not configure the hierarchy provider plug-in, then the Position list builder does not work.

When you define a hierarchy extension, if you do not define the property `mustUseSpecifiedProvider`, then its default value is `true`.

You can configure the Supervisory and Job Level list builders to not throw an exception when there is a problem with the hierarchy plug in. To configure the list builders, you must add the `mustUseSpecifiedProvider` property to the `workflow-identity-config.xml` configuration file, and set the value attribute to `false`.

By default, the `workflow-identity-config.xml` file does not include the `mustUseSpecifiedProvider` property. If this property is present and its value is `false`, then the Supervisory and Job Level list builders use the LDAP management chain when there is a problem with the hierarchy plugin.

The following example shows a `workflow-identity-config.xml` file that specifies the `mustUseSpecifiedProvider` property. The value of this property is set to `true` so that the Supervisory and Job Level builders fail when the hierarchy plug in is not available.

```
<ISConfiguration xmlns="http://www.oracle.com/pcbpel/identityservice/isconfig">
  <configurations>
    <configuration realmName="jazn.com">
      <provider providerType="JPS" name="JpsProvider" service="Identity">
        <property name="jpsContextName" value="default"/>
        <property name="IdentityServiceExtension"
                  value="HCMIdentityServiceExtension"/>
      </provider>
    </configuration>
  </configurations>
  <property name="caseSensitive" value="false"/>
  <property name="mustUseSpecifiedProvider" value="true"/> <!-- Fail when the
hierarchy plug ins are not available-->
  <serviceExtensions>
...
</ISConfiguration>
```

How to Model an Approval Groups List Builder

Approval groups are a statically defined or a dynamically generated list of approvers. Approval groups usually are configured by the process owner using the worklist application. Typically, they are used to model subject matter experts outside the transaction's managerial chain of authority, such as human resources or legal counsel, that must act on a task before or after management approval.

Static approval groups are predetermined lists of approvers, while dynamic approval groups generate approver lists at runtime. Dynamic approval groups require:

- Delivery of an implementation according to the dynamic approver list interface by the developer

- Registration of the above implementation as a dynamic approval group using the Oracle BPM Worklist's UI by the IT department
- Availability of the class file in a globally well-known directory that is part of the SOA class path

Use dynamic approval groups when you need to calculate the approval group dynamically based on the task payload. Specially in line level approval where each line may require different approval group. For example, each cost center may require the approval of a different cost center owner. Each line may have different cost centers that require the approval of different cost center owners. When the number of cost centers is greater than one hundred, this may become difficult to manage with business rules.

Two views of the Approval Groups list builder are shown in the following figures.

Figure 1-22 Value-Based Approval Groups List Builder Dialog

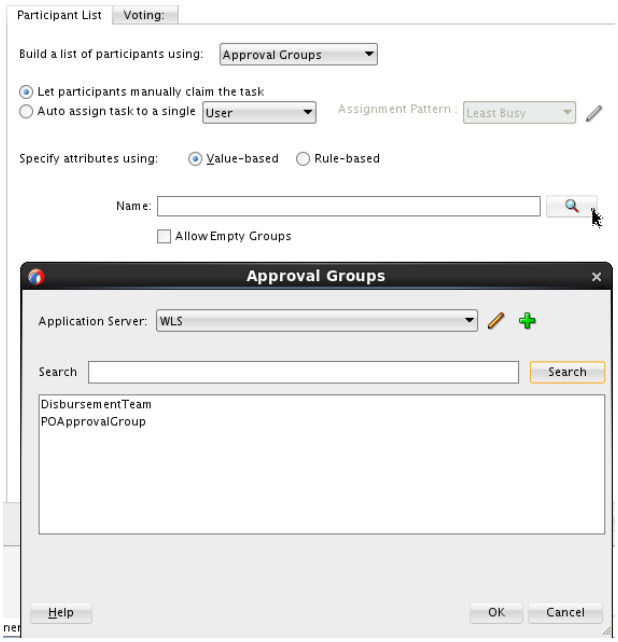
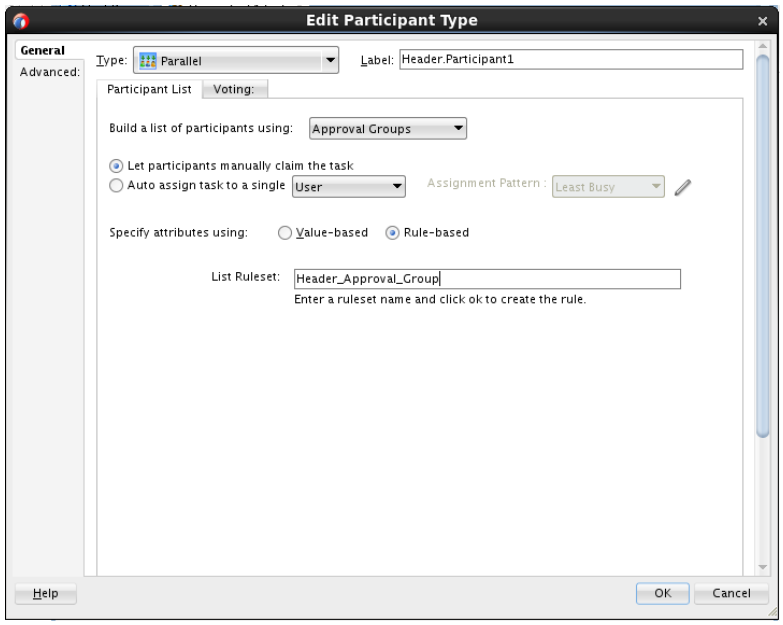


Figure 1-23 Rule-Based Approval Groups List Builder Dialog



To model an Approval Groups list builder, first specify if the list builder's attributes are to be value-based or rule-based, and then select the options on the corresponding dialog box. For information about the options, see Table 1-12.

> **Note:**
>
> If you configure the resource list with a group, then it behaves as a single type participant regardless of the serial or parallel type configuration.

How to Model a Job Level List Builder

The Job Level list builder ascends the supervisory hierarchy, starting at a given approver and continuing until an approver with a sufficient job level is found.

Two views of the Job Level list builder are shown in the following figures.

Figure 1-24 Value-Based Job Level List Builder Dialog

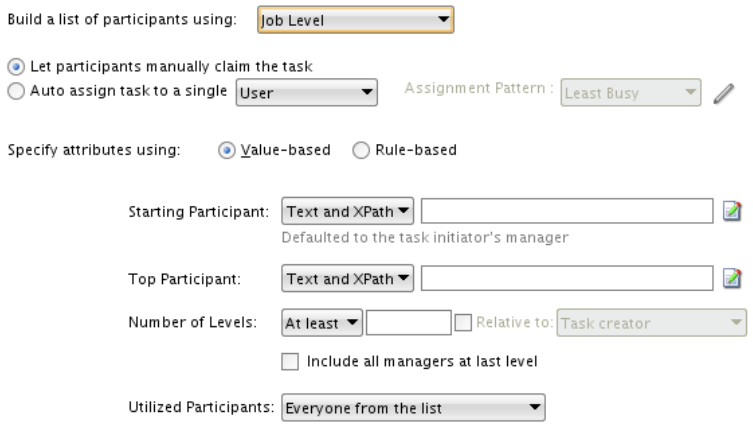
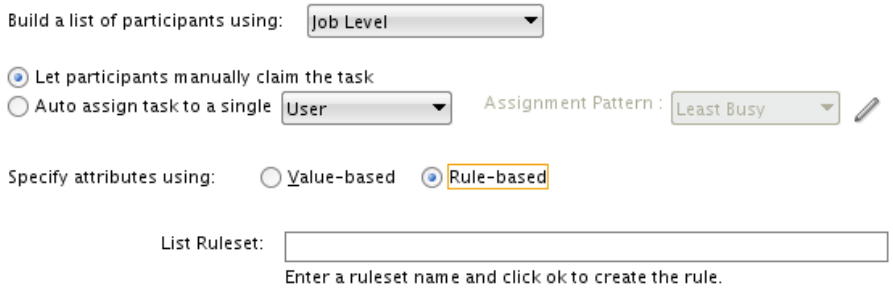


Figure 1-25 Rule-Based Job Level List Builder Dialog



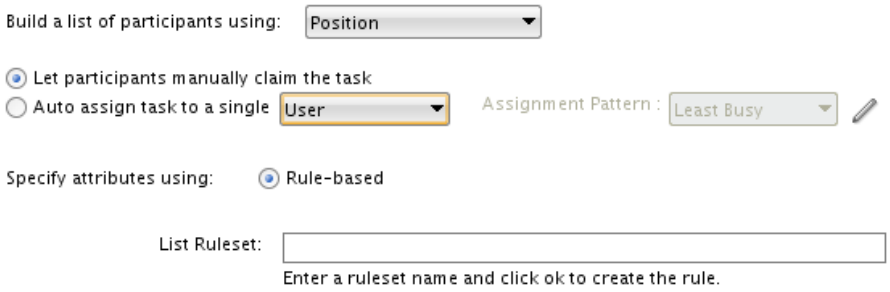
To model a Job Level list builder, first specify if the list builder's attributes are to be value-based or rule-based, and then select the options on the corresponding dialog box. For information about the options, see Table 1-12.

How to Model a Position List Builder

The Position list builder ascends the position hierarchy, starting at the requester's or at a given approver's position, and goes up a specified number of levels or to a specific position.

The following figure shows a view of the Position list builder.

Figure 1-26 Rule-Based Position List Builder Dialog



To model a Position list builder, first specify if the list builder's attributes are to be value-based or rule-based, and then select the options on the corresponding dialog box. For information about the options, see Table 1-12.

How to Model a Supervisory List Builder

The Supervisory list builder ascends the primary supervisory hierarchy, starting at the requester or at a given approver, and generates a chain that has a fixed number of approvers in it.

Two views of the Position list builder are shown in the following figures.

Figure 1-27 Value-Based Supervisory List Builder Dialog

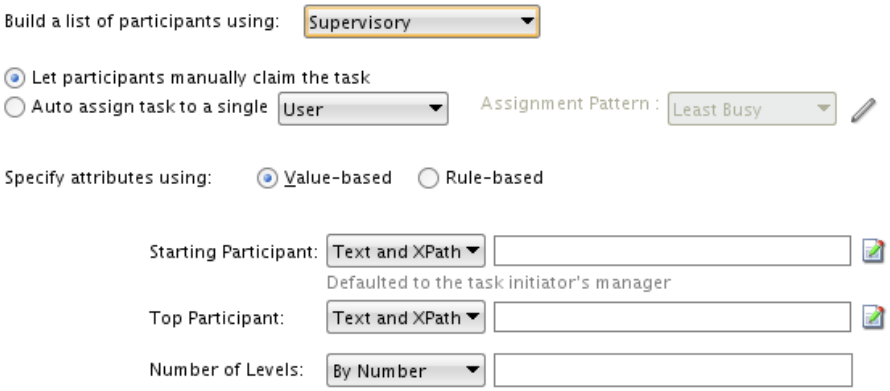
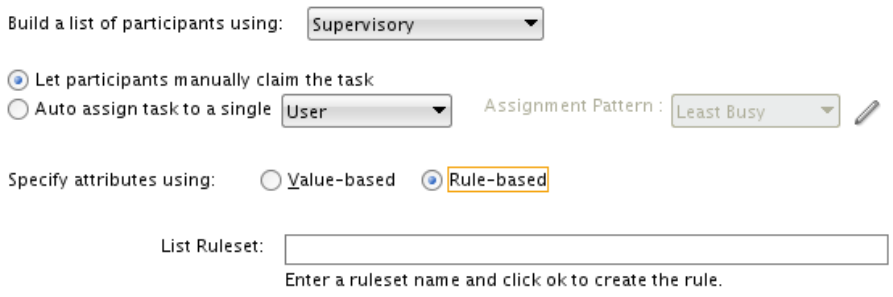


Figure 1-28 Rule-Based Supervisory List Builder Dialog



To model a Supervisory list builder, first specify if the list builder's attributes are to be value-based or rule-based, and then select the options on the corresponding dialog box. For information about the options, see Table 1-12.

How to Use Business Rules to Specify List Builders

Approvers of a task can be defined either inline in a task definition or by using business rules to specify the list builders that identify the actual approvers of a task. In addition, you can use business rules to specify approver substitution and list modifications. These rules are defined with the help of Oracle Business Rules and can vary between organizations. Typically, however, they are defined by the customer.

Business rules are a combination of conditions and actions. Optionally, priority and validity periods can be defined for these rules. In Human Workflow rules, rule conditions are defined using fact types that correspond to the task, and to the task message and entity attributes (which are XML representation of SDO objects). Rule actions consist of approver list builders and their parameters. Approver list builders move up a particular hierarchy and construct or modify the approver list according to the parameters defined. Approver list builders are implemented as XML (JAXB) fact types.

For more information about these concepts, see the chapters in *Using the Business Rules Service Component* in *Developing SOA Applications with Oracle SOA Suite*.

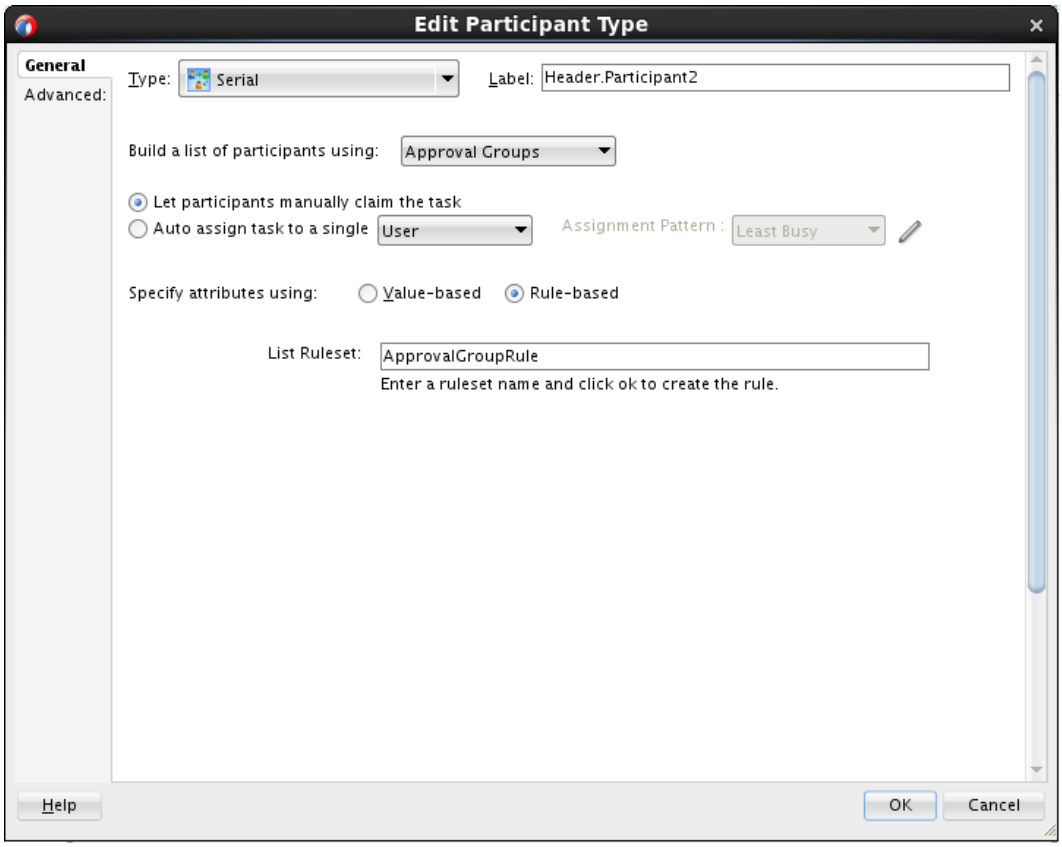
The sections that follow explain list creation, approver substitution, list modification, and repeating node attributes using Oracle Business Rules.

How to Create Lists

You can use business rules to define the list builders you want to use. There are two types of business rules:

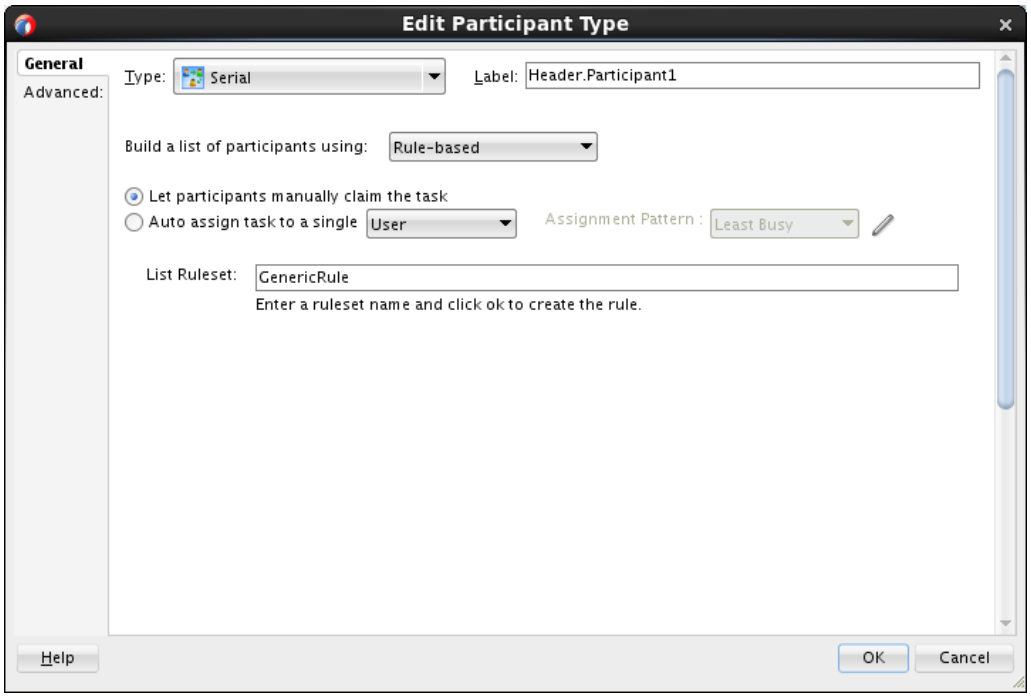
- Rules that define the parameters of a specific list builder. In this case, the task routing pattern dialog box is modeled to use a specific list builder. The parameters in the list builder come from rules. With this option, rules should return a list builder of the same type as the one modeled in JDeveloper. The following figure shows a sample configuration.

Figure 1-29 Specific List Builder Configuration



- Rules that define the list builder and the list builder parameters. In this case, the list itself is built using rules. The following figure shows a sample configuration.

Figure 1-30 List Builder and Parameters Configuration

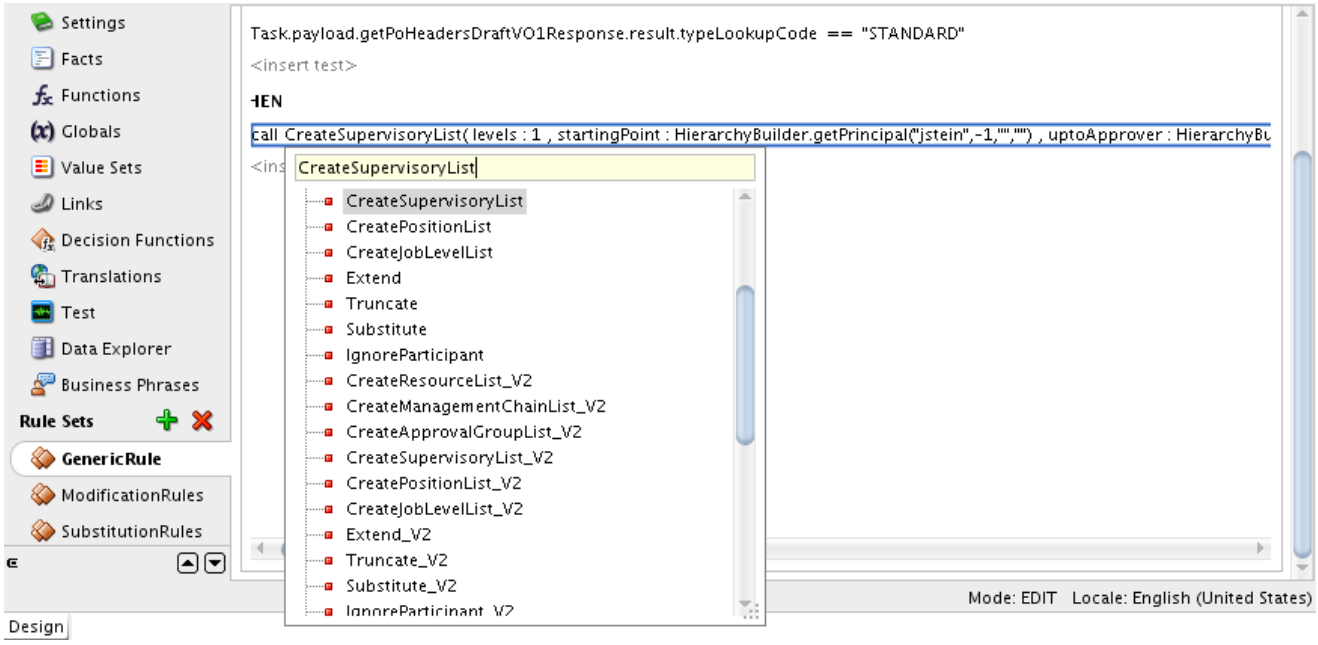


In the rule dictionary, rule functions are seeded to facilitate the creation of list builders. The list builder functions are:

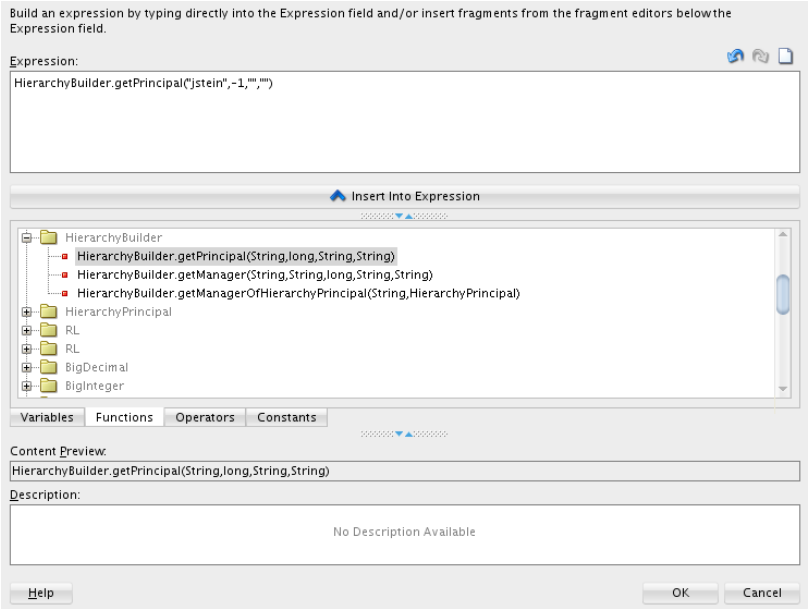
- `CreateResourceList`
- `CreateSupervisoryList`
- `CreateManagementChainList`
- `CreateApprovalGroupList`
- `CreateJobLevelList`
- `CreatePositionList`

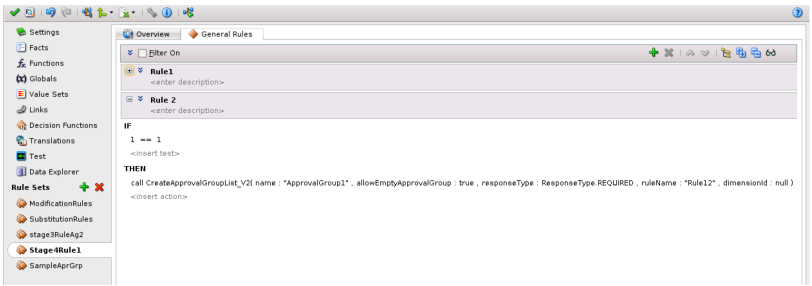
In Rules Designer, model your conditions and, in the action part, call one of the functions above to complete building your lists, as shown in the following figure.

Figure 1-31 Modeling Conditions in Rules Designer



The parameters for the rule functions are similar to those in JDeveloper modeling. In addition to the configurations in JDeveloper, some additional options are available in Rules Designer:

Parameter	Description
`startingPoint` `topApprover`	In JDeveloper, starting point and top approver are specified as users. In Rules Designer, you can build a hierarchy principal as the starting point and top approver using the `HierarchyBuilder` function, as shown in the following figure. **Note**: If you want to leave the job level attribute undefined when using the `HierarchyBuilder` function, then you must set its value to a negative integer.  `HierarchyBuilder` has a number of functions including `getManager`, `getPrincipal`, and `getManagerOfHierarchyPrincipal`. • `HierarchyBuilder.getManager` builds an approval list using the following parameters: – `ListbuilderType` (string). Valid values: `"supervisory"`, `"joblevel"`, `"position"` – `ReferenceUser` (string). For example, `Task.creator` – `AssignmentID` (long). The default value is `-1`, otherwise it is set to the user. – `EffectiveDate` (string). For example, `"2021-06-15"` – `HierarchyType` (string). The type of manager to look for when the list is built. Example values are: `"LINE_MANAGER"`, `"RESOURCE_MANAGER"`, `"CORPORATE_MANAGER"`, `"PROJECT_MANAGER"` Example: `HierarchyBuilder.getManager("supervisory",Task.creator,-1,"2021-06-15","LINE_MANAGER")` • `HierarchyBuilder.getPrincipal` locates an approval list member and can be used, for example, to identify the top approver in an approval list. It takes the following parameters: – `PrincipalName` (string). Valid values: `"supervisory"`, `"joblevel"`, `"position"` – `AssignmentID` (long). The default value is `-1`, otherwise it is set to the user. – `EffectiveDate` (string). For example, `"2021-06-15"`

Parameter	Description
	– `HierarchyType` (string). The type of manager to look for when the list is built. Default: `"LINE_MANAGER"`. Other possible values are: `"RESOURCE_MANAGER"`, `"CORPORATE_MANAGER"`, `"PROJECT_MANAGER"`
`allowEmptyApprovalGroup`	In Rules Designer, you can specify whether or not to allow the use of approval groups with no members using the `CreateApprovalGroupList` function, as shown in the following figure. Valid values: • `false`: When an approval group has no members or is empty, the rules engine generates an error notification that the approval group is empty. • `true`: When an approval group has no members or is empty, the rules engine does not generate an error and continues to evaluate other rules and participants. 
`autoActionEnabled` `autoAction`	In Rules Designer, you can configure that the users resulting from a particular list builder can act automatically on the task. Valid values for `autoAction`: null | `Approve` | `Reject`
`responseType`	If the response type is `REQUIRED`, the assignee has to act on the task; otherwise, the assignment is converted to an `FYI` assignment. Valid values: `REQUIRED` | `NOT_REQUIRED`
`ruleName`	Rule name is used to create an assignment reason. The *`rule_set_name_rule_name`* is used as a key to look up the resource bundle for a translatable reason for assignment. This resource is looked up first in the project resource bundle, then in the custom resource bundle, and last in the system resource bundle. Valid values: any valid string
`lists`	This is an object that is a holder for all the lists that are built. Clicking this option shows a pre-asserted fact that a `Lists` object is used as the parameter.

The following figures show examples of rules.

Figure 1-32 Example Rules (1)

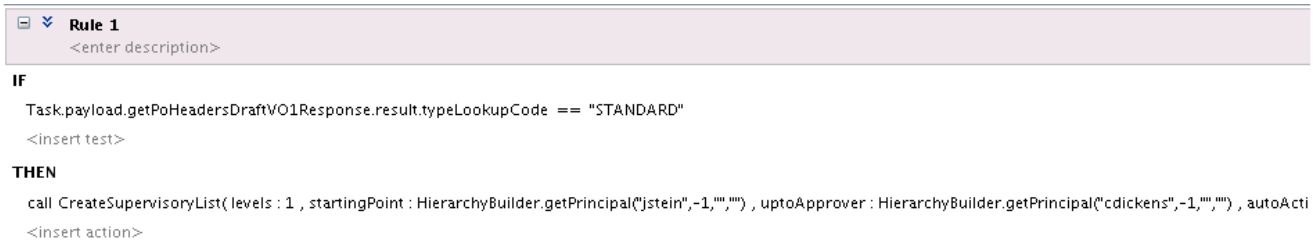


Figure 1-33 Example Rules (2)

```
archyBuilder.getPrincipal("cdickens",-1,"","") , autoActionEnabled : true , autoAction : "APPROVE" , responseType : ResponseType.REQUIRED , ruleName : "Rule_1" , lists : Lists )
```

> **Note:**
>
> If multiple rules fire, the list builder created by the rule with the highest priority is selected.
>
> If the rules have the same priority, they are fired in random order, the first one fired is selected.

> **WARNING:**
>
> An improper or incomplete rules definition in a list-creation rule set can cause runtime errors. Errors can be caused by the following:
>
> - No rule was defined in the rule set.
> - None of the conditions defined in the rule was met.
>
> Ensure that rules are properly defined to handle all conditions.

How to Make Approver Substitutions

List substitution enables you to substitute users, groups, and application roles that appear in a list. List substitution is available from Rules Designer and does not require any configuration in JDeveloper. In each rule dictionary there is a pre-seeded rule set named "SubstitutionRules." Also in the rule dictionary, a "Substitute" rule function is seeded to configure list substitutions. Table 1-13 lists the "Substitute" functions and their parameters.

Table 1-13 "Substitute" Function Parameters

Parameter	Description
fromId	The ID of the user/group/application role from which to substitute.
toId	The ID of the user/group/application role which to substitute to.
ruleName	Used to create an assignment reason. Rule set name + "_" + rule name is used as a key to look up the resource bundle for a translatable reason for assignment. This resource is looked up first in the project resource bundle, then in the custom resource bundle, and last in the system resource bundle.

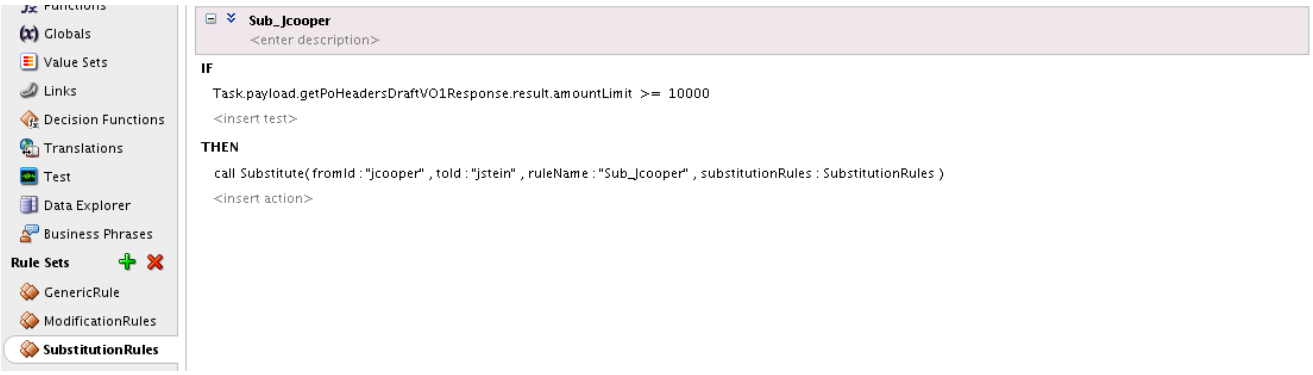
Table 1-13 (Cont.) "Substitute" Function Parameters

Parameter	Description
substitutionRules	An object that is a holder for all the substitutions. Clicking this option shows a pre-asserted fact 'SubstitutionRules' object to be used as the parameter.

> **Note:**
>
> In a Human Task with a substitution rule, the resulting approval list might have a duplicate participant. It is not possible to edit the duplicate approvers in the Future Participants list.

The following figure shows a sample approver-substitution action.

Figure 1-34 Sample Approver-Substitution Action



How to Make List Modifications

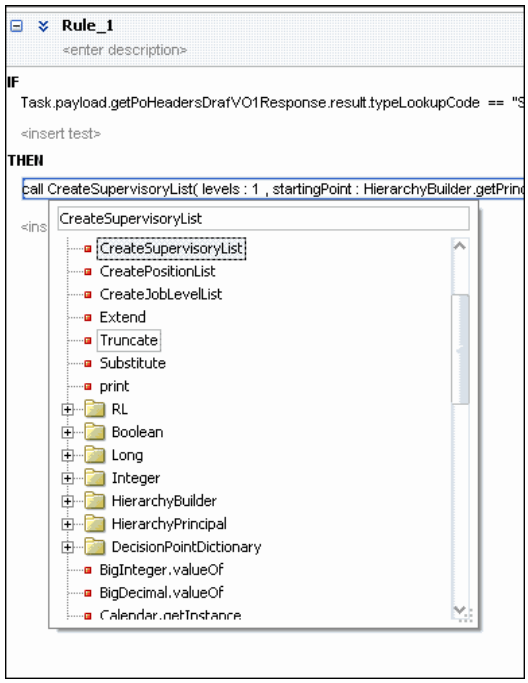
List modification enables you to extend or truncate the Job Level and Position list builders from rules. List modification is applied after the list is created. This feature does not require any configuration from JDeveloper. In each rule dictionary there is a pre-seeded rule set named "ModificationRules." This rule set is called only when the Job Level and Position list builders are asserted in the list that created the rule sets. Only the highest priority applicable rule is applied.

In Rules Designer, rule functions are seeded to facilitate list modifications. These functions are the following:

- Extend
- Truncate

These rule functions are shown in the following figure.

Figure 1-35 Rule Functions



Extend and truncate parameters are listed in Table 1-14 and Table 1-15.

Table 1-14 "Extend" Function Parameters

Parameter	Description
ifFinalApproverLevel	The level at which final approver is at or below.
extendBy	The number of levels to add to the final job level.
ruleName	Used to create an assignment reason. Rule set name + "_" + rule name is used as a key to look up the resource bundle for a translatable reason for assignment. This resource is looked up first in the project resource bundle, then in the custom resource bundle, and last in the system resource bundle.
lists	An object that is a holder for all the lists that are built. Clicking this option shows a pre-asserted fact 'Lists' object to be used as the parameter.

Table 1-15 "Truncate" Function Parameters

Parameter	Description
afterLevel	The level after which to truncate.
ruleName	Used to create an assignment reason. Rule set name + "_" + rule name is used as a key to look up the resource bundle for a translatable reason for assignment. This resource is looked up first in the project resource bundle, then in the custom resource bundle, and last in the system resource bundle.

Table 1-15 (Cont.) "Truncate" Function Parameters

Parameter	Description
lists	An object that is a holder for all the lists that are built. Clicking this option shows a pre-asserted fact 'Lists' object to be used as the parameter.

The following figure shows a sample list-modification action.

Figure 1-36 Sample List-Modification Action

How to Define Repeating-Node Attributes of a Business Rule Condition

When defining a business rule, you can base a rule condition on an attribute that comes from a repeating node. For example, there can be multiple line items for each purchase-order header in a purchase-order scenario. In this case, PurchaseOrderHeader is a non-repeating node, and PurchaseOrderLines is a repeating node.

When defining a rule like the following:

IF line item's amount is <50000, THEN create supervisory list containing jcooper up to two levels

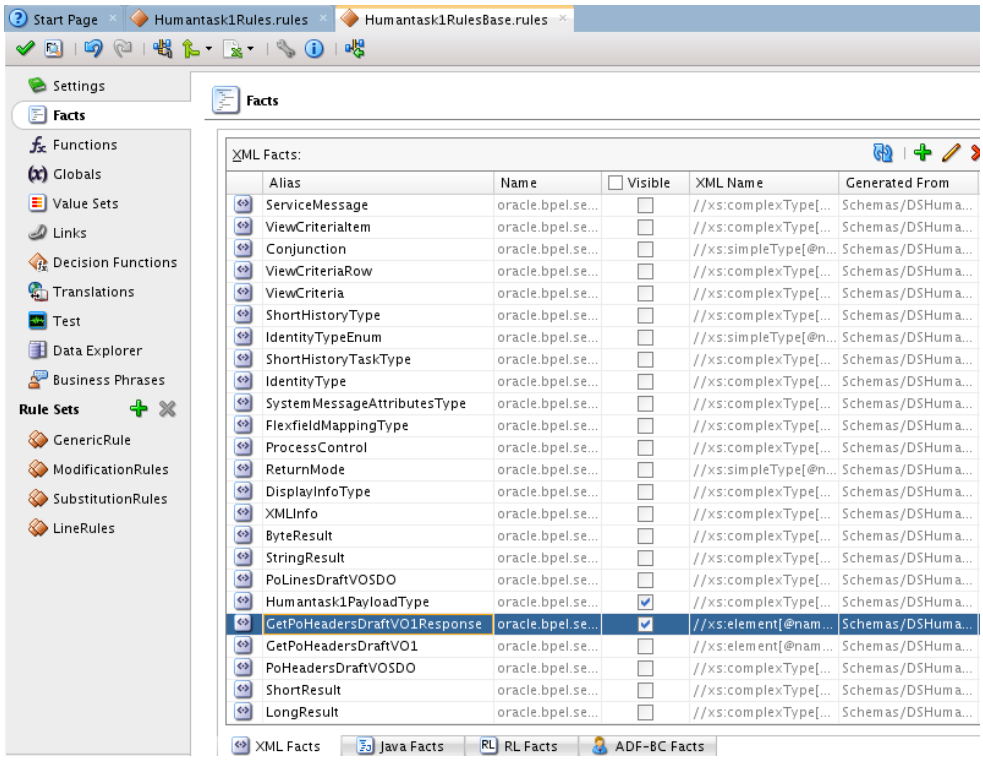
the amount is an attribute of *line*, that is, it is an attribute of a repeating node.

Use the following procedure to define repeating-node attributes:

1. In Base Dictionary, select **Facts**.

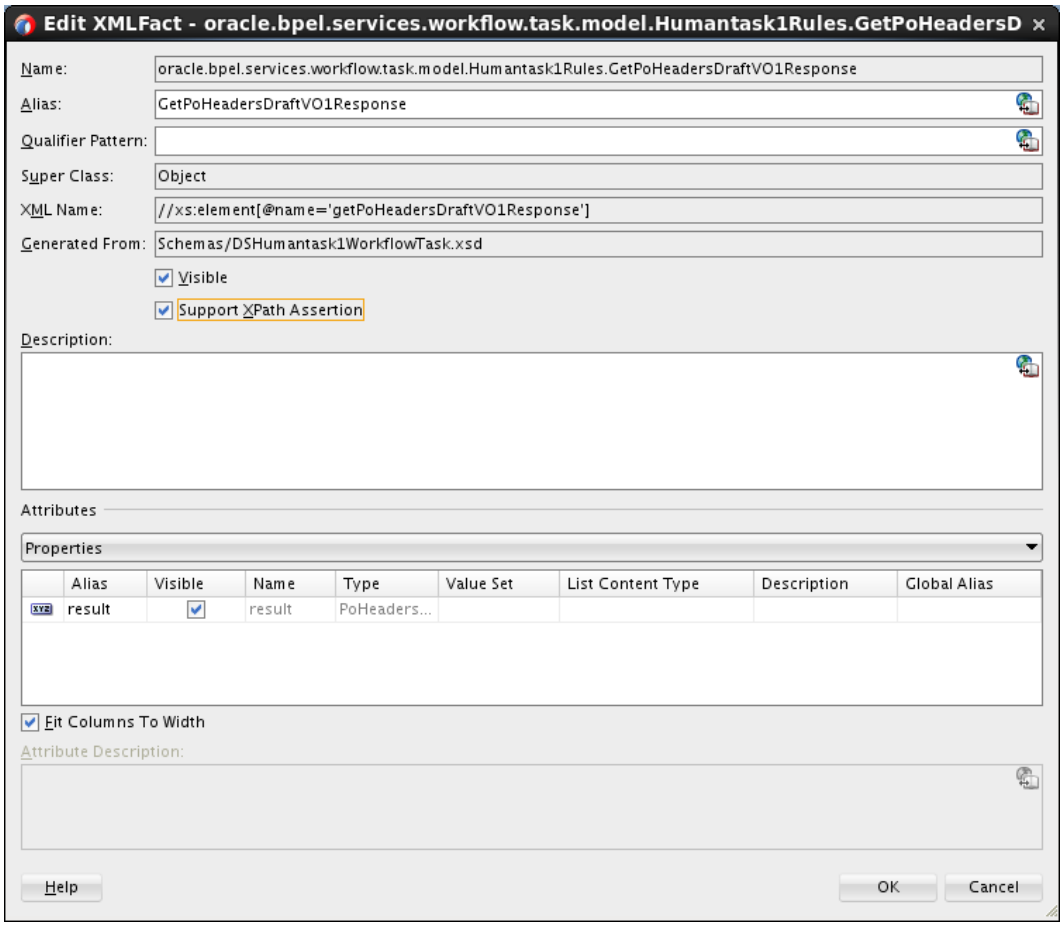
   In the Humantask1RulesBase rules tab, a list of facts displays as follows.

Figure 1-37 Facts List



2. Edit each appropriate fact to ensure that it is visible, as shown in the following figure.

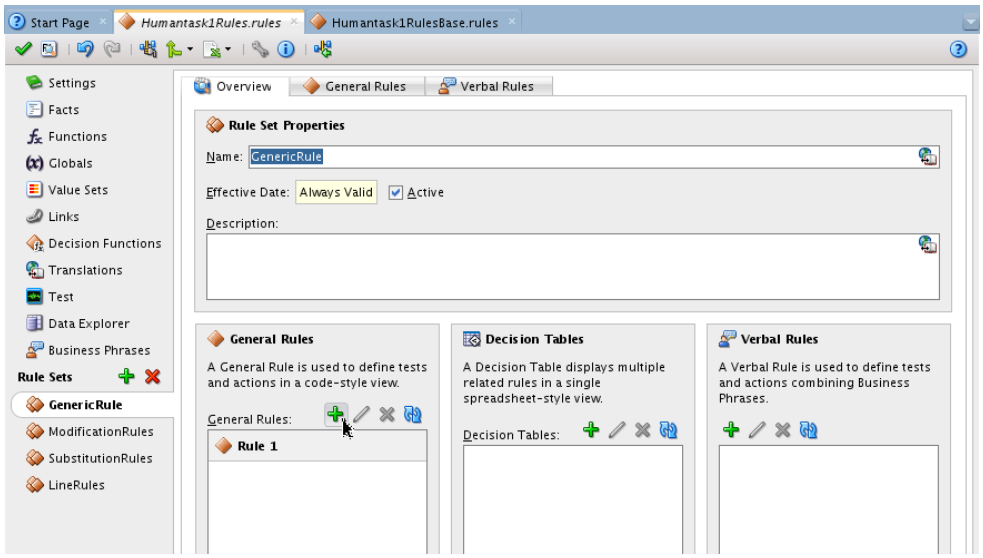
Figure 1-38 Edit XML Fact Dialog



3. Decide whether you want to add a generic rule, a decision table, or a verbal rule. Once you decide, click the Add (+) button. In Rules Designer, select a rule and click **Add** icon (+).

   The following rule-definition section displays.

Figure 1-39 Rule-Definition Section



4. Click the double down arrows to the left of the rule name to show advanced settings, as shown in the following figure.

Figure 1-40 Advanced Settings

5. Select **Tree Mode**, then click **<fact type>** to display a list of options from which to choose a ROOT, as shown in the following figure.

Figure 1-41 ROOT Options

6. Define the rule conditions.

How to Use Assignment Context

Assignment context is information that is present in the task. During a task's life cycle, it progresses through various assignees. As the context of the task assignees changes, the assignment-context value also changes.

When browsing through the history of a task, you can see the various assignment contexts that the task contained during its life cycle. The Oracle BPM Worklist uses assignment context when it displays task history.

Configuring Assignment Context

You configure assignment context in the Add (or Edit) Participant Type dialog box in JDeveloper in the following ways:

- Select the **Rule-based** option in the **Participant Type** section.

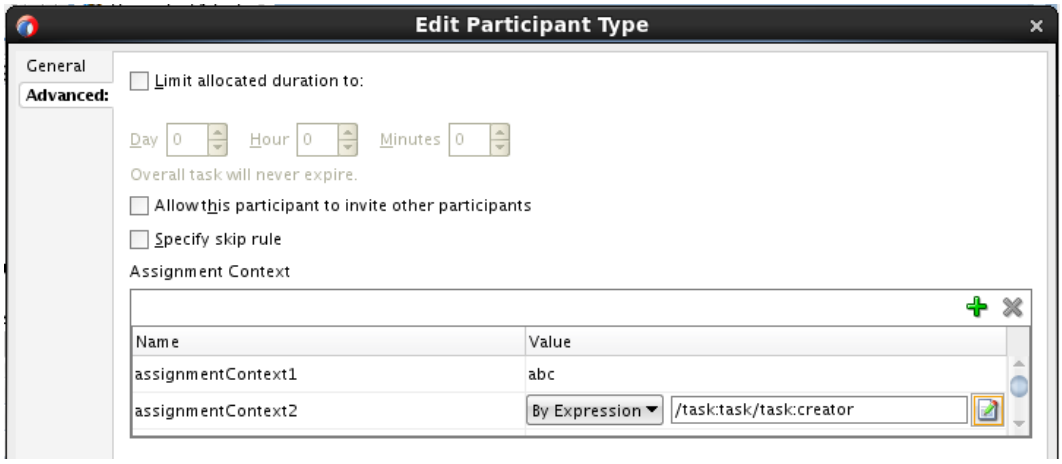
  In this case, the assignment context is configured implicitly, behind the scenes. The Rules layer resolves the list of assignees based on the rule. As the task progresses through the various assignees, the assignment context value is computed based on the rule.

  Assignment context can also be assigned in value-based context. For more information, see *Assigning Task Participants* in *Developing Business Processes with Oracle Business Process Management Studio*.

- Select the **Advanced** finger tab to configure any number of assignment contexts.

  In this case, you can customize assignment contexts by entering your own information into the Assignment Context fields. the following figure shows the fields.

Figure 1-42 Assignment Context Section



The following table contains field descriptions:

Table 1-16 Assignment-Context Field Descriptions

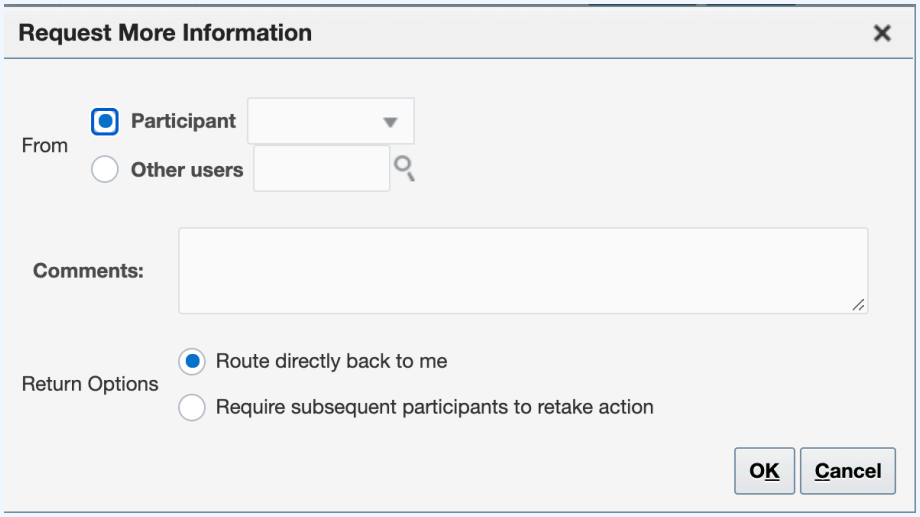
Field	Description
Name	Assignment-context name, which can be whatever you choose. This is a string field.
Value	Assignment-context value, which can be whatever you choose. This is a string field.
Type	Associated with the Value field. Possible values are: – **By name** - A user-provided Value parameter. – **By Expression** - A Value parameter created by the Expression Builder.

How to Aggregate Task Approvals

A task can be assigned multiple times to one user during the task life cycle. The Human Task Editor enables you to configure how often a user sees the task.

Note:

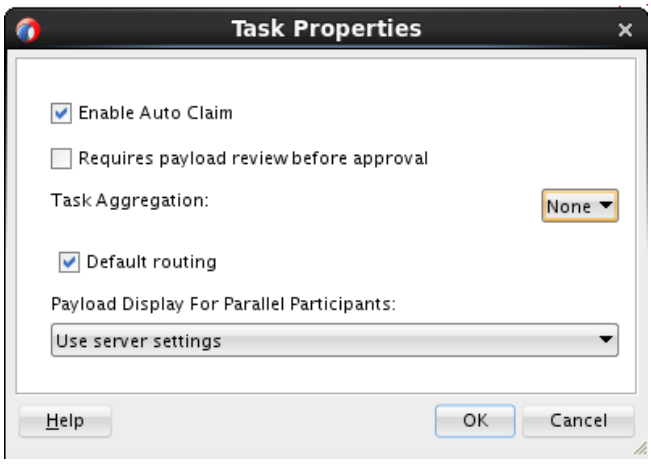
In Oracle BPM Worklist, the **Request Information** action allows a task approver to request information from any prior participant or the task initiator. In the case of an aggregated task, the Request Information return option of **Route directly back to me** behaves the same way as **Require subsequent participants to retake action**. When the information is submitted, the task is assigned to the requester.



The following procedure explains how to configure task-approval aggregation.

1. In JDeveloper, click **Configure** at the top.

   The Task Properties window is displayed.



2. Select a task-aggregation option from the drop-down list:
   - **None** - Indicates there is no approval aggregation, which means the user sees the task as many times as it is assigned to him or her.
   - **Stage** - A user sees the task only one time in a stage.
   - **Task** - A user sees the task only one time in the task life cycle.
3. Click **OK**.

When the task is aggregated and assigned to a user, the task has a collection table in the Oracle BPM Worklist that displays all the collections in the task the user is approving. After the user performs an action, the action is recorded and then replayed to all the user's assignments, either in the stage or task.

An aggregated task is a proxy task for all the regular assignments.

Aggregated tasks are business tasks and show the actions approve and reject. If you can aggregate FYI tasks, then they show the approve and reject actions. In this case the approve and reject actions are treated as an acknowledgement.

> **Note:**
>
> Aggregation is available only when the assignees are from the same set. For example, if you assign a task to user A and another to both user A and user B; then user A sees two separate tasks. The two assignments are not aggregated because the assignees are not exactly the same.

Defining Escalation and Renewal Policies

This feature is not specific to AMX. For more information, see *Escalating, Renewing, or Ending the Task* in *Developing SOA Applications with Oracle SOA Suite*.

> **Note:**
>
> Escalation is only applicable to management chain.

Specifying Notification Settings

This feature is not specific to AMX. For more information, see *Specifying Participant Notification Preferences* in *Developing SOA Applications with Oracle SOA Suite*.

Using Advanced Settings

Using advanced settings includes the following tasks:

How to Add Callbacks for Notes, Attachments, and Validation

Callbacks are mechanisms that allow you to do the following:

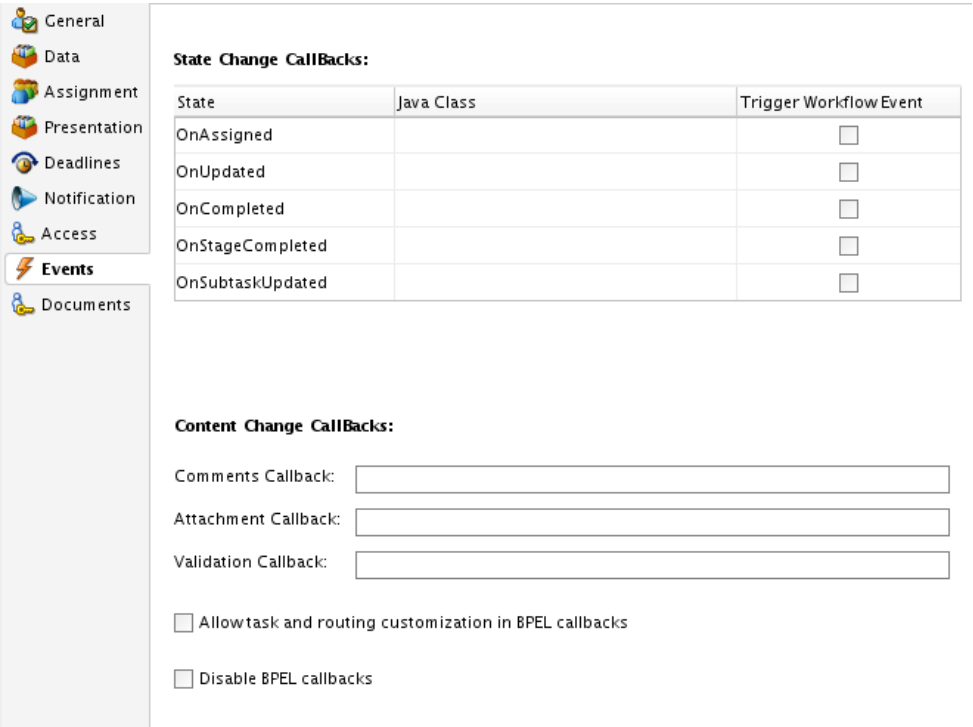
- Access notes and attachments associated with business objects from external content-management systems or custom schemas
- Perform custom validation of workflow tasks at various points in a task life cycle by defining validation logic for each task action

Use the following procedure to add callbacks:

1. From the Task Editor, select the 4 finger tab to configure the callbacks.

   The Callback Details dialog box opens as follows.

Figure 1-43 Callback Details Dialog



2. Use one of these options:
   - In the Comments Callback field, enter the appropriate Java class for the notes callback.

- In the Attachments Callback field, enter the appropriate Java class for the attachments callback.
- In the Validation Callback field, enter the appropriate Java classes, separated by commas, for the validation callback.

3. Click **OK**.

How to Define Security Access Rules

Access rules restrict the actions that a user can perform by overriding default actions and permissions. At runtime, the system checks every operation in a task against any defined access rules to see if a user is permitted to make changes, such as approve, add, delete, and so on If the user is not permitted to make changes, the operation errors out with an appropriate error message.

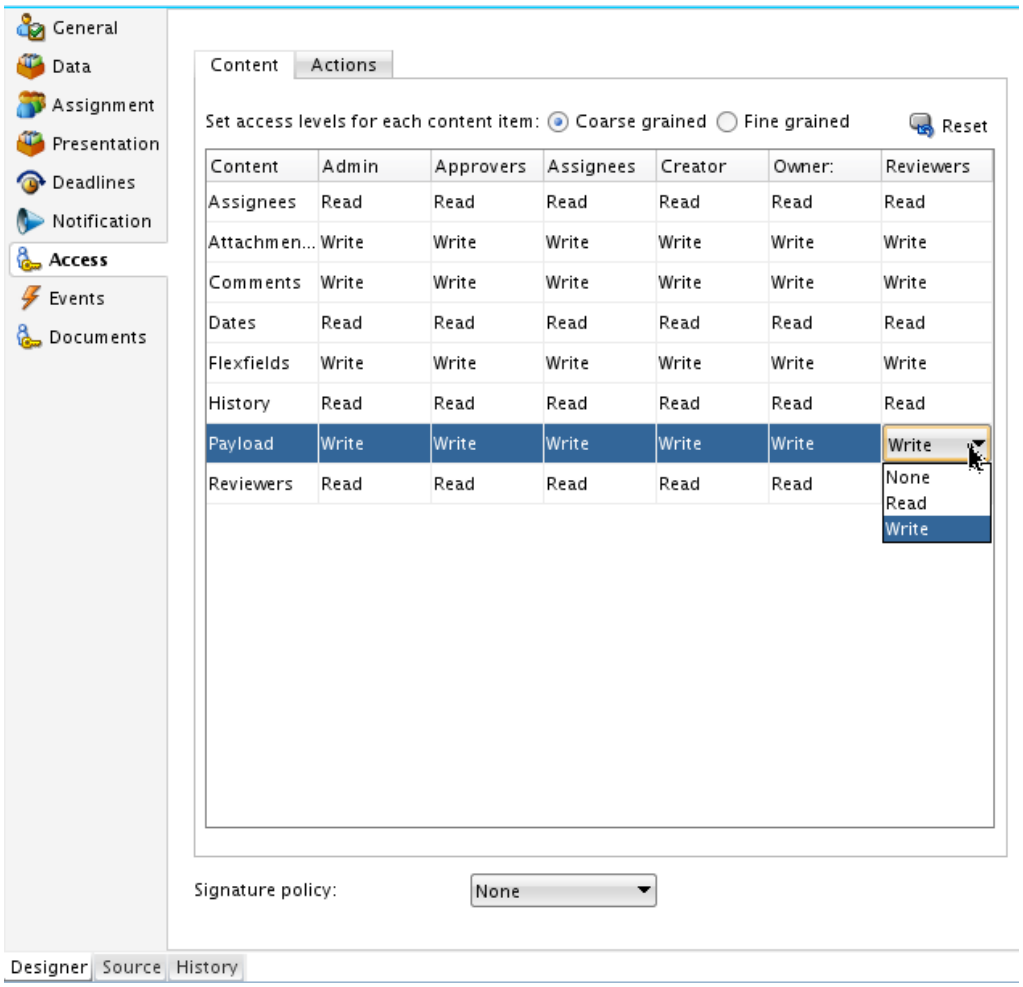
In AMX, access rules can be defined for Groups and Application Roles. For example, if an access rule is defined to restrict the "Withdraw" action for a group called Operators, then any user belonging to that group is not allowed to withdraw the task. Similarly, if an access rule is defined to restrict the "Withdraw" action for an application role called SOAAuditViewer, then any user who has been granted the SOAAuditViewer application role is not allowed to withdraw the task.

To define a security access rule:

1. Select the **Access** finger tab to display security access rules.
2. Click **Configure Visibility**.

   The Configure Task Content Access dialog box displays, as shown in the following figure.

Figure 1-44 Configure Task Content Access Dialog (1)



3. Click the **Task Content** or **Task Actions** tab. (This procedure assumes the **Task Content** tab has been selected.)
4. Look up the appropriate content and role in the grid.
5. From the drop-down list, select the appropriate privilege or action.
6. Click **OK** to close the dialog box.

Use the same procedure to define access rules for Application Groups, with the following exceptions:

- Click the **Task Actions** tab to select it.
- Select **Application** from the drop-down list.
- Select application roles to include in the access rule from the Select an Application Role dialog box, as shown in the following figure.

For more information, see *Specifying Access Policies and Task Actions on Task Content* in *Developing SOA Applications with Oracle SOA Suite*.

Using the End-to-End Approval Management Samples

You can use samples of end-to-end approval management.

Table 1-17 shows the end-to-end workflow examples included in the `ORACLE_HOME\samples\soa-infra\workflow\amx` directory.

In addition to the demonstration features listed in the table, all samples show the use of worklist applications and workflow notifications.

Table 1-17 End-to-End Samples

Sample	Description	Location
Expense Line Approval	Illustrates line-level approval with approval policy defined.	`ORACLE_HOME\samples\soa-infra\workflow\amx\amx-101-expense-line`
Employee Hiring	Illustrates ad-hoc insertion capabilities for an approval having two stages - Approval Group List Builder in "Order" voting regime and a Supervisory list builder.	`ORACLE_HOME\samples\soa-infra\workflow\amx\amx-102-hiring-approval-group`
Purchase Order Approval	Illustrates the Purchase Order approval scenario with header and line-level approvals.	`ORACLE_HOME\samples\soa-infra\workflow\amx\amx-103-purchaseOrder-2dimensions`
Employee Transfer	Illustrates the Employee Transfer scenario from one team to another through parallel job level participants.	`ORACLE_HOME\samples\soa-infra\workflow\amx\amx-104-employee-transfer`
Self Approval	Illustrates how to implement self-approval through auto-action rules.	`ORACLE_HOME\samples\soa-infra\workflow\amx\amx-105-self-approval`
Position List Builder	Illustrates the use of the Position list builder.	`ORACLE_HOME\samples\soa-infra\workflow\amx\amx-108-position-list`

Using the User Metadata Migration Utility

The user metadata migration utility, **hwfMigrator**, automates the process of migrating Workflow user-configurable data from one SOA server to another by executing a shell script.

For more information about the user metadata migration utility, see *Moving Human Workflow Data from a Test to a Production Environmen* in *Administering Oracle SOA Suite and Oracle Business Process Management Suite*.

GET_SEARCH_RESULTS

Service that returns a list of content items that match specific search criteria.

Access Level: Read (1)

Calls SubService: SUB

Location: *`IdcHomeDir`*`/resources/core/tables/std_services.htm`

Additional Required Service Parameters

- QueryText: The search expression.

You can append values for Title, Content ID, and so forth, on the QueryText parameter to refine this service.

Optional Service Parameters

- ResultCount: The number of results to return. Defaults to 25.
- SearchEngineName: The name of the search engine to be used. The default is the value specified in the `config/config.cfg` file.

  Values can be `databasefulltext` or `database`. If set to `database` or `databasefulltext`, you must pass SQL in the QueryText parameter, as in this example:

  ```
  dDocTitle like 'test'
  ```

  This is equivalent to the Verity query:

  ```
  dDocTitle <substring> 'test'
  ```

  - SortField: The name of the metadata field to sort on.
    * Examples: *dInDate*, *dDocTitle*, *Score*.
    * Defaults to dInDate.
  - SortOrder: The sort order. Allowed values are *ASC* (ascending) and *DESC* (descending).
  - SortSpec: Enables sorting on more than one field. Set this parameter to the following sequence:

    ```
    <sort field> <sort order> <sort field> <sort order>...
    ```

  For example, `SortSpec=dDocTitle ASC dInDate DESC`.

  - StartRow: The row to begin the search results display. For example, if *ResultCount=25*, setting *StartRow=26* displays the second page of results.
  - EndRow: The row to end the search results display.
- vcrContentType: The name of a searchable content type. The server modifies the query text of the search to limit the results to documents of that type. For example, if the content type specified is one describing a profile, then the query text is modified to limit the documents returned to those whose profile trigger value matches that of the profile.
- vcrAppendObjectClassInfo: When set to *true*, the server adds an additional column to the SearchResults ResultSet called vcrObjectClass. This column lists the content type associated with each document in the results. The default is *true*.

Example

```
IdcService=GET_SEARCH_RESULTS
QueryText=benefits
```

How to Use Advanced Mode Action Forms

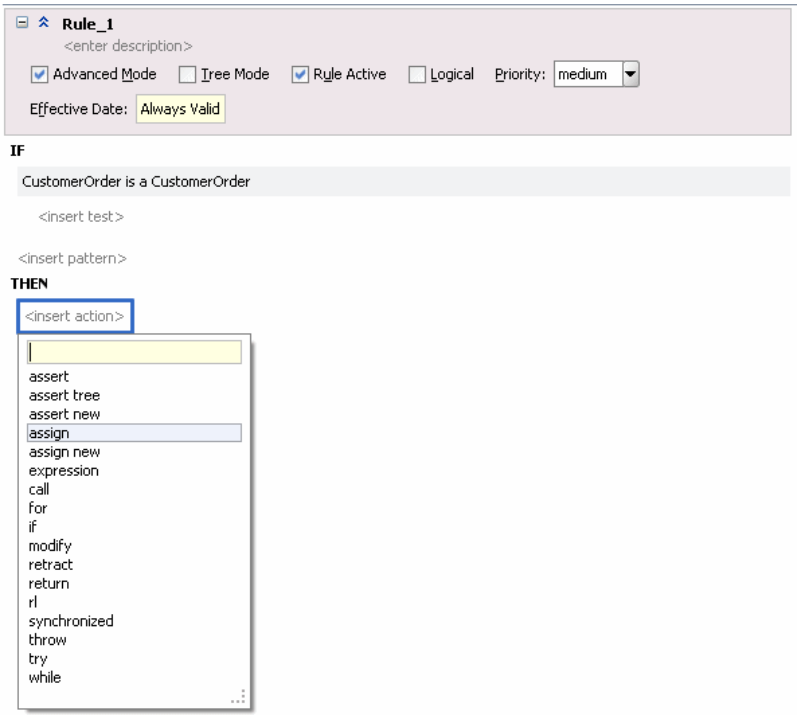
This section explains how to use Advanced Mode Action forms.

When you create a rule with Advanced Mode, Rules Designer presents a list with the available actions shown in Advanced Mode Action Options in Rule Designer. For each form shown in Advanced Mode Action Options, the options that Rules Designer presents are context sensitive. Thus, the lists and the number of items you see when you work with the action types are context sensitive, depending on which action you add and the choices you make while you enter the action.

To use advanced mode action forms:

1. In Rules Designer, select a ruleset from the **Rulesets** navigation tab.
2. Select or add a rule or a Decision Table.
3. In the rule or Decision Table click the **Show Advanced Settings** button next to the rule or Decision Table name.
4. Select **Advanced Mode**.
5. With the insertion areas showing, in a rule in the **THEN** area select **<insert action>**. This displays the action list.

Figure 1-45 Adding an Action to a Rule in Advanced Mode



6. In the list, select the action you want to add.

   For example, select **assign new**.

7. In the **THEN** area, select the context sensitive parameters for the action and enter appropriate values.

Advanced Mode Action Options in Rule Designer

Table 1-18 Advanced Mode Action Options

Action Form	Description
`Assert`	Assert a fact
`Assert Tree`	Asserts a tree of facts given the root.
`Assert New`	Assert a new fact.
`Assign`	Assign a value to a variable.
`Assign New`	Assign a value to a new variable.
`Expression`	Perform expression.
`Call`	Call a function.
`For`	Oracle RL, like Java, has a for loop. A for loop includes a type with a variable and a collection. The type and variable defines the loop variable that holds the collection value used within the loop. Collection is an expression that evaluates to a collection of the correct type for the loop variable. You can use a for loop to iterate through any collection. A return, throw, or halt may exit the action block.
`If`	Using the if else action, if the test is true, execute the first action block, and if the test is false, execute the optional else part, which may be another if action or an action block. Oracle RL, unlike Java, requires action blocks and does not allow a single semicolon terminated action.
`Modify`	Modify a data value associated with a matched fact.
`Retract`	Retract a fact.
`Return`	The return action returns from the action block of a function or a rule. A return action in a rule pops the ruleset stack, so that execution continues with the activations on the agenda that are from the ruleset that is currently at the top of the ruleset stack.
`rl`	Use an Oracle RL expression that you supply.
`synchronized`	As in Java, the synchronized action is useful for synchronizing the actions of multiple threads. The synchronized action block lets you acquire the specified object's lock, then execute the action-block, then release the lock.
`throw`	Throw an exception, which must be a Java object that implements java.lang.Throwable. A thrown exception may be caught by a catch in a try action block.
`try`	The try, catch, and finally in Oracle RL is like Java both in syntax and in semantics. There must be at least one catch or finally clause.
`while`	While the test is true, execute the action block. A return, throw, or halt may exit the action block.

Working with Decision Tables

Use Decision Tables to create and use business rules in an easy to understand format. Decision Tables provide an alternative to the IF/THEN rule format. Get an overview of the various components of a Decision Table such as conditions, conflicts, actions, and the various operations that you can perform on a Decision Table.

Introduction to Working with Decision Tables

Businesses invest in software to automate their business processes. Historically, this automation focused on the collection, presentation, and manipulation of data to facilitate human decision-making about that data. Increasingly, however, software designers and developers are called upon to automate the decision making process by putting detailed rules about business processes into software architectures. In addition, many enterprises are experiencing increasing pressure to make software systems more responsive to business changes.

In some cases, the role of writing and testing business rules is no longer assigned to software engineers, but is passed to trained business users. Alternatively, some organizations attempt to separate changes in the business behavior of software from the traditional software development cycles, and tie changes to business driven imperatives like product or sales cycles.

A Decision Table provides a mechanism for describing data processing tasks, especially when that description is done by business analysts rather than computer programmers.

The Decision Table format is intuitive for business analysts who are familiar with spreadsheets. The formal structure that a Decision Table provides makes it easier to author, understand, and change multiple similar rules and lets software check for rule completeness and consistency.

Oracle Business Rules Decision Tables provide the following features:

- Powerful Visualization: Compact and structured presentation. This visualization matches the way real world business policies are expressed: with many tables, declarative, and organized into simple steps.
- Error Prevention: Avoids incompleteness and inconsistency. Because a Decision Table is well structured, automated tools can check for conflicts, redundancy, and incompleteness to speed development of valid, consistent business rules.
- Modular Knowledge Organization: Group rules into a single table. A spreadsheet metaphor puts groups of rules that work together onto a single viewable pane. For example, if there are six rules that check an applicant's eligibility, it is more convenient to see all the rules than to view the rules as individual but related rules.
- Optimization of Rules and Performance Benefits: Oracle Business Rules Decision Tables provide automated features that can reduce the number of required rules, as compared to the IF/THEN rules (this is called rule coalescing).
- Rule Validation and Verification: Provides capabilities for ensuring the logical consistency of rules before deployment. Automated tools for checking conflicts or incompleteness help speed development of valid, consistent business rules.

Ease of verification and visualization are the major reasons for using Decision Tables.

For information, see *Working with Rulesets and Rules* in *Designing Business Rules with Oracle Business Process Management*.

What is a Decision Table?

A Decision Table displays multiple related rules in a single spreadsheet-style view. In Rules Designer a Decision Table presents a collection of related business rules with condition rows, rules, and actions presented in a tabular form that is easy to understand. Business users can compare cells and their values at a glance and can use Decision Table rule analysis features

by clicking buttons and selecting values in Rules Designer to help identify and correct conflicting or missing rules.

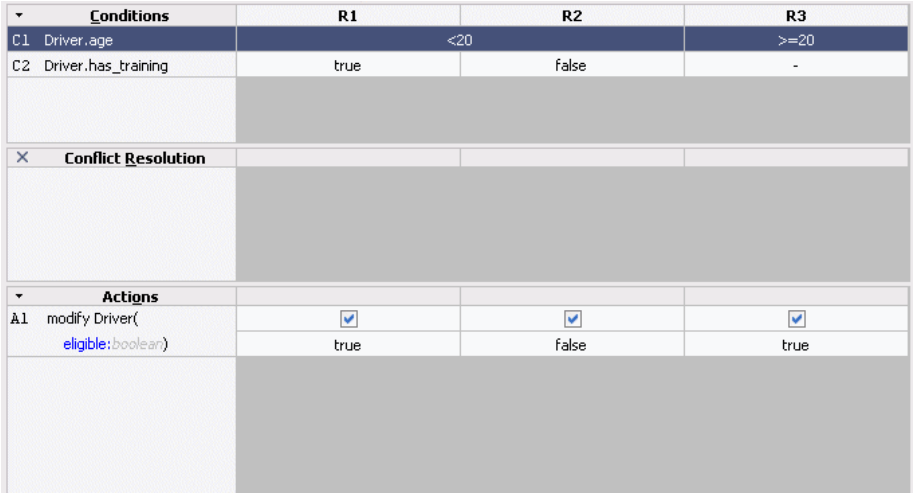
To help understand Decision Table concepts, consider a set of IF/THEN rules that determines if a driver is eligible for a license, and an equivalent Decision Table. Note if a driver has taken a driver training class then the driver has training certification.

The IF/THEN rules follow:

```
if driver.age < 20 and driver.has_training then training = true
if driver.age < 20 and driver.has_training = false then driver.eligible = false
if driver.age >= 20 then driver.eligible = true (do not care about training for this case)
```

Figure 1-46 shows a Decision Table representation of these rules that includes areas for Decision Table **Conditions** and **Actions**.

Figure 1-46 Sample Decision Table with Conditions and Actions



What You Need to Know About Decision Table Conditions

The **Conditions** area in a Decision Table includes one or more condition rows. Each condition row has a condition expression and, for each rule, a condition cell. A **condition expression** is an expression that you build in Rules Designer. The condition expression is often a fact property or a function result, but it can be any expression that has a type that can be associated with a value set. Test expressions are often used, such as `Driver.age<16`. These expressions are associated with the built-in boolean value set, with values `true` and `false`. The value or the range for a given **condition cell** takes its value or its range from one or more values or ranges in the associated LOV or Ranges value set.

For example, Figure 1-46 shows the condition expression for a `Driver` fact with the `Driver.age` property. The corresponding row in the Decision Table shows condition cells including values for the ranges `<20`, and `>=20`. The values in the cells come from the global value set named `driver_ages`.

Figure 1-46 also shows a condition row for the `Driver` fact with the `Driver.has_training` property. This condition row shows condition cells with the values, true, false, and -. The hyphen (–) means "do not care" (that is, `Driver.has_training` could be `true` or `false` in this case). The values for these

condition cells come from the default value set associated with boolean types (this consists of default values for the values `true` and `false`).

The '-' (don't care) value is useful to ensure that a decision table will not have gaps when new values are added to a value set. For example, if a valueset initially contains 1, 2, and otherwise, a rule matching otherwise will fire if the input is 3. But after 3 is explicitly added to the valueset, then otherwise no longer matches an input value of 3. If no rule contains a '-' for this input, then no rule will fire when the input value is 3 and the decision table is said to have a gap.

Use 'otherwise' when you explicitly want to match the 'otherwise' value in the valueset, and not any other value. 'Otherwise' is useful to avoid conflicts in a decision table. '-' is used to match any value, and will often cause conflicts. These conflicts can be automatically resolved using the 'auto override' conflict policy.

Decision Tables show rules in bucket order, and to change the order of rules you need to change the order of buckets in the value set. Thus, the order of the buckets in the value set associated with a condition row determines the order of the condition cells, and thus the order of the rules. You can control rule ordering in a Decision Table by changing the relative position of the buckets in an LOV value set associated with a condition row; however, you cannot reorder range buckets (values).

What You Need to Know About Decision Table Actions

Actions are associated with rules in a Decision Table. At runtime, when facts match for condition cells, the Rules Engine prepares to run the actions associated with the rule.

Table 1-19 shows the types of actions you can choose in the **Actions** area. Thus, in an action you can call a function, assert a new fact, retract a fact, or modify a fact, and so on. In the **Actions** area the cells corresponding to an individual action for a rule are called **action cells**.

Table 1-19 Decision Table Actions for Action Cells

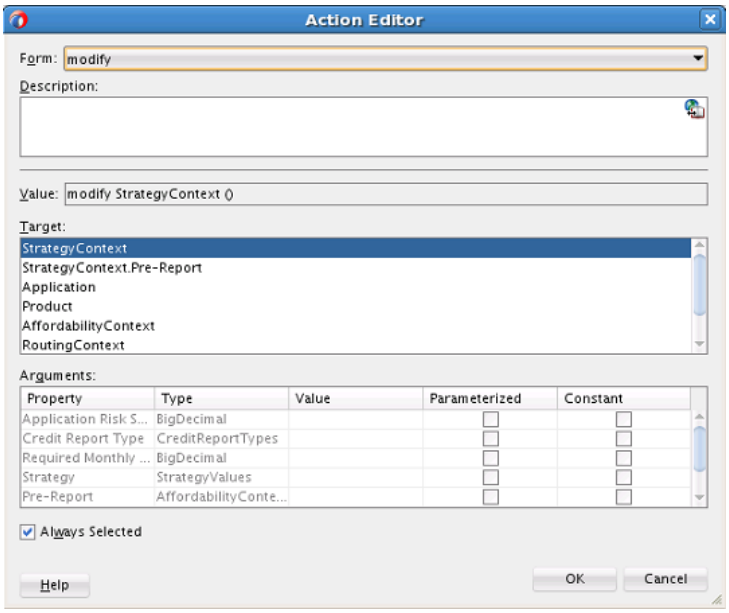
Action	Description
`assert new`	Assert a new fact.
`assign`	Assign a value to a variable.
`call`	Call a function.
`modify`	Modify a data value associated with a matched fact.
`retract`	Retract a fact.
`assert`	Assert a fact.
`assert tree`	Asserts a tree of facts given the root.
`assign new`	Assign a value to a new variable.
`expression`	Perform expression.
`return`	The return action returns from the action block of a function or a rule. A return action in a rule pops the ruleset stack, so that execution continues with the activations on the agenda that are from the ruleset that is currently at the top of the ruleset stack.
`throw`	Throw an exception, which must be a Java object that implements java.lang.Throwable. A thrown exception may be caught by a catch in a try action block.

When you add multiple actions the actions that you add in the **Actions** area are ordered; actions appearing in the higher rows run before actions in the following rows.

The Decision Table actions such as `modify` can refer to facts matched in the condition cells. For example, given a Decision Table with condition rows on the `Driver` fact that includes condition rows for `Driver.age` and `Driver.has_training`, actions can modify the property `Driver.eligible` and you can specify a value for `Driver.eligible` for each action cell.

Certain types of actions in the **Actions** area include a **Parameterized** check box. This check box specifies that a property from the action can have its value set in the action cell associated with a rule in the Decision Table. When the parameterized check box is selected, the value you supply for the expression value in the action, in the **Actions** area, becomes the default value for the property if a value is not supplied in the action cell. For example, see Figure 1-47 where the value `false` is assigned as the default value for the action property `eligible`.

Figure 1-47 Action Editor Showing Parameterized Action with Default Value



What You Need to Know About Decision Table Rules

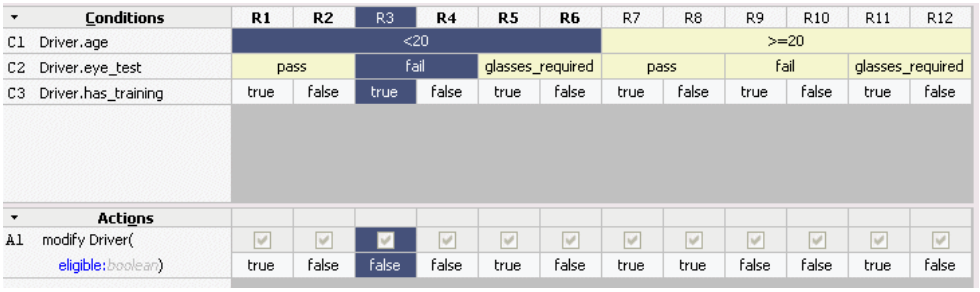
A ruleset contains a Decision Table; this provides a way to group the Decision Table along with IF/THEN rules. When rules and Decision Tables are grouped in a ruleset, the IF/THEN rules and the Decision Table rules all execute as a set of interrelated rules.

A rule in a Decision Table is not named. Although Rules Designer shows rules in a Decision Table with labels, for example, R1, R2, and R3, these rule labels are not names for individual rules but are labels derived from the current ordering of the rules in the Decision Table. Thus, a rule with the label R1 could be moved to position 3 and then Rules Designer relabels this rule R3.

Rules in a Decision Table are organized as a table that contains a tree of condition cells. The condition cells in the first row span the cells of later condition rows. A parent cell in row *i* spans its children in row *i*+1.

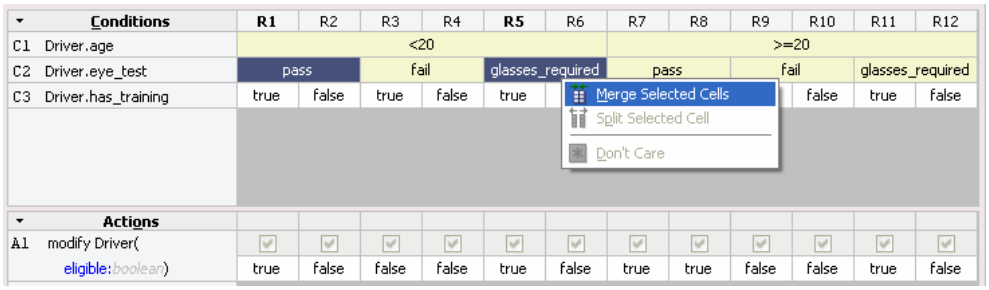
Figure 1-48 shows rules in a Decision Table where each rule consists of one cell from each row in the **Conditions** area, and an associated action cell in the same column in the **Actions** area. Figure 1-48 shows the rule with the label R3 defined by the first cell from condition 1 (the `Driver.age < 20` value), the second cell from condition 2 (the `Driver.eye_test` equals `fail` value), and the third cell from condition 3 (the `Driver.has_training` equals `true` value). Likewise for each of the other rules, R1 to R12, there is a unique path through the Decision Table.

Figure 1-48 Rules in a Decision Table



As shown in Figure 1-48, it is significant for a cell to be a parent of another cell and a parent cell spans lower cells. In the **Conditions** area, when condition cells have the same parent condition cell the cells are called **siblings**. Certain operations only apply for condition cells that are siblings. For example, Figure 1-49 shows two sibling cells that are selected; with these cells selected the **Merge Selected Cells** operation is valid. For these cells, the corresponding value set with the value `fail` for `Driver.eye_test` is also a sibling (as shown in the R3 and R4 columns in Figure 1-49). For more information, see How to Merge or Split Conditions in a Decision Table.

Figure 1-49 Sibling Condition Cells in a Decision Table



Rules Designer lets you easily reorder rows by selecting the row and clicking a **Move** button. By reordering rows in the **Conditions** area you can perform operations on condition cells at the desired granularity. Thus, the move operations can assist you when you want to split, merge, or assign certain values that might only be appropriate at a particular level in the tree, depending on the location of a condition cell or depending on the location of the parent, children, or siblings of a condition cell.

Understanding Condition Cell Values

By default, when you create a condition row, Rules Designer creates a single condition cell and assigns the "`?`" value to the cell. A condition cell with the value "`?`" indicates that the

value of the cell is undefined in the value set. For example, Figure 1-50 shows a "?" value for `StrategyContext`. Note that contiguous value ranges in a condition cell are combined. For example, if you select `<20` and `[20..40]` it will display as `<=40`.

Figure 1-50 Sample Decision Table Showing Undefined in Condition Cell

Conditions	R1	R2	R3	R4	R5	R6	R7
C1 Existing Customer	true			false			
C2 Application Risk Score	<100	[100..120]	>120	<80	[80..90)	[90..110]	>110
C3 StrategyContext	?	?	?	?	?	?	?
Actions							
A1 modify StrategyContext	☑	☑	☑	☑	☑	☑	☑
Pre-Report Risk Category:R...	HIGH	MED	LOW	REJECT	HIGH	MED	LOW

Understanding Action Cell Values

In the Decision Table **Actions** area you can specify that an action cell "do nothing." In this case, clear the action cell. When the action cell check box is cleared, this means do not perform this action when the pattern matches for the specified condition values in the Decision Table. Thus, for each action cell you can specify whether the rule associated with the action cell should activate the action, or does not perform the action.

In a Decision Table, when a condition cell represents a value that has been removed from the value set, Rules Designer provides a validation warning such as the following:

```
RUL-05831: Decision table value reference not found
```

To fix this type of validation warning you can do one of the following:

- Define a value by double-clicking the condition cell and selecting one or more values from the list.
- Add the missing value to the value set or associate the condition with another value set that contains the missing value.

What You Need to Know About Decision Table Loops

A Decision Table loop occurs when the value for a condition row is changed by an action. Loops can occur across the rules in a single Decision Table or spread over several Decision Tables, or spread over rules and Decision Tables in the same ruleset. Try not to create Decision Table actions that modify fact properties that are used in Decision Table conditions. This could cause an infinite loop.

> **Note:**
>
> You can prevent infinite loops by using the rule firing limit on the containing decision function.

Creating Decision Tables

You add a Decision Table by performing several steps.

These steps include creating a Decision Table, creating value sets, and then adding conditions and actions to Decision Table, and using the Decision Table to operate to validate, correct, and modify the Decision Table.

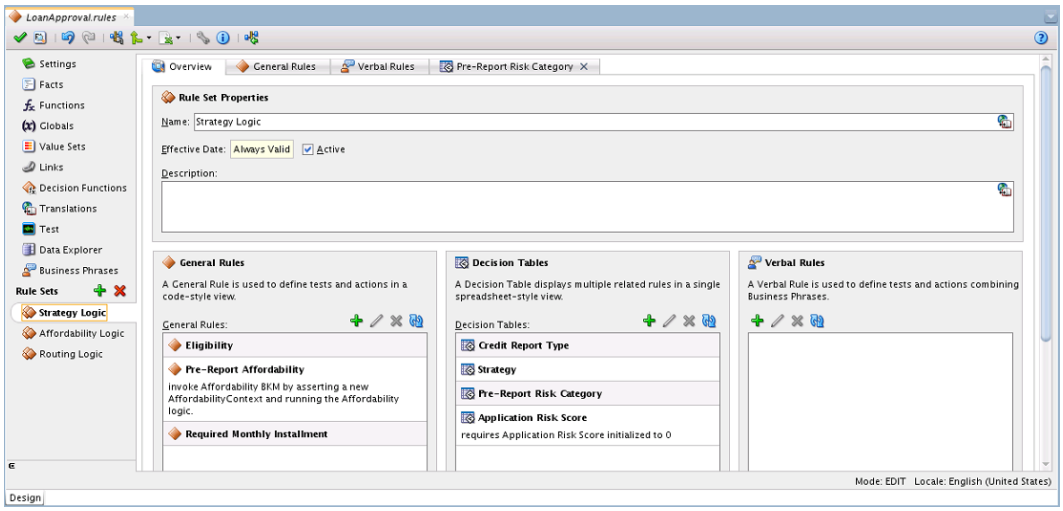
How to Create a Decision Table

To work with a Decision Table, start by creating a Decision Table in a ruleset.

To create a decision table:

1. From Rules Designer select an existing ruleset from the rulesets tab or create a ruleset by clicking **Create Rule Set...**.
2. Click **Create** from the **Decision Tables** area on the **Overview** tab, as shown in Figure 1-51. This creates a Decision Table.

Figure 1-51 Adding a Decision Table



> **Note:**
>
> When you add a Decision Table the rules validation log displays validation warnings. The Decision Table is not complete and does not validate without warnings until you add conditions and actions to the Decision Table.

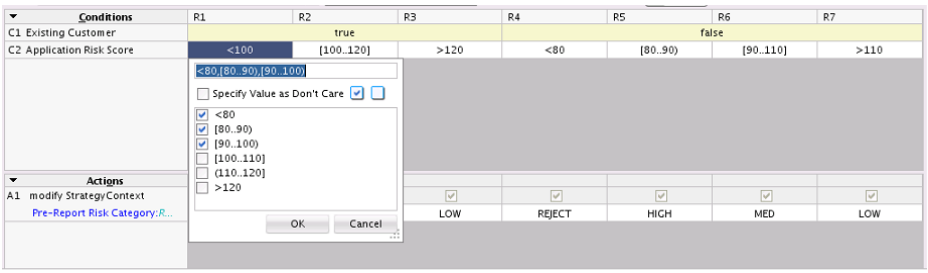
How to Add Condition Rows to a Decision Table

A Decision Table includes a **Conditions** area where you specify Decision Table condition rows. The condition rows determine the facts that the Oracle Rules Engine matches at runtime. To create a Decision Table you need to add one or more condition rows to the Decision Table.

To add condition rows to a decision table:

1. From Rules Designer select a ruleset from the **Rulesets** navigation tab and select the Decision Table where you want to add conditions.
2. In the Decision Table area, from the list next to the **Add** button select **Condition**.
3. In the **Conditions** area, double-click **<edit-condition>** to display the navigator to select or enter an expression as shown in Figure 1-52.

Figure 1-52 Adding a Condition to a Decision Table

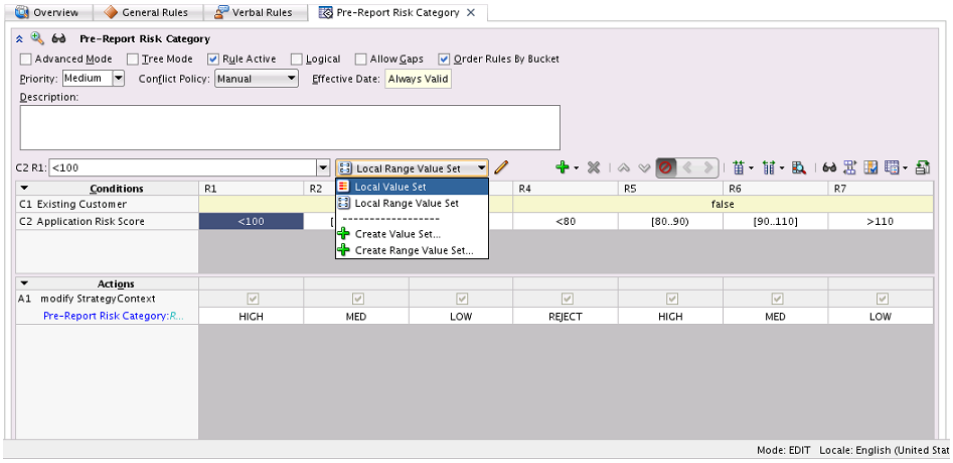


4. Enter an expression by clicking in the navigator to select a variable or click the **Expression Builder** button to display the **Expression Builder** window. The **Expression Builder** lets you build expressions.
5. Each condition row requires a value set from which to draw the values for each cell. When the value you select has an associated global value set, then by default the value set is associated with the condition row.
6. Repeat **Step 2** through **Step 5**, as required to add additional condition rows in the Decision Table.

How to Use or Specify the Value Set for a Decision Table Condition

1. Each condition row requires a value set from which to draw the values for each cell. When the value you select has an associated global value set, then by default the value set is associated with the condition row.
2. If there is no global value set associated with the value, then after you add a condition row to a Decision Table you need to specify either a Local List of Values or a Local List of Ranges value set to associate with the condition row, or specify an existing global value set. To add a value set for the condition, in the **Conditions** area select the condition and then select from the value set list to associate a value set, as shown in Figure 1-53. The value set list includes available global value sets of the appropriate type.

Figure 1-53 Specifying a Value Set For a Condition Row in a Decision Table



3. If you do not specify a global value set, then you can create and use a local value set by selecting either **Local Value Set** or **Local Range Value Set** to create and use the specified type of value set.
4. Repeat **Step 2** through **Step 3**, as required to define additional condition rows in the Decision Table.

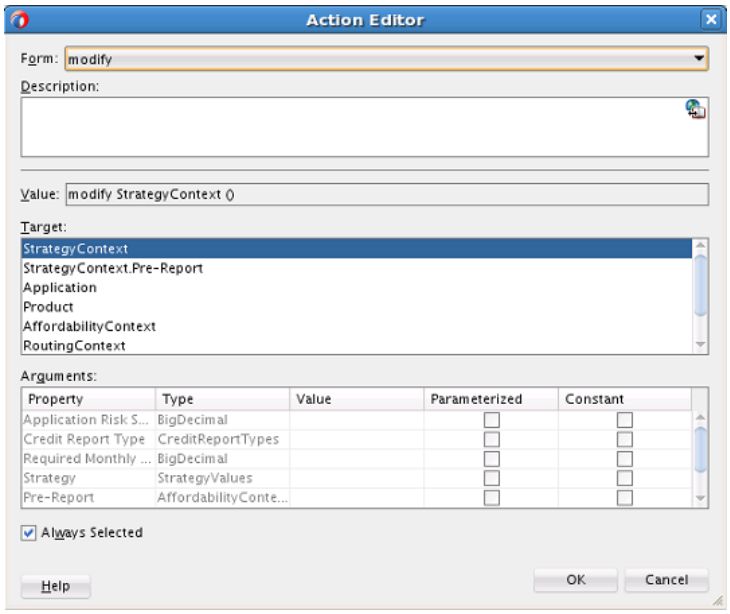
How to Add Actions to a Decision Table

A Decision Table includes an **Actions** area where you specify Decision Table actions. The actions determine actions for rules in a Decision Table. To create a valid Decision Table, add actions to a Decision Table. For each action cell, where specific values apply, set the values for the action cells. For each action cell, when the action does not apply to the rule, deselect the action cell. By default when you add an action to a Decision Table, actions for all the rules are unselected

To add actions to a decision table:

1. From Rules Designer select a ruleset from the **Rulesets** navigation tab and select the Decision Table where you want to add actions.
2. From the list next to the **Add** button, select **Action** and select an available action from the list. Table 1-19 lists the available actions. For example, select **Modify**. Rules Designer displays the **Action Editor** dialog as shown in Figure 1-54.

Figure 1-54 Adding an Action to a Decision Table



3. In the **Action Editor** dialog select the action target in the **Target** area. This specifies the data model object the action applies to.
4. For the specified target, as needed to make the action do what is required, modify the fields in the **Arguments** table. In the Action Editor dialog the **Arguments** table includes the fields shown in Table 1-20.

Table 1-20 Action Editor Dialog Arguments Fields

Field	Description
Property	Displays the property names for the specified target.
Type	Displays the type for the property.
Value	Select the default value for the action from the list of available actions. The specified value applies to either the entire action, as the default value, or when a particular action cell is selected, the value specified applies for that particular action cell.
Parameterized	This specifies a parameterized value. A parameterized value displays in a Decision Table action cell. When you select parameterized value for a property, this generally means that each rule supplies a different parameter value.
Constant	Select to specify a constant value.

5. In the Action Editor dialog, to select action cells for all the rules, select the **Always Selected** check box.
6. Repeat **Step 2** through **Step 5**, as required to define additional actions for the Decision Table.

How to Set Values for Action Cells in a Decision Table

To set values for action cells:

1. From Rules Designer, select a ruleset from the **Rulesets** navigation tab and select the Decision Table where you want to specify action cell values.
2. In the **Actions** area, check that the appropriate action cells are selected. If the **Always Selected** check box is specified in the Action Editor dialog, then all action cells should be selected. If **Always Selected** is not selected, then select the appropriate action cells using the action cell check box.
3. In the **Actions** area, enter the appropriate value for parameterized properties for each selected action cell. To do this select the action cell property cell, and either enter a value, select a value from the list, or click the **Expression Builder** button to use the Expression Builder dialog.

How to Deselect an Action Cell in a Decision Table

To deselect an action cell:

1. From Rules Designer select a ruleset from the **Rulesets** navigation tab and select the Decision Table where you want deselect an action cell.
2. In the **Actions** area, select the action cell and deselect the check box in the action cell. You are not allowed to deselect action cell values when **Always Selected** is selected for the action.

When you add actions, you may need to change the order of the actions. In Rules Designer you can use the **Move Down** button or **Move Up** button to change the order of actions.

How to Add a Rule to a Decision Table

You can add a rule to a Decision Table. Rules Designer adds a column for the rule to the left of the existing rules and each condition cell is initialized to "?", which actually means a validation error prompting you to populate the cell with relevant values.

To add a rule to a decision table:

1. From Rules Designer select a ruleset from the **Rulesets** navigation tab and select the Decision Table where you want to add the rule.
2. From the list next to the **Add** button, select **Rule**.
3. Enter values for the condition cells. Notice that the new rule is added as the first rule of the Decision Table on the left and the other rules have moved as required to keep the values in their defined order.
4. Enter values for the action cells.

The **Order Rules By Bucket** check box under the Advanced Settings of a Decision Table is selected by default. In this case, the Decision Table layout changes automatically on adding new rules.

When you add a new rule to a Decision Table, the new rule is added as the first rule of the Decision Table and the other rules move as required to keep the values in their defined order. This is because **Order Rules By Bucket** is enabled, which means rule ordering in a Decision Table is set according to the relative position of values associated with a condition

expression. If **Order Rules By Bucket** is not enabled when you add a rule, the new rule is added as the last rule of the Decision Table. In either case, the cells in the new rule column have "?" symbols, indicating the cells do not have values yet.

> **Note:**
>
> When **Order Rules By Bucket** is selected, the rules are ordered and duplicate rules (rules with exactly the same values) are combined. So, you cannot add two rules without any values to a Decision Table, because in that case, the rules are duplicates and would immediately be combined. When **Order Rules By Bucket** is cleared, then duplicate rules are allowed.

In addition, the **Move** buttons pertaining to a rule column are also enabled. You can use them to reposition rules. Use the **Flip the Table Rows and Columns** button to change the view of the Decision Table. This also affects the Move buttons: the move direction may be **Up** or **Down**, **Left** or **Right**. The **Merge**, **Compact** and **Span** options are also enabled. You can also cut, copy, or paste rules.

For more information, see Introduction to Decision Table Operations.

How to Define Tests in a Decision Table

You can define tests in a Decision Table. The tests must evaluate to true for any rule in the decision table to fire. For more information about defining tests and working with rule conditions, see *Working with Rules* in *Designing Business Rules with Oracle Business Process Management*.

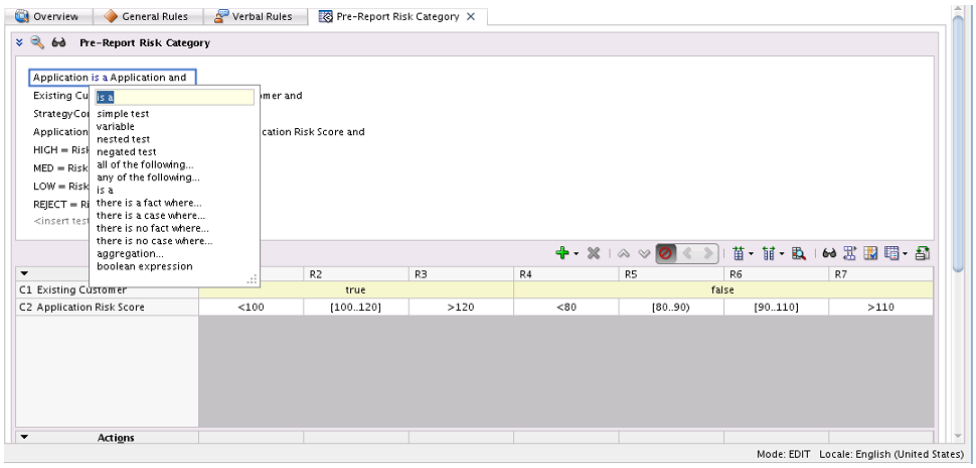
You can use the **Data Explorer** tab to find fact types and value sets in the data model.

To add tests to a Decision Table:

1. From Rules Designer, select a ruleset from the **Rulesets** navigation tab and select the Decision Table where you want to add the rule.
2. Click the **Show Patterns/Tests** button (magnifying glass) left of the Decision Table name. If **Advanced Mode** is selected, clear the check box.
3. Select any of the options according to your requirements, as shown in the following image:

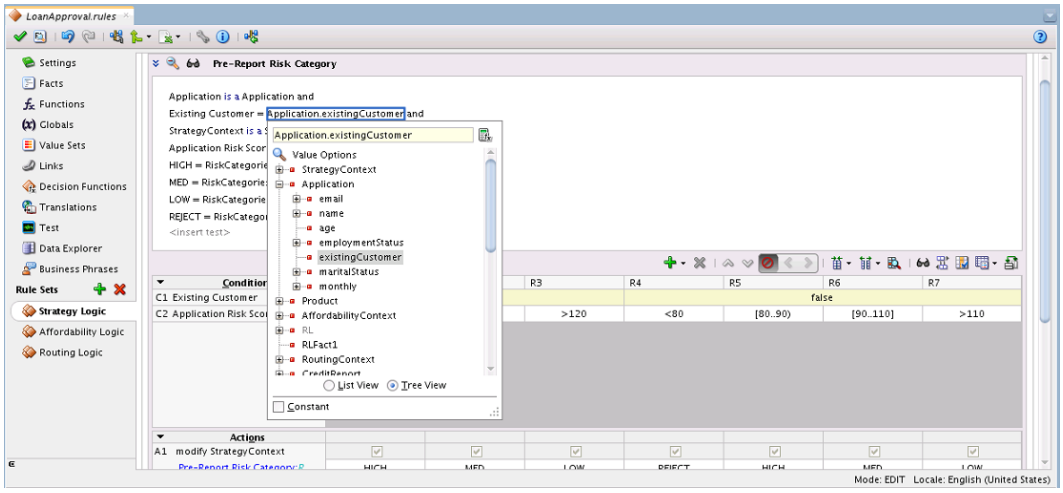
> **Note:**
>
> Variables without any tests are often used so that the variables can be used in the decision table conditions and actions.

Figure 1-55 Options List



4. Click the left and the right **<operand>** to enter the operand values, and the operator list to select an operator, as in the following image:

Figure 1-56 Value Options List



For more information about writing tests, see *Testing and Validating Business Rules* in *Designing Business Rules with Oracle Business Process Management*.

Creating and Running an Oracle Business Rules Decision Table Application

The Order Approval application demonstrates the integration of a SOA composite application with Oracle Business Rules and the use of a Decision Table.

In this application a process is modeled that uses the decision component to:

- Process rules from XML inputs including: a credit score and the annual spending of a customer, and the total cost of the incoming order.
- Provide output that determines if an order is approved, rejected, or requires manual processing.

To complete this procedure, you need to:

- Obtain the Source Files for the Order Approval Application
- Create an Application for Order Approval
- Create a Business Rule Service Component for Order Approval
- View Data Model Elements for Order Approval
- Add Value Sets to the Data Model for Order Approval
- Associate Value Sets with Order and CreditScore Properties
- Add a Decision Table for Order Approval
  - Split the Cells in the Decision Table and Add Actions
  - Compact the Decision Table
  - Replace Several Specific Rules with One General Rule
  - Add a General Rule
- Check Dictionary Business Rule Validation Log for Order Approval
- Deploy the Order Approval Application
- Test the Order Approval Application

How to Obtain the Source Files for the Order Approval Application

The source code for Oracle Business Rules-specific samples and SOA samples are available online in the Oracle SOA Suite Samples and Tutorials page.

To work with the Order Approval application, you first need to obtain the `order.xsd` schema file either from the sample project that you obtain online or you can create the schema file and create all the application, project, and other files in Oracle JDeveloper. You can save the schema file provided in the following example locally to make it available to Oracle JDeveloper.

The following example shows the `order.xsd` schema file.

```
<?xml version="1.0" ?>
<schema attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://example.com/ns/customerorder"
        xmlns:tns="http://example.com/ns/customerorder"
        xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="CustomerOrder">
    <complexType>
      <sequence>
        <element name="name" type="string" />
        <element name="creditScore" type="int" />
        <element name="annualSpending" type="double" />
        <element name="value" type="string" />
        <element name="order" type="double" />
      </sequence>
    </complexType>
  </element>
  <element name="OrderApproval">
    <complexType>
      <sequence>
        <element name="status" type="tns:Status"/>
      </sequence>
    </complexType>
  </element>
  <simpleType name="Status">
    <restriction base="string">
        <enumeration value="manual"/>
        <enumeration value="approved"/>
```

```
        <enumeration value="rejected"/>
    </restriction>
  </simpleType>
</schema>
```

How to Create an Application for Order Approval

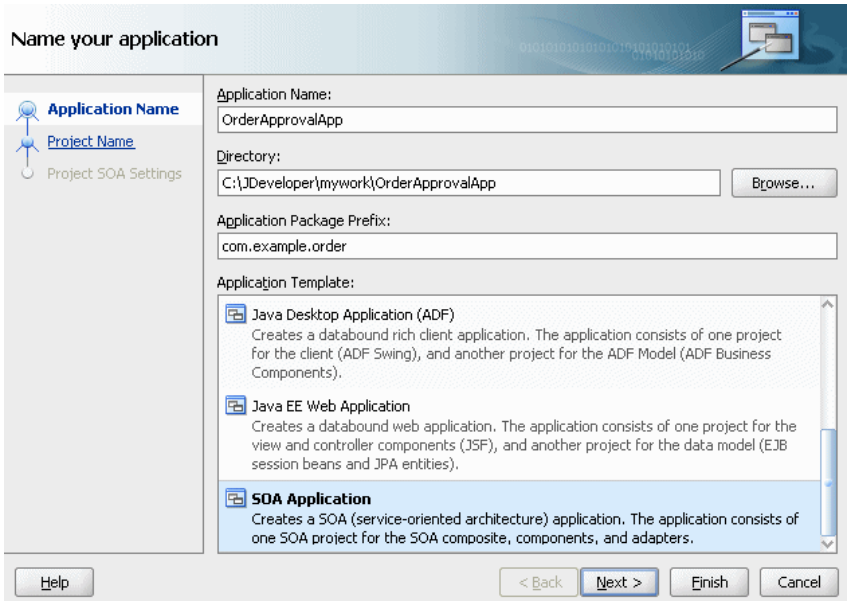
To work with Oracle Business Rules, you first create an application in Oracle JDeveloper.

To create an application for order approval:

1. In the Application Navigator, click **New Application**.
2. In the Name your application dialog, enter the name and location for the new application.
   a. In the **Application Name** field, enter an application name. For example, enter `OrderApprovalApp`.
   b. In the **Directory** field, specify a directory name or accept the default.
   c. In the **Application Package Prefix** field, enter an application package prefix, for example `com.example.order`.

      The prefix, followed by a period, applies to objects created in the initial project of an application.
   d. For a SOA composite with Oracle Business Rules, in the Application Template area select SOA Application for the application template. For example, see Figure 1-57.
   e. Click **Next**.

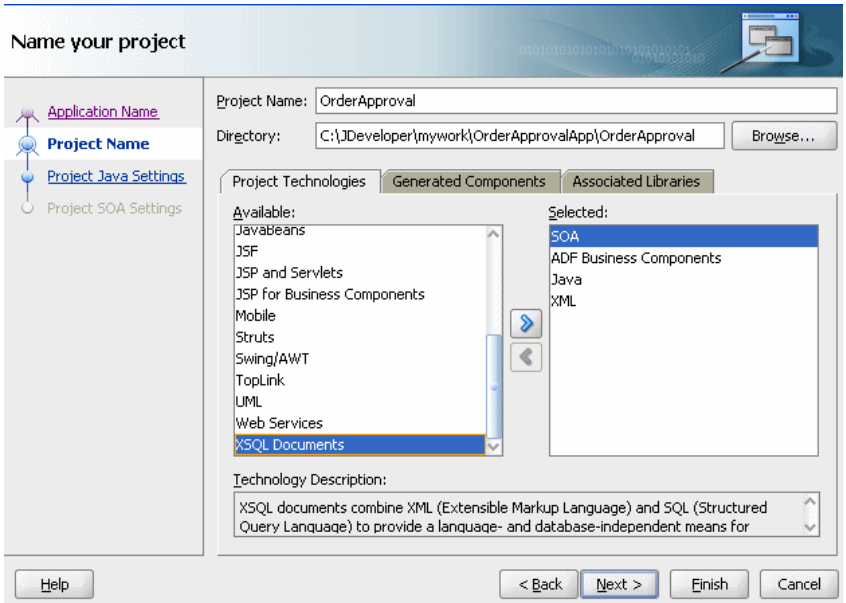
Figure 1-57 Adding the Order Approval Application



3. In the Name your project page enter the name and location for the project.
   a. In the **Project Name** field, enter a name. For example, enter `OrderApproval`.
   b. Enter or browse for a directory name, or accept the default.
   c. For an Oracle Business Rules project, in the **Project Technologies** area ensure that SOA, ADF Business Components, Java, and XML are in the **Selected** area on the

Project Technologies tab, as shown in Figure 1-58. If an item is missing, select it in the **Available** pane and add it to the **Selected** pane using the **Add** button.

Figure 1-58 Adding a Project to an Application



4. Click **Finish**.

How to Create a Business Rule Service Component for Order Approval

After creating a project in Oracle JDeveloper you need to create a Business Rule Service component within the project. When you add a business rule you can create input and output variables to provide input to the service component and to obtain results from the service component.

To use business rules with Oracle JDeveloper, you do the following:

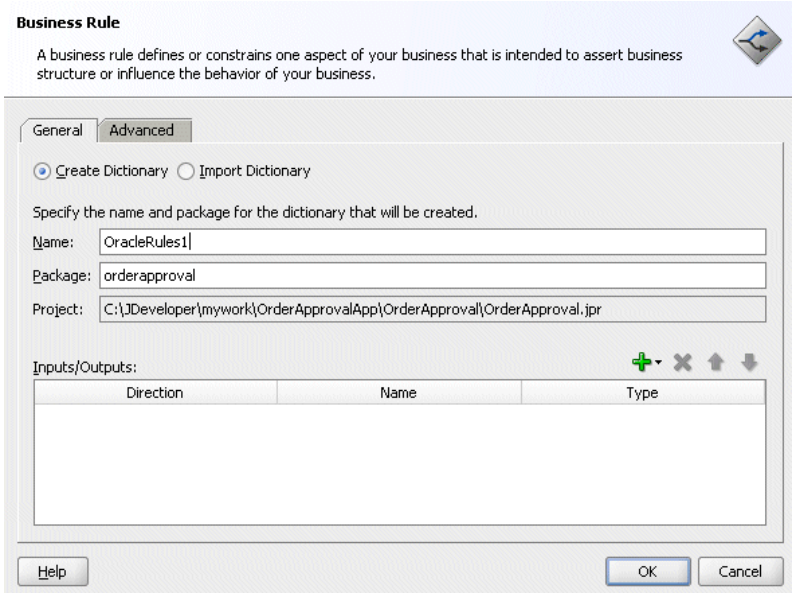
- Add a business rules service component
- Create input and output variables for the service component
- Create an Oracle Business Rules dictionary in the project

To create a business rule service component:

1. In the Application Navigator, in the **OrderApproval** project expand **SOA Content** and double-click `composite.xml` to launch the SOA composite editor (this may already be open after you create the project).
2. From the Component Palette, drag-and-drop a **Business Rule** from the **Service Components** area of the SOA menu to the **Components** lane of the `composite.xml` editor.

   Oracle JDeveloper displays a Create Business Rules page, as shown in Figure 1-59.

Figure 1-59 Adding a Business Rule Dictionary with the Create Business Rules Dialog



3. To add an input, from the list next to the **Add** button select **Input** to enter input for the business rule.
4. In the Type Chooser dialog, click the **Import Schema File**... button. This displays the Import Schema File dialog.
5. In the Import Schema dialog click **Browse Resources** to choose the XML schema elements for the input variable of the process. This displays the SOA Resource Lookup dialog.
6. In the SOA Resource Lookup dialog, navigate to find the `order.xsd` schema file and click **OK**.
7. In the Import Schema File dialog, make sure **Copy to Project** is selected, as shown in Figure 1-60. Click **OK**.

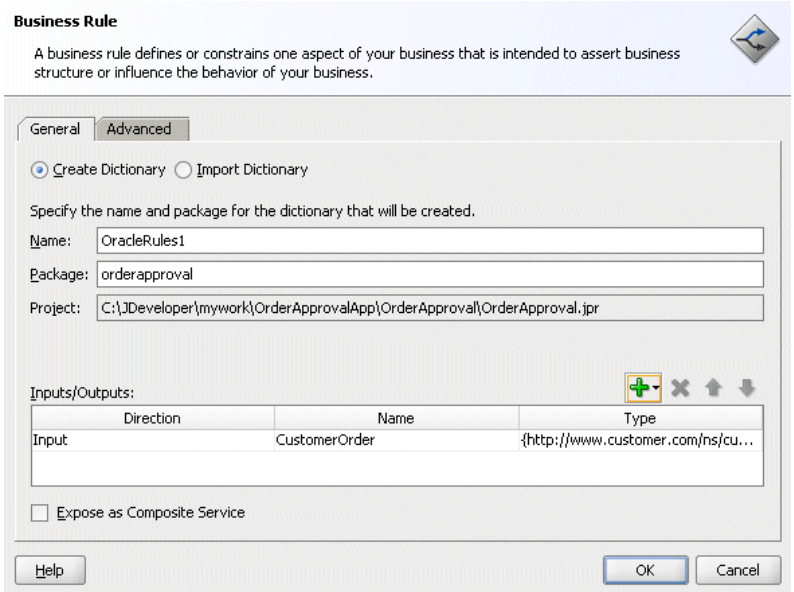
Figure 1-60 Importing the Order.xsd Schema File

8. If the Localize Files dialog displays, click **OK** to copy the schema to the composite process directory.
9. In the Type Chooser, navigate to the Project Schemas Files folder to select the input variable.

   For this example, select `CustomerOrder` as the input variable.

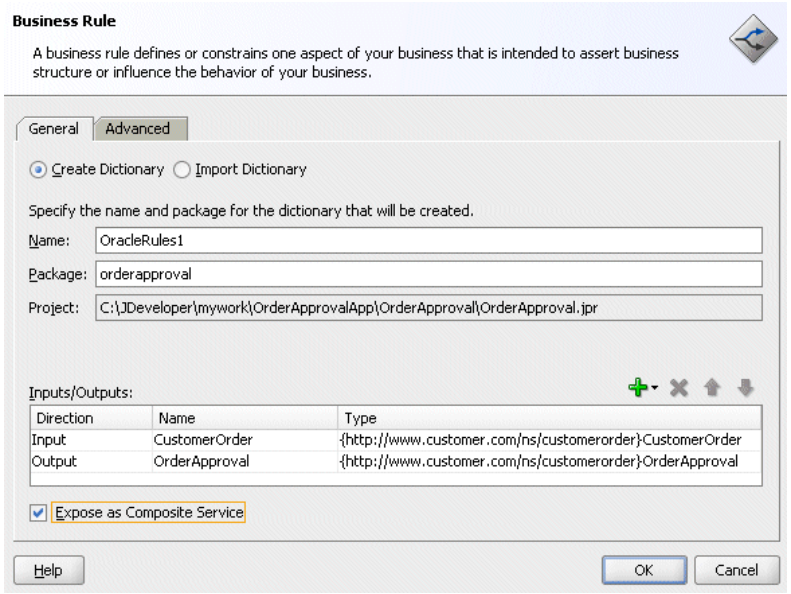
   On the Type Chooser window, click **OK**. This displays the Create Business Rules dialog, as shown in Figure 1-61.

Figure 1-61 Create Business Rules Dialog with CustomerOrder Input



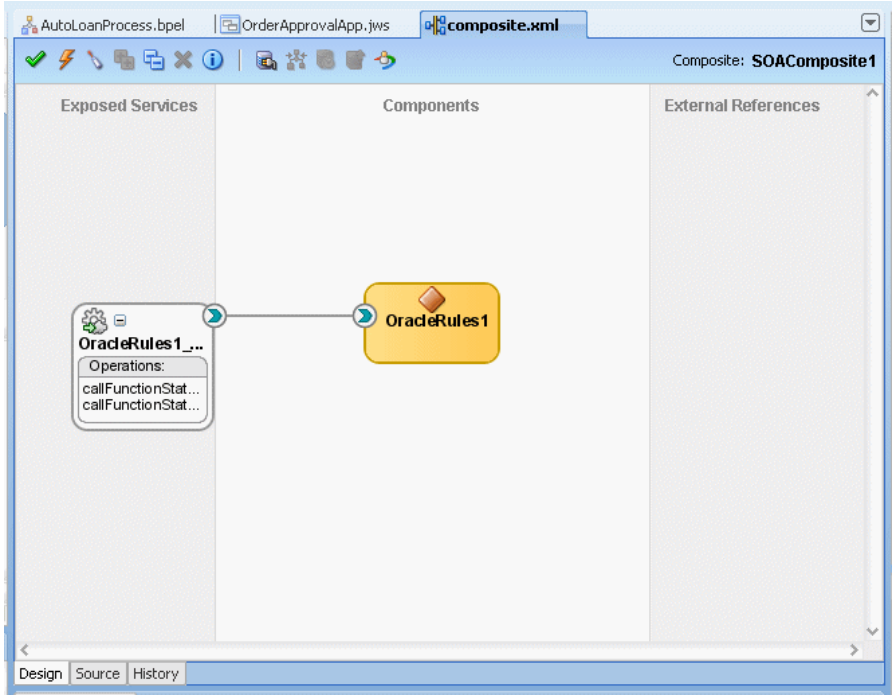
10. In a similar manner, add the output fact type `OrderApproval` from the imported `order.xsd`.
11. In the Create Business Rules dialog, select **Expose as Composite Service**, as shown in Figure 1-62.

Figure 1-62 Create Business Rules Dialog with Input and OrderApproval Output



Click **OK**. This creates the Business Rule component and Oracle JDeveloper shows the Business Rule in the canvas workspace, as shown in Figure 1-63.

Figure 1-63 Business Rules Component in OrderApproval Composite



The business rule service component enables you to integrate your SOA composite application with a business rule. This creates a business rule dictionary and enables you to execute business rules and make business decisions based on the rules.

How to View Data Model Elements for Order Approval

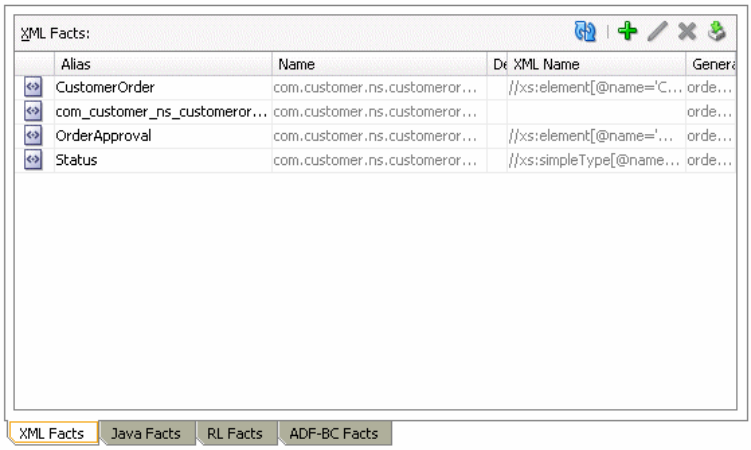
Before adding rules you need to create the Oracle Business Rules data model. The data model contains the business data definitions (types) and definitions for facts that you use to create rules. For example, for this sample the data model includes the XML schema elements from `order.xsd` that you specify when you define inputs and outputs for the business rule activity.

At times when you work with Rules Designer to create a rule or a Decision Table, you may need to create or modify elements in the data model.

To view data model elements for Oracle business rules:

1. Select the composite tab with the value **composite.xml**, and in the Components lane select the business rule (this surrounds the component, **OracleRules1** with a dashed selection box).
2. Double-click the selection box to launch Rules Designer.
3. In Rules Designer select the **Facts** navigation tab.
4. Select **XML Facts** tab in the **Facts** navigation tab as shown in Figure 1-64.

Figure 1-64 Opening a Business Rules Dictionary with Rules Designer



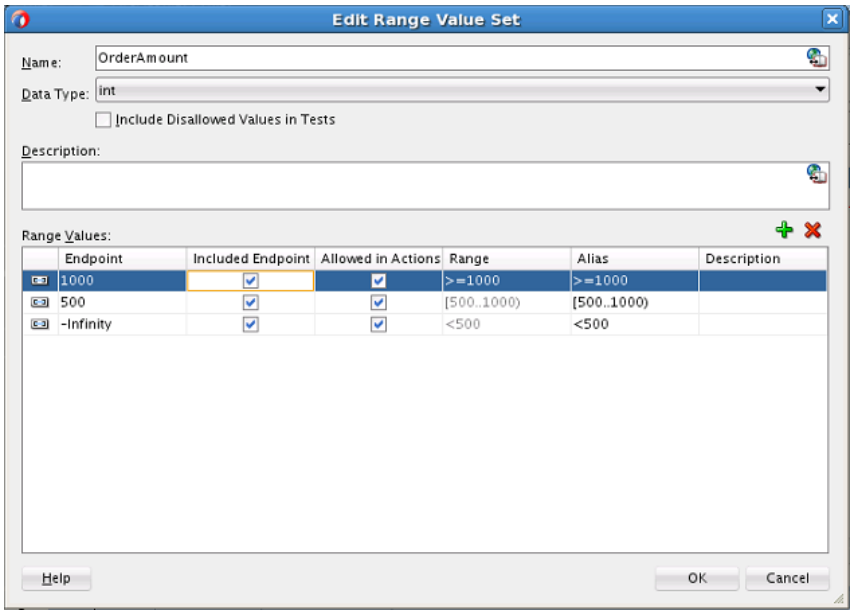
How to Add Value Sets to the Data Model for Order Approval

To use a Decision Table you need to define value sets that specify how to draw values for each cell for the conditions that constitute the Decision Table. For this example the value sets are defined with a list of ranges that you define in Rules Designer.

To add OrderAmount value set to the data model:

1. In Rules Designer, select the **Value Sets** navigation tab.
2. From the drop down next to the **Create Value Set...** button, select **Range Value Set**.
3. In the **Name** field, enter `OrderAmount`. Press **Enter** to accept the name.
4. Double-click the **OrderAmount** value set icon to display the **Edit Range Value Set** dialog.
5. Click **Add Value** to add a value.
6. Click **Add Value** again to add another value.
7. In the **Range Values** area, in the top **Endpoint** field enter 1000 for the endpoint value.
8. In the **Range Values** area, for the middle bucket in the **Endpoint** field enter 500 for the endpoint value.
9. In the **Included Endpoint** field for each value set ensure the check box is selected, as shown in Figure 1-65.

Figure 1-65 Adding the OrderAmount Value Set



10. Modify the **Alias** field for each value to **High, Medium, and Low**. Click **OK**.

How to Add CreditScore Value Set to the Data Model

To add CreditScore value set:

1. In Rules Designer select the **Value Sets** navigation tab.
2. From the drop down next to the **Create Valueset...** button, select **List of Ranges**.
3. In the **Name** field, enter `CreditScore`.
4. Double-click the **CreditScore** valueset icon to display the Edit Valueset dialog.
5. Click **Add Value** to add a value.
6. Click **Add Value** again to add another value.
7. In the top valueset, in the **Endpoint** field enter `750`.
8. For the middle valueset, in the **Endpoint** field enter `400`.
9. In the **Included Endpoint** field for each valueset, ensure the check box is selected.
10. Modify the **Alias** field for each endpoint value to **solid** for 750, **avg** for 400, and **risky** for -Infinity. Click **OK**.

How to Associate Value Sets with Order and CreditScore Properties

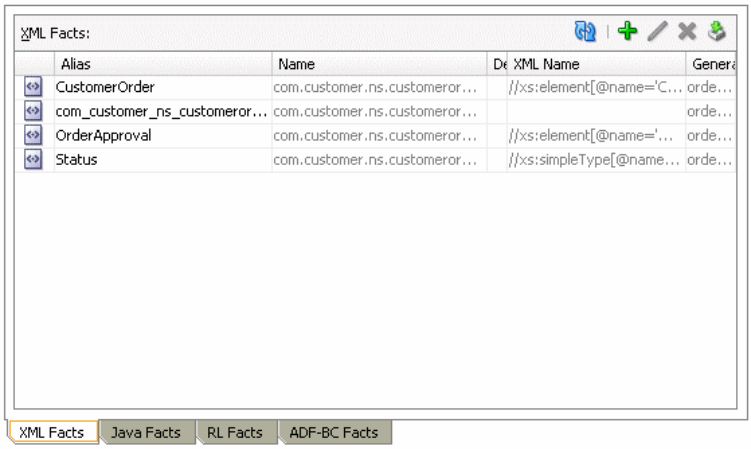
To prepare for creating Decision Tables you can associate a value set with fact properties in the data model. In this way condition cells in a Decision Table **Conditions** area can use the valuesets when you create a Decision Table.

Note that the `OrderApproval.status` property is associated with the `Status` value set when the `OrderApproval` fact type is imported from the XML schema. In the schema, `Status` is a restricted `String` type and is therefore represented as an enum valueset. Rules Designer creates the status value set.

To associate value sets with Order and CreditScore properties:

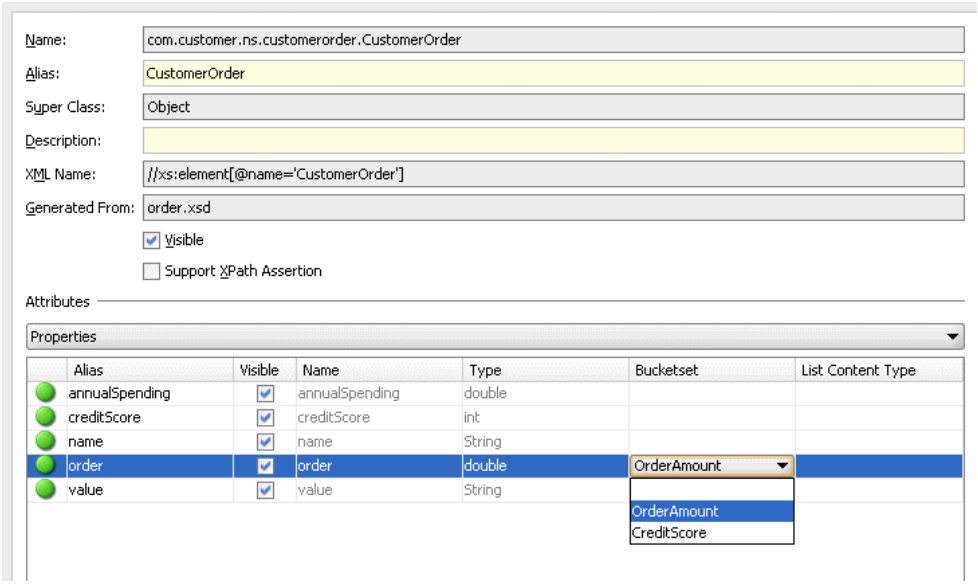
1. In Rules Designer select the **Facts** navigation tab.
2. Select the **XML Facts** tab in the **Facts** navigation tab as shown in Figure 1-66.

Figure 1-66 Opening a Business Rules Dictionary with Rules Designer



3. Select the type you want to modify. For example in the XML Facts table double-click the icon next to the **CustomerOrder** entry. This displays the Edit XML Fact dialog.
4. In the Edit XML Fact dialog, in the **Properties** table in the **Value Set** column select the cell for the appropriate property and from the list select the valueset you want to use. For example, for the property **order** select the **OrderAmount** valueset, as shown in Figure 1-67.

Figure 1-67 Associating the OrderAmount Valueset with CustomerOrder.order



5. In a similar manner, for the property **creditScore** select the **CreditScore** valueset.
6. Click **OK**.

How to Add a Decision Table for Order Approval

You create a Decision Table to process input facts and to produce output facts, or to produce intermediate conclusions that Oracle Business Rules can further process using additional rules or in another Decision Table.

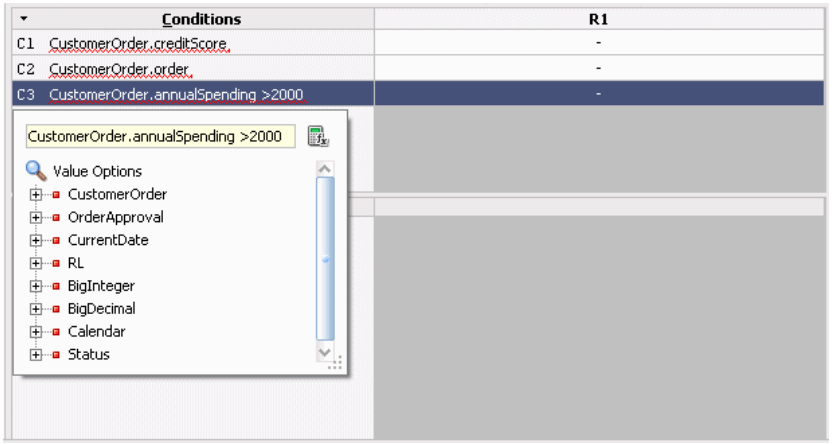
While you work with rules you can use the rule validation features in Rules Designer to assist you. Rules Designer performs dictionary validation when you make any change to the dictionary. To show the validation log window, click the **Validate** button or select **View**>**Log** and select the **Business Rule Validation** tab. If you view the rules validation log you should see warning messages. You remove these warning messages as you create the Decision Table.

To use a Decision Table for rules in this sample application you work with facts representing a customer spending level and a customer credit risk for a particular customer and a particular order. Then, you use a Decision Table to create rules based on customer spending, the order amount, and the credit risk of the customer.

To add a Decision Table for order approval:

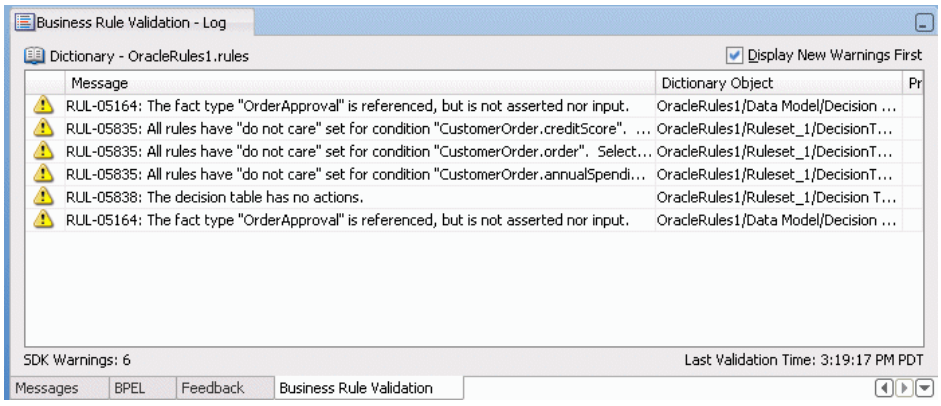
1. In Rules Designer, select **Ruleset_1** under the **Rulesets** navigation tab.
2. Click the **Add** button and from the list and select **Create Decision Table**.
3. In the Decision Table, click the **Add** button and from the list select **Condition**.
4. In the Decision Table, double-click **<edit condition>**. Then, in the navigator expand **CustomerOrder** and select **creditScore**. This enters the expression `CustomerOrder.creditScore` in the **Conditions** column.
5. Again, in the Decision Table, click the **Add** button and from the list select **Condition**.
6. In the Decision Table, in the **Conditions** area double-click the **<edit condition>**. Then, in the navigator expand **CustomerOrder** and select **order**. This enters the expression `CustomerOrder.order`.
7. Again, in the Decision Table, click the **Add** button and from the list select **Condition**.
8. In the Decision Table, double-click **<edit condition>**.
9. In the navigator expand **CustomerOrder** and select **annualSpending**. In the text entry area, add `>2000`.

Figure 1-68 Adding the Annual Spending Entry to a Decision Table



10. Type **Enter** to accept the value. If you view the rules validation log you should see the warning messages. You remove these warning messages as you modify the Decision Table in later steps.

Figure 1-69 Adding Conditions to the CustomerOrder Decision Table

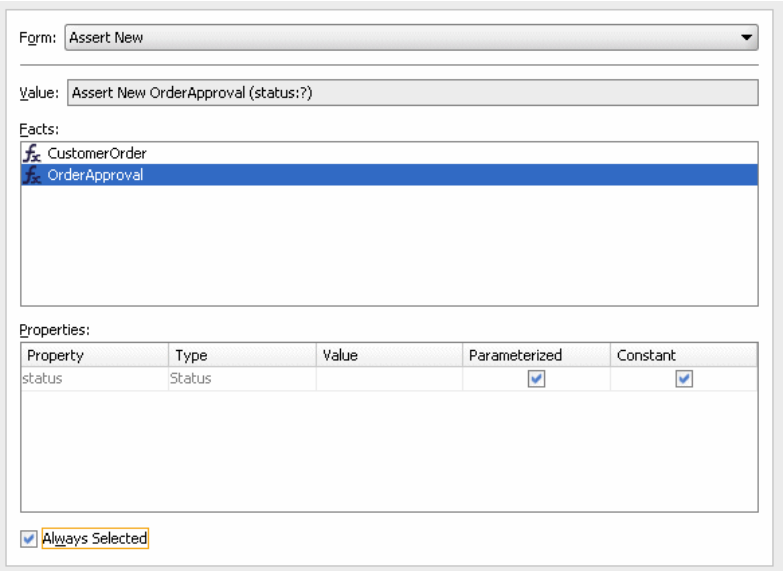


How to Create an action in a Decision Table

To create an action in a Decision Table:

1. In the Decision Table click the **Add** button and from the list select **Action** > **Assert New**.
2. In the **Actions** area, double-click **assert new(**. This displays the Action Editor dialog.
3. In the Action Editor dialog, in the **Facts** area select **OrderApproval**.
4. In the Action Editor dialog, in the Properties table for the property `status` select the **Parameterized** check box and the **Constant** check box. This specifies that each rule independently sets the status.
5. In the Action Editor dialog, select the **Always Selected** check box as shown in Figure 1-70.

Figure 1-70 Adding an Action to a Decision Table with the Action Editor Dialog



6. In the Action Editor dialog, click **OK**.

Next you need to add rules to the Decision Table and specify an action for each rule.

Split the Cells in the Decision Table and Add Actions

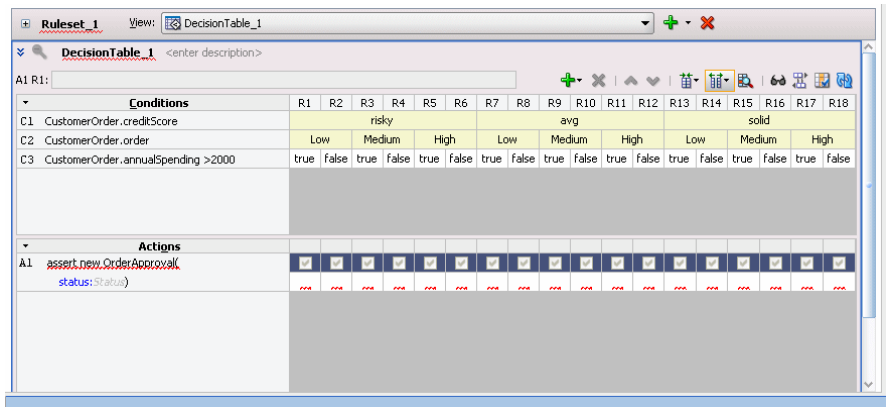
You can use the Decision Table split operation to create rules for the valuesets associated with the condition rows in the Decision Table. This creates one rule for every combination of condition valuesets. There are three order amount valuesets, three credit score valuesets, and two boolean valuesets for the annual spending amount for a total of eighteen rules (3 x 3 x 2 = 18).

To split cells in a decision table:

1. Select the Decision Table.
2. In the Decision Table, click the **Split Table** button and from the list select **Split Table**. The split table operation eliminates the "do not care" cells from the table. The table now shows eighteen rules that cover all ranges as shown in Figure 1-71.

These steps produce validation warnings for action cells with missing expressions. You fix these in later steps.

Figure 1-71 Splitting a Decision Table Using Split Table Operation



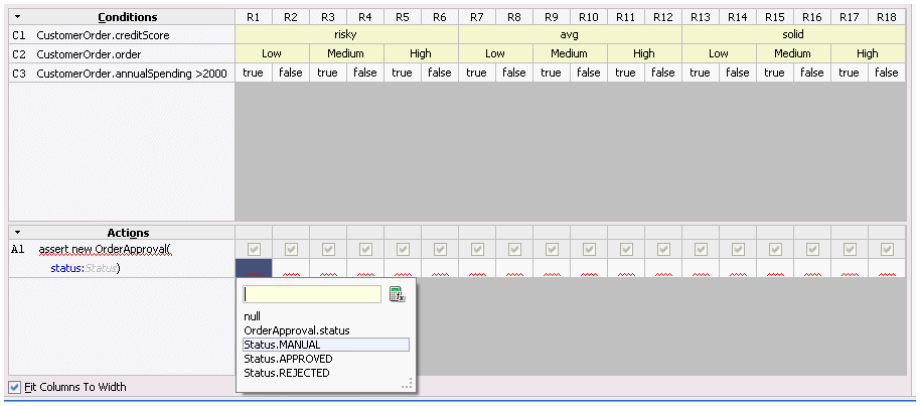
How to Add Actions for Each Rule in the Decision Table

In the Decision Table you specify a value for the status property associated with OrderApproval for each action cell in the **Actions** area. The possible choices are: `Status.MANUAL`, `Status.REJECTED`, or `Status.ACCEPTED`. In this step you fill in a value for status for each of the 18 rules. The values you enter correspond to the conditions that form each rule in the Decision Table.

To add actions for each rule in the decision table:

1. In the **Actions** area, double-click the action cell for the rule you want to work with, as shown in Figure 1-72.

Figure 1-72 Adding Action Cell Values to a Decision Table



2. In the list, select and enter a value for the action cell. For example, enter `Status.MANUAL`.
3. For each action cell, enter the appropriate value as determined by the logic of your application. For this sample application use the values for the Decision Table actions as shown in Table 1-21.
4. Select **Save All** from the **File** main menu to save your work.

Table 1-21 Values for Decision Table Actions

Rule	C1 creditScore	C2 order	C3 annualSpending > 2000	A1 OrderApproval status
R1	risky	Low	true	`Status.MANUAL`
R2	risky	Low	false	`Status.MANUAL`
R3	risky	Medium	true	`Status.MANUAL`
R4	risky	Medium	false	`Status.REJECTED`
R5	risky	High	true	`Status.MANUAL`
R6	risky	High	false	`Status.REJECTED`
R7	avg	Low	true	`Status.APPROVED`
R8	avg	Low	false	`Status.MANUAL`
R9	avg	Medium	true	`Status.APPROVED`
R10	avg	Medium	false	`Status.MANUAL`
R11	avg	High	true	`Status.MANUAL`
R12	avg	High	false	`Status.MANUAL`
R13	solid	Low	true	`Status.APPROVED`
R14	solid	Low	false	`Status.APPROVED`
R15	solid	Medium	true	`Status.APPROVED`
R16	solid	Medium	false	`Status.APPROVED`
R17	solid	High	true	`Status.APPROVED`
R18	solid	High	false	`Status.MANUAL`

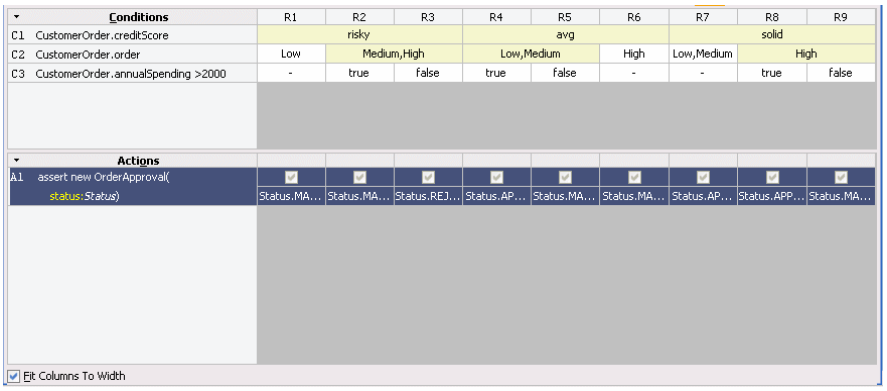
Compact the Decision Table

In this step you compact the rules to merge from eighteen rules to nine rules. This automatically eliminates the rules that are not needed and preserves the no gap, no conflict properties for the Decision Table.

To compact the decision table:

1. Select the Decision Table.
2. Click the **Resize All Columns to Same Width** button.
3. Click the **Compact Table** button and from the list select **Compact Table**. The compact table operation eliminates rules from the Decision Table. The Decision Table now shows nine rules, as shown in Figure 1-73.

Figure 1-73 Compacting a Decision Table Using Compact Table



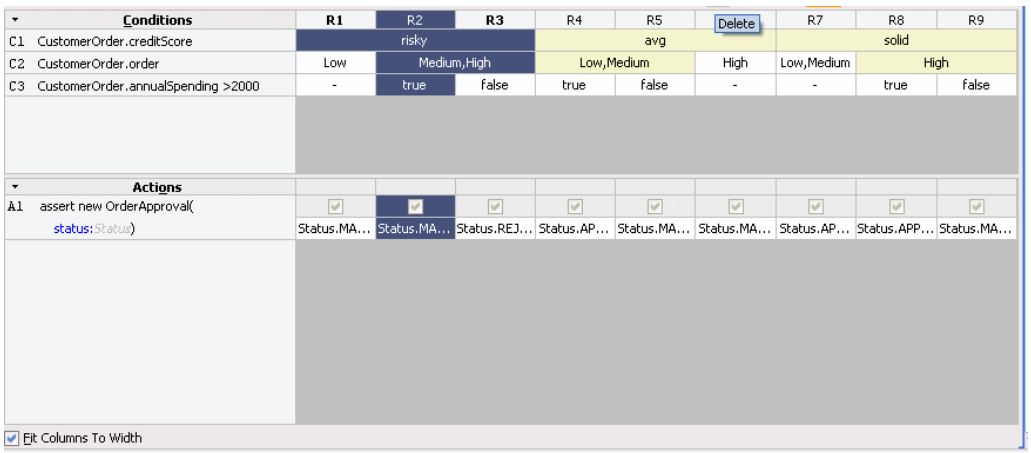
Replace Several Specific Rules with One General Rule

Notice that five of the nine remaining rules result in a manual order approval status. You can reduce the number of rules by deleting these five rules. Note it is often best practice to not do this (that is not replace several specific rules with one general rule). You need to compare the benefits of having fewer rules with the added complexity of managing the conflicts introduced when you reduce the number of rules.

To replace several specific rules with one general rule:

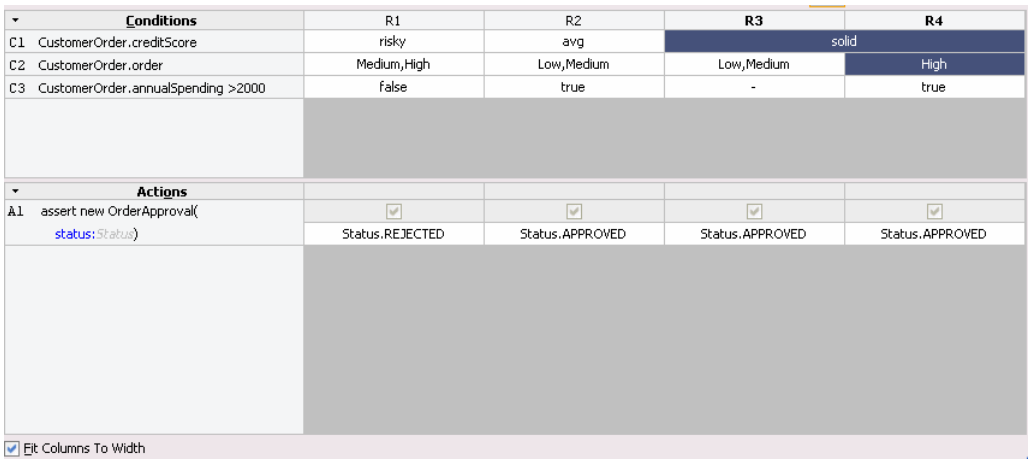
1. Select the Decision Table.
2. In the Decision Table, select a rule with OrderApproval status action set to `Status.MANUAL`. To select a rule, click the column heading. For example, click rule **R2** as shown in Figure 1-74.
3. Click **Delete** to remove a rule in the Decision Table. Be careful to click the delete button in the Decision Table area to delete a rule in the decision table (there is also a delete button shown in the **Ruleset** area that deletes the complete Decision Table).

Figure 1-74 Deleting Rules from a Decision Table



4. Repeat these steps to delete all the rules with action set to `Status.MANUAL`. This should leave the Decision Table with four rules as shown in Figure 1-75.

Figure 1-75 Decision Table After Manual Actions Removed



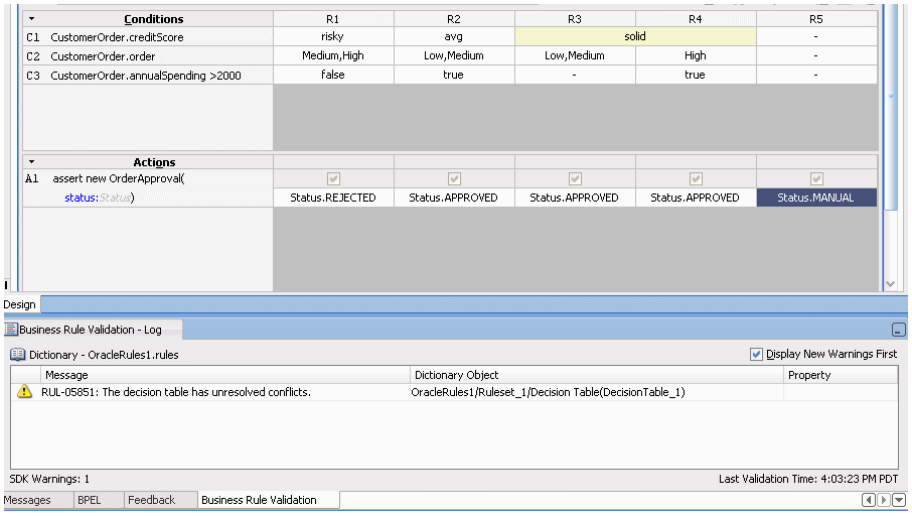
Add a General Rule

Now you can add a single rule to handle the manual case. After adding this rule you set the conflict policy with the option **Conflict Policy auto override** for conflict resolution.

To add a general rule:

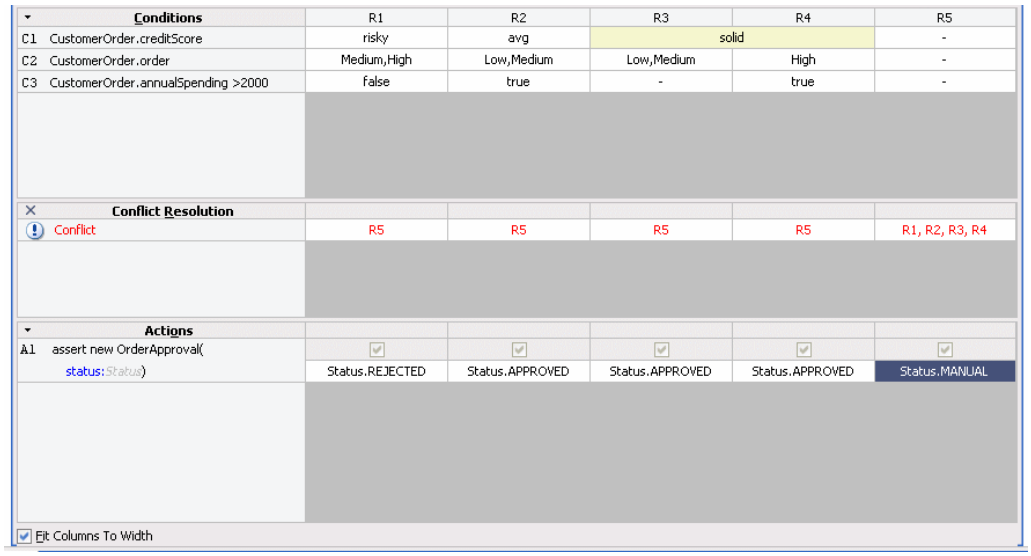
1. In the Decision Table, click the **Add** button and from the list select **Rule**.
2. In the **Conditions** area, for the three conditions leave the "-" do not care value for each cell in the rule.
3. In the **Actions** area, enter `Status.MANUAL`, as shown in Figure 1-76. Notice that the **Business Rule Validation** log includes the warning `RUL-05851` for unresolved conflicts.

Figure 1-76 Decision Table with Conflicting Rules



4. Show the conflicting rules by clicking the **Toggle Display of Conflict Resolution** button, as shown in Figure 1-77.

Figure 1-77 Adding a Rule to Handle Status Manual



Conditions	R1	R2	R3	R4	R5
C1 CustomerOrder.creditScore	risky	avg	solid		-
C2 CustomerOrder.order	Medium,High	Low,Medium	Low,Medium	High	-
C3 CustomerOrder.annualSpending >2000	false	true	-	true	-
Conflict Resolution					
Conflict	R5	R5	R5	R5	R1, R2, R3, R4
Actions					
A1 assert new OrderApproval(	☑	☑	☑	☑	☑
status:Status)	Status.REJECTED	Status.APPROVED	Status.APPROVED	Status.APPROVED	Status.MANUAL

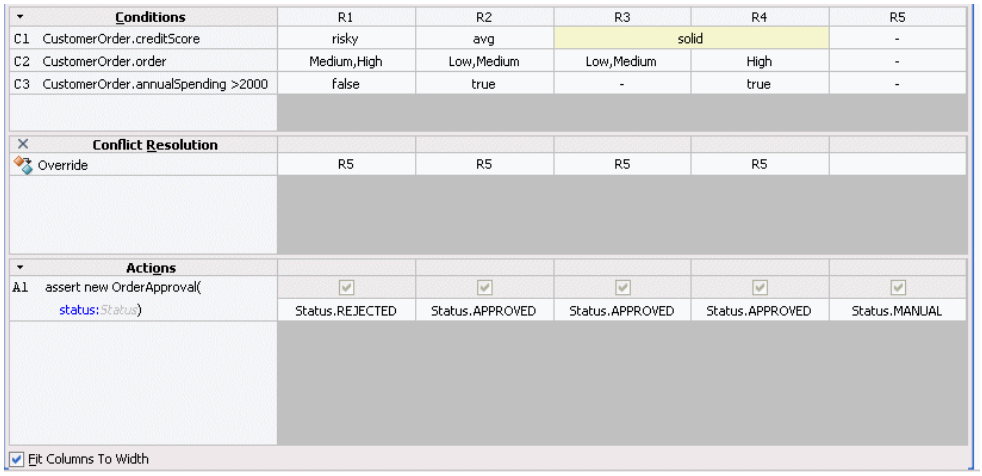
Fit Columns To Width

How to Enable the Auto Override Conflict Resolution Policy

To enable the auto override conflict resolution policy:

1. In the Decision Table click **Show Advanced Settings** (next to the Decision Table name).
2. In the Conflict Policy list, select **auto override**. After adding the manual case rule and selecting **auto override**, notice that the conflicts are resolved and special cases override the general case, as shown in Figure 1-78.

Figure 1-78 Adding a Rule to Handle Status Manual with Auto Override Conflict Policy



Conditions	R1	R2	R3	R4	R5
C1 CustomerOrder.creditScore	risky	avg	solid		-
C2 CustomerOrder.order	Medium,High	Low,Medium	Low,Medium	High	-
C3 CustomerOrder.annualSpending >2000	false	true	-	true	-
Conflict Resolution					
Override	R5	R5	R5	R5	
Actions					
A1 assert new OrderApproval(	☑	☑	☑	☑	☑
status:Status)	Status.REJECTED	Status.APPROVED	Status.APPROVED	Status.APPROVED	Status.MANUAL

Fit Columns To Width

How to Check the Business Rule Validation Log for Order Approval

Before you can deploy the application you need to make sure the dictionary validates without warnings. If there are any validation warnings, you need to fix any associated problems. To validate the dictionary, in the Business Rule Validation Log, check for any

validation warnings. If there are warnings, perform appropriate actions to correct the problems.

How to Deploy the Order Approval Application

Business rules created in a SOA application are deployed as part of the SOA composite when you create a deployment profile in Oracle JDeveloper. You deploy a SOA composite application to Oracle WebLogic Server.

To deploy and run the order approval application:

1. If you have not started your application server instance, then start the Oracle WebLogic Server.
2. In the Application Navigator, right-click the **OrderApproval** project and select **Deploy** > **OrderApproval** > to the appropriate server name.

   Then the SOA Deployment Configuration dialog displays. Select your Application connection which you either have created already or you can create it now. The connection contains the authorization and other connection information (server name, port, etc).
3. Click **Next**.
4. In Select Server select or create and then select your application connection.
5. Click **Next, Next** and **Finish.**

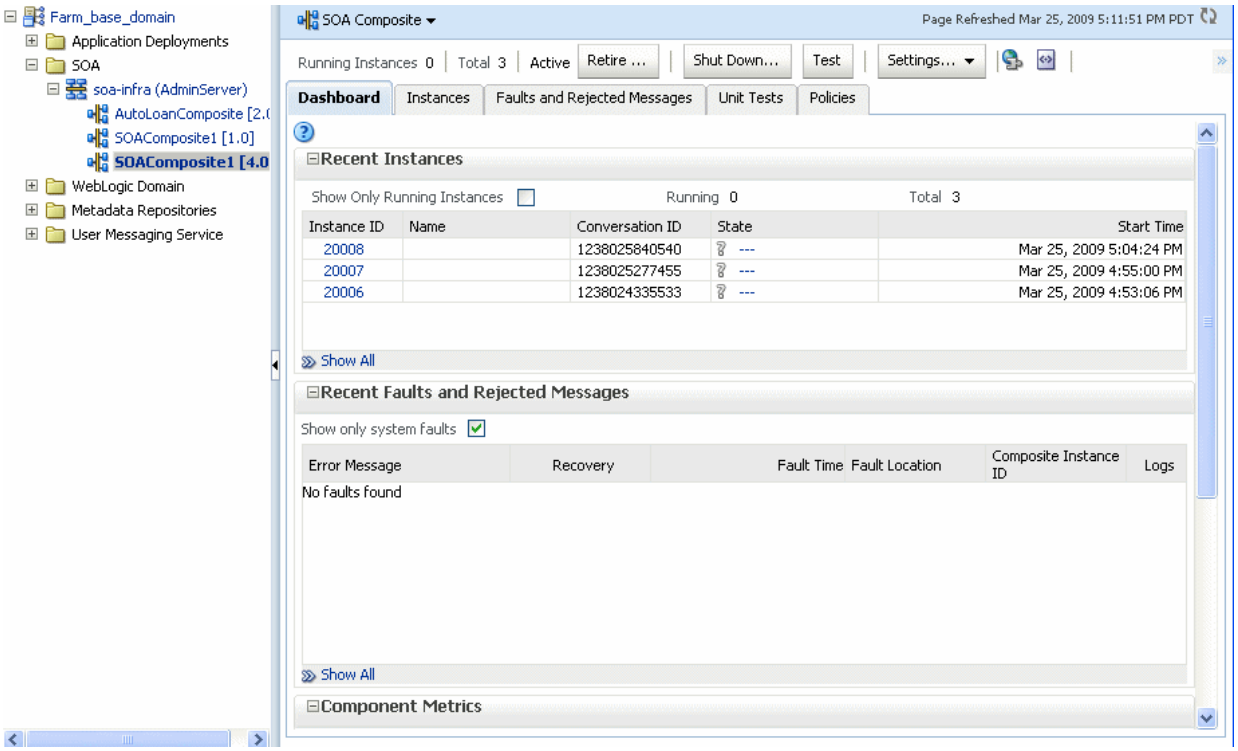
How to Test the Order Approval Application

After deploying the application you can test the Decision Table in the SOA composite application with the Oracle Enterprise Manager Fusion Middleware Control Console.

To test the application:

1. Open the composite application in Oracle Enterprise Manager Fusion Middleware Control Console, as shown in Figure 1-79.

Figure 1-79 Testing the Order Approval Application



2. Click **Test**.
3. In the **Input Arguments** area, select **XML View**. Replace the XML with the contents of the sample input for testing Order Approval application example as shown below.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body xmlns:ns1="http://xmlns.oracle.com/OracleRules1/
OracleRules1_DecisionService_1">
        <ns1:callFunctionStateless name="OracleRules1_DecisionService_1">
                   <ns1:parameterList xmlns:ns3="http://example.com/ns/
customerorder">
                           <ns3:CustomerOrder>
                                   <ns3:name>Gary</ns3:name>
                                       <ns3:creditScore>600</ns3:creditScore>
                                       <ns3:annualSpending>2001.0</
ns3:annualSpending>
                                       <ns3:value>High</ns3:value>
                                       <ns3:order>100.0</ns3:order>
                    </ns3:CustomerOrder>
                   </ns1:parameterList>
           </ns1:callFunctionStateless>
       </soap:Body>
</soap:Envelope>
```

4. Replace the values in the input shown in step 3 as desired for your test.
5. Click **Test Web Service**.
6. In the **Response** tab, view the results. For example, for this input:

```
/OracleRules1_DecisionService_1" xmlns:ns2="http://xmlns.oracle.com/bpel">
      <resultList>
          <OrderApproval:OrderApproval      xmlns:OrderApproval="http://
```

```
example.com/ns/customerorder"
xmlns="http://example.com/ns/customerorder">
              <status>approved</status>
          </OrderApproval:OrderApproval>
       </resultList>
  </callFunctionStatefulDecision>
```

Introduction to Decision Table Operations

After you create a Decision Table you may want to modify the contents of the Decision Table to produce a Decision Table that includes a complete set of rules for all cases, or to produce a Decision Table that provides the least number of rules for the cases.

After you create a Decision Table there are operations that you may want to perform on the Decision Table, including the following:

- Compact or split cells in a Decision Table.
- Merge a condition or split a condition in a Decision Table.
- Finding and resolving conflicts between rules in a Decision Table.
- Find and fix gaps (missing rules) in a Decision Table.

Understanding Decision Table Split and Compact Operations

The split and compact operations enable you to manipulate the contents of the condition cells in a Decision Table.

The split table operation creates a rule for every combination of values across the conditions. For example, in a Decision Table with 3 boolean conditions, 2 x 2 x 2 = 8 rules are created. In a Decision Table with 32 boolean conditions, 2**32 ~ 2 billion rules are created. Thus, you only use split table when the number of rules created is small enough that filling in the action cells is feasible.

When you want to apply match conditions for the "do not care" values in a Decision Table and create a match case for each cell, you use the split table operation.

Split can be applied to an entire Decision Table or to a single condition row. Additionally, split may be performed on an individual condition cell.

Depending on what is selected in the Decision Table, the split operation can create condition cells. Thus, using the split operation you can create rules in a Decision Table. Table 1-22 summarizes the split operation for a selected condition cell, condition row, or for a complete Decision Table.

Table 1-22 Summary of Split Operation

Operator	Description
Condition Cell	Creates one sibling condition cell for each value represented by the cell. If the condition cell value is "do not care", then the cell is split into one sibling cell for each value in the valueset that is not represented by a sibling condition cell, and "do not care" is no longer represented.
Condition Row	For each condition cell in the proceeding condition expression, create a sibling group which contains a cell for each value in the value set. The effect of this operation is the same as adding a "do not care" to each sibling group and calling split on each condition cell in each sibling group.

Table 1-22 (Cont.) Summary of Split Operation

Operator	Description
Decision Table	Same as calling split on each condition row in the Decision Table.

Depending on what is selected in the Decision Table, the compact table or merge cells operations remove condition cells. The compact table operation can be applied to an entire Decision Table. Additionally, the merge operation may be performed on sibling cells or on an entire condition row. Thus, using compact table or merge you can remove rules from a Decision Table. Table 1-23 summarizes the compact table and merge operations.

Table 1-23 Summary of Merge Operation

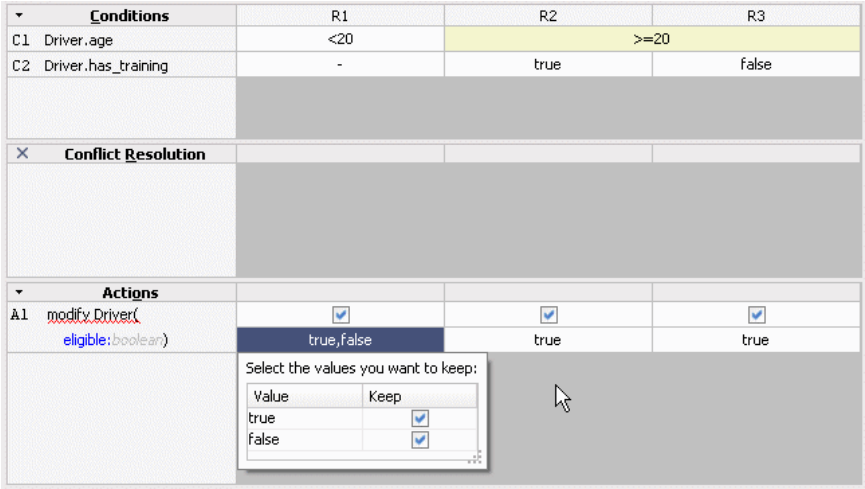
Operator	Description
Condition Cell	Merging two or more condition cells adds all values in the cells to a single cell, and removes all but one of the cells. If one of the cells represents "do not care", then the merged cell represents "do not care". This operation may merge action cells and this can create warnings for duplicate action cells, such as, `RUL-05847: Duplicate decision table action parameter`.
Condition Row	Combine all values in each sibling group into a single "do not care" cell for each condition cell in the proceeding condition expression. The effect of this is the same as calling merge on all cells in each sibling group. This operation may merge action cells and this can create warnings for duplicate action cells, such as, `RUL-05847: Duplicate decision table action parameter`.
Decision Table	Compacts the Decision Table by merging conditions of rules with identical actions.

Split and merge are inverse operations when conflicting action cells are not associated with the operation. In this case, without conflicting action cells, a merge operation combines all the values from the siblings into one sibling, and discards the other sibling condition cells, and as a result of merging the condition cells, when a Decision Table contains action cells, the action cells are also merged. Thus, the merge operation combines multiple condition cells into a single condition cell and adds all values to the single cell.

When there are conflicting values for the action cells, a merge operation merges the action cells in a form that requires additional manual steps. Thus, if two action cells have conflicting parameters, after the merge the action cell contains multiple conflicting parameter values. These conflicting values are appended to the action cell and must be manually resolved by selecting and deleting the unwanted duplicate parameters. For example, see Figure 1-80 that shows conflicting values in an action cell.

An action cell that contains multiple values for a property is invalid. When you select the action cell Rules Designer shows a popup window with check boxes to allow you to select a single value for the action cell. As shown in the validation log in Figure 1-80, Rules Designer shows a validation warning until you select a single value.

Figure 1-80 Conflicting Properties to be Resolved for a Merged Action Cell

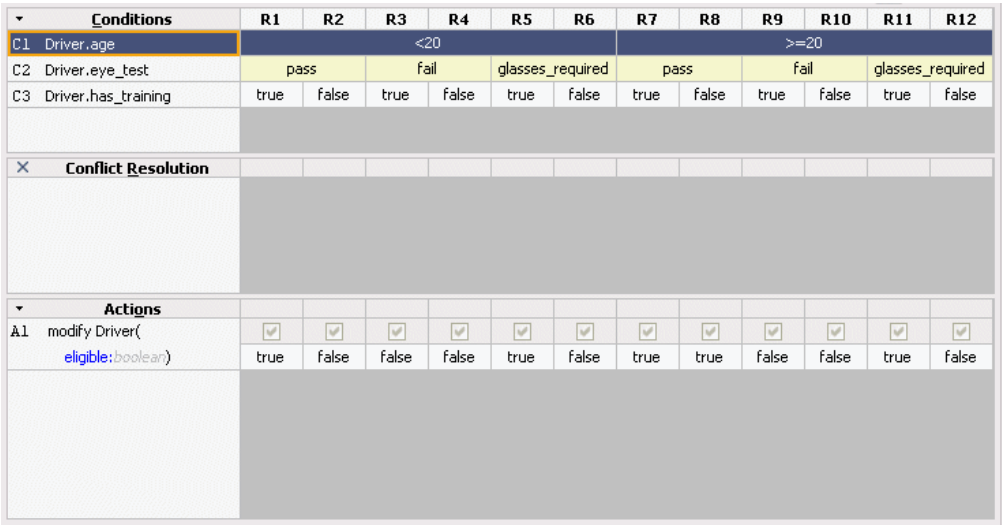


Understanding Decision Table Move Operations

You can move the conditions or actions in a Decision Table. The **Move** buttons let you reorder condition rows in the **Conditions** area and actions in the **Actions** area. Moving conditions up or down may reorder visual display of the rules, but these operations does not change the logic in any way. For example, if (`x.a == 1 and x.b == 1`) is logically the same as if (`x.b == 1 and x.a == 1`).

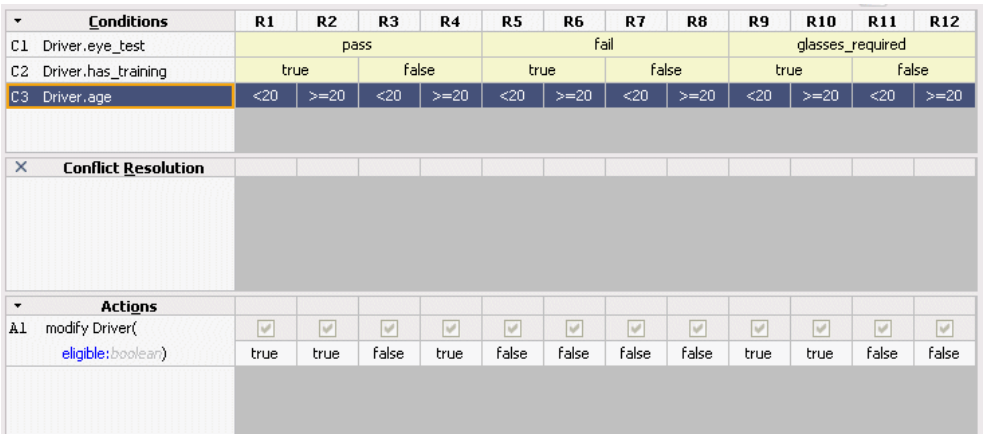
When you work with Decision Tables some operations only apply for condition cells that are siblings. Using the **Move** button you can reorder rows so that Decision Table operations apply to the tree at the desired granularity. For example, when you want to change the action of a condition cell for a single rule, then you need to move that condition cell to the last row in the Decision Table **Conditions** area. For example, consider the Decision Table shown in Figure 1-81.

Figure 1-81 Rules in a Decision Table



To view this table with granularity for the `Driver.age`, move the `Driver.age` condition from the first row to the third row, as shown in Figure 1-82.

Figure 1-82 Decision Table After Move Down with Age Condition Last



Conditions	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12
C1 Driver.eye_test	pass				fail				glasses_required			
C2 Driver.has_training	true		false		true		false		true		false	
C3 Driver.age	<20	>=20	<20	>=20	<20	>=20	<20	>=20	<20	>=20	<20	>=20
Conflict Resolution												
Actions												
A1 modify Driver(	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
eligible:boolean)	true	true	false	true	false	false	false	false	true	true	false	false

Now to make the `Driver.age` conditions "do not care" for the first two rules, where the driver passes the eyesight test and has had driver training is true, you can easily apply changes to these particular conditions when the `Driver.age` condition is in the last row. Thus, in this table, it is easier to view and modify age related rules when `Driver.age` is in the last row, with the finest granularity.In general, the move operations can assist you when you want to split, merge, or assign certain values that might only be appropriate at a particular level in the tree, depending on the location of a condition cell, or depending on the location of the parent, children, or siblings of a condition cell.

For actions in the **Actions** area, clicking **Move Up** or **Move Down** lets you reorder the actions. Actions are ordered so that when multiple actions apply, the first action runs before subsequent actions. Thus, using the **Move Up** or **Move Down** operation on an action may be appropriate, depending on your application.

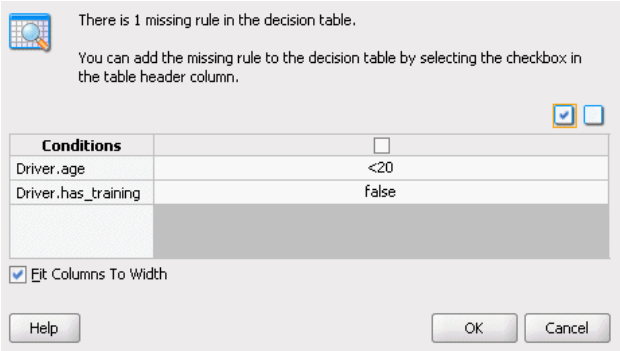
Understanding Decision Table Gap Checking

A gap is a "missing" rule in a Decision Table. A Decision Table has a gap if there is a combination of values, one from each condition, that is not covered by an existing rule. Rules Designer provides Gap Checking to check for gaps. When you click the **Gap Analysis** button, Rules Designer finds gaps and presents a dialog to fix any gaps that are found.

You can choose to make existence of gaps a validation warning. When you clear **Allow Gaps** in the **Advanced Settings** area, the Decision Table reports a validation warning when a gap is found. For more information, see *Using Advanced Settings with Rules and Decision Tables* in *Designing Business Rules with Oracle Business Process Management*.

For example, using the Driver example if you create a gap by deleting the rule that covers the case for `Driver.age` < 20 and `Driver.has_training false`, and then you click **Gap Analysis**, Rules Designer shows the Gap Analysis dialog as shown in Figure 1-83. Clicking **OK** with the check boxes selected adds either all rules or the selected rules to the Decision Table (this example only shows a single rule to add).

Figure 1-83 Checking Gaps



Gap checking generates different new rules for the following cases:

- Sibling rules: multiple missing sibling rules are added as a single new rule. For example, consider a rule with two conditions, `Driver.age` and `Driver.hair`. When there are two missing rules for different hair colors and the rules are siblings, that is, they have a common parent, then gap checking shows a single rule as shown in Figure 1-84.
- Non-sibling rules: multiple missing non-sibling rules are added as individual new rules. For example, when there are two different rules missing that do not have the same parent, then gap checking provides two rules, as shown in Figure 1-85.

Figure 1-84 Gap Checking with Missing Sibling Rules

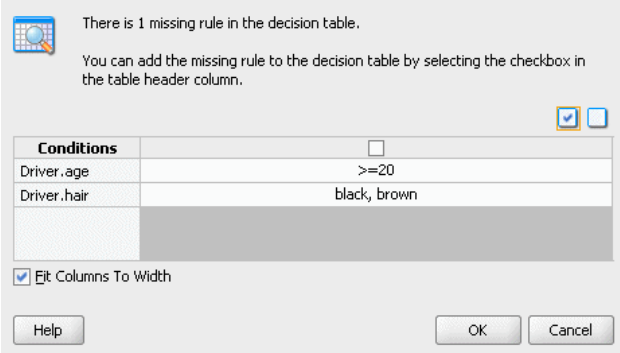
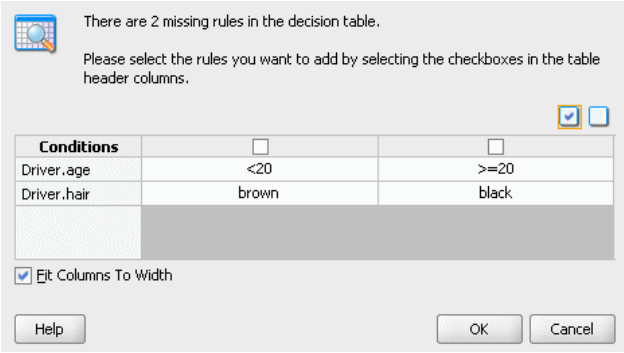


Figure 1-85 Gap Checking with Missing Non-Sibling Rules



In both of these cases shown in Figure 1-84 and Figure 1-85 there are two missing values, but for sibling rules the multiple values are combined in a single new rule. Thus, in general gap checking suggests fewer more general rules in preference to many more specific rules.

For sibling rules you can add multiple rules then edit each cell to pick the values you want. Alternatively, you can use **Find Gaps** to add a rule and then split the cell with multiple values, and delete the rules you do not want to keep.

Understanding Decision Table Conflict Analysis

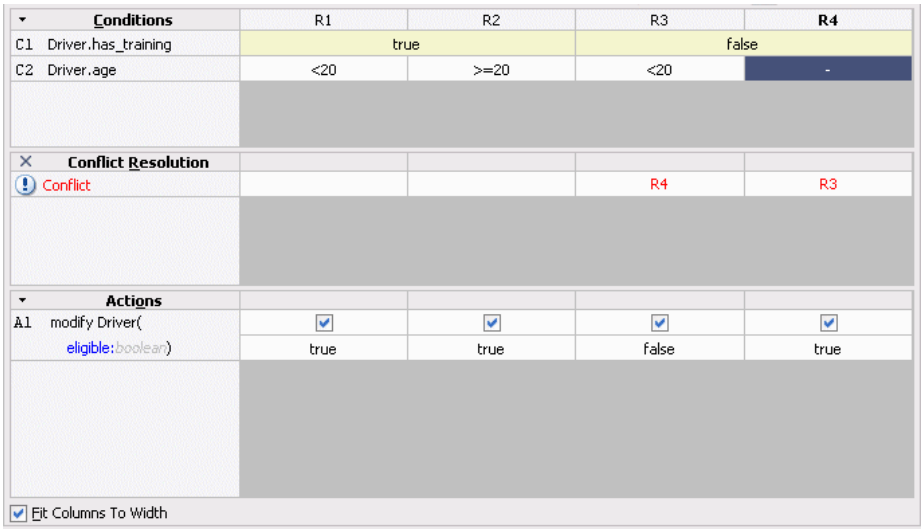
The rules in a Decision Table can conflict. Two rules conflict when they overlap and they have different actions. Two rules overlap when at least one of their condition cells has a value in common. Overlap is common when a Decision Table contains "do not care" condition cells. Overlap without conflict is common and harmless.

Rules Designer finds conflicts and you can see the conflicts in the **Conflict Resolution** row in the Decision Table when you click **Show Conflicts**. How you handle and resolve conflicts depends on the specified conflict policy. You can choose a conflict policy or use the default manual conflict policy. When you set a conflict policy using the **Conflict Policy** option in the **Advanced Settings** area, Rules Designer sets the conflict policy for the Decision Table. The **Conflict Policy** specifies the Decision Table conflict policy and is one of the following:

- **manual**: Conflicts are resolved by manually specifying a conflict resolution for each conflicting rule.
- **auto override**: Conflicts are resolved automatically using an override conflict resolution when this is possible, using the Oracle Business Rules automatic conflict resolution policies.
- **ignore**: Conflicts are ignored.

For more information, see *Using Advanced Settings with Rules and Decision Tables* in *Designing Business Rules with Oracle Business Process Management*. For example, Figure 1-86 shows a Decision Table with conflicting rules that you resolve with the default manual conflict policy.

Figure 1-86 Decision Table Showing Conflicting Rules in the Conflicts Area



By clicking on the cells in the Decision Table **Conflict Resolution** area Rules Designer lets you resolve conflicts between rules as follows:

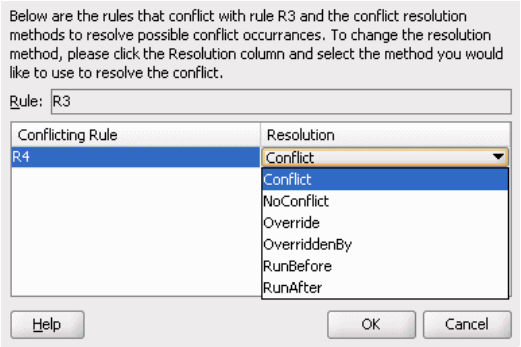
- Override (**Override** and **OverriddenBy**): You override one rule with the other. Override specifies that one rule fires. Override is a combination of prioritization and mutual exclusion. Prioritization is transitive and not symmetric. Mutual exclusion is both transitive and symmetric. If A overrides C and B overrides C, then A or B runs before C but only one of A, B, or C runs.
- Run Before (**RunBefore** and **RunAfter**): You prioritize the rules. Run before lets the two rules fire in a prescribed order. Prioritization is transitive but not symmetric. That is, if A runs before B runs before C, then A runs before C but B does not run before A. This uses a Decision Table runBefore list specifying that the rule that runs before has a higher priority than rules in the list.
- Ignore (**NoConflict**): You OK the conflict. Ignore lets the two rules fire in arbitrary order. For example, consider the following conflicting rules in a decision table:

  ```
  rule1: everybody gets a 10% raise (as specified with a do not care value in a decision table
  condition cell)
  rule2: employee with Top Performer set to true gets a 5% raise
  ```

  In these rules, if rule2 overrides rule1, then a top performer gets a 5% raise, and everyone else gets a 10% raise. However, in this case, you would like to have both rules fire. Because it does not matter which rule fires first, and there is no conflict, then a top performer gets a 15.5% raise either way. In this case, use the NoConflict list to remove the conflict. Note that no conflict is what you get with IF/THEN rules with equal priorities, only you are not warned of a conflict and you have to think carefully if you want one rule to override the other.

Figure 1-87 shows the Rules Designer Conflict Resolution dialog shown when you select a conflicting rule in the **Conflict Resolution** area. This dialog lets you resolve conflicts between rules by selecting overrides, prioritization with RunBefore or RunAfter options, and a NoConflict option.

Figure 1-87 Using the Decision Table Conflict Resolution Dialog



You can use the Decision Table Advanced Settings **Conflict Policy auto override** option to specify that, where possible, conflicts are automatically resolved. The automatic override conflict resolution policy specifies that a special case overrides a more general case.

Thus, when there are conflicts in a Decision Table, you can do one or more of the following to resolve the conflicts:

- Use auto override conflict resolution by selecting the **Conflict Policy** and then **auto override** option in the Decision Table.
- Ignore conflicts by selecting the **Conflict Policy** and then **ignore** option in the Decision Table.
- Use manual conflict resolution by selecting the **Conflict Policy** and then **manual** option in the Decision Table and set Conflict Resolution for each conflicting rule in the dialog by selecting cells in the **Conflict Resolution** area with the **Show Conflicts** check box selected.
- Change the Decision Table to remove an overlap.
- Combine actions to remove conflicts.

How to Compact or Split a Decision Table

Use the **Compact Table** and **Split Table** buttons to compact or split cells in a Decision Table. For more information, see Understanding Decision Table Split and Compact Operations.

To compact or split cells in a decision table:

1. In Rules Designer, select a rule set from the **Rule Sets** navigation tab. On the **Overview** tab, select the Decision Table and click **Edit.**
2. Click the **Compact Table** button to compact or the **Split Table** button to split the cells.

How to Merge or Split Conditions in a Decision Table

Use the merge condition and split condition operations to merge or split a condition in a Decision Table. For more information, see Understanding Decision Table Split and Compact Operations.

To merge or split a condition in a decision table:

1. In Rules Designer, select a rule set from the **Rule Sets** navigation tab. On the **Overview** tab, select the Decision Table where you want to merge or split a condition and click **Edit.**
2. In the **Conditions** area, select the condition you want to merge or split.
3. Right-click, and from the list select **Merge Condition** or **Split Condition**.

How to Use the Condition Cell Operations

Use the condition cell operations to split a condition cell, to merge sibling condition cells, or to specify a "do not care" value for a condition cell in a Decision Table. For more information, see Understanding Decision Table Split and Compact Operations.

How to Merge Sibling Cells in a Condition in a Decision Table

1. In Rules Designer, select a rule set from the **Rule Sets** navigation tab. On the **Overview** tab, and select the Decision Table where you want to merge condition cells and click **Edit.**
2. Select the sibling condition cells to merge.

3. Right-click, and from the list select **Merge selected cells**.

How to Split a Cell in a Condition in a Decision Table

1. In Rules Designer, select a rule set from the **Rule Sets** navigation tab. On the **Overview** tab, and select the Decision Table where you want to split a condition cell and click **Edit.**
2. Select the cell to split.
3. Right-click, and from the list select **Split selected cell**.

How to a "Do Not Care" Value for a Cell in a Condition in a Decision Table

1. From Rules Designer select a ruleset from the **Rulesets** navigation tab and select the Decision Table where you want to set the "do not care" value.
2. Select the appropriate condition cell.
3. Right-click, and from the list select **Do Not Care**.

How to Select all Value Sets to Specify a "Do Not Care" Value for a Cell in a Condition:

1. In Rules Designer, select a rule set from the **Rule Sets** navigation tab. On the **Overview** tab, and select the Decision Table where you want to set the "do not care" value and click **Edit.**
2. Select the appropriate condition cell.
3. Double-click, and from the list select all the available check boxes for all possible values.

How to Perform Decision Table Gap Checking

A gap is a "missing" rule in a Decision Table. A Decision Table has a gap if there is a combination of values, one from each condition, that is not covered by an existing rule. Rules Designer provides Gap Checking to check for gaps. When you use this operation Rules Designer presents a window to fix gaps. For more information, see Understanding Decision Table Gap Checking.

You can choose to make existence of gaps a validation warning. When you clear **Allow Gaps** in the **Advanced Settings** area, the Decision Table reports a validation warning when a gap is found. For more information, see *Using Advanced Settings with Rules and Decision Tables* in *Designing Business Rules with Oracle Business Process Management*.

To perform decision table gap checking:

1. In Rules Designer, select a rule set from the **Rule Sets** navigation tab. On the **Overview** tab, and select the Decision Table where you want to perform gap checking and click **Edit.**
2. Click the **Gap Analysis** button.

How to Perform Decision Table Manual Conflict Resolution

The rules in a Decision Table can conflict. Two rules conflict when they overlap and they have different actions. Two rules overlap when at least one of their condition cells has a value in common. For more information, see Understanding Decision Table Conflict Analysis.

To perform manual decision table conflict resolution:

1. In Rules Designer, select a rule set from the **Rule Sets** navigation tab. On the **Overview** tab, and select the Decision Table where you want to check conflicts and click **Edit.**
2. Set the conflict policy to **manual** (this is the default conflict policy). For more information, see Understanding Decision Table Conflict Analysis.
3. In the **Conditions** area, in the conflicts area, when conflicts exist for each rule with a conflict double-click the appropriate condition cell to display the Conflict Resolution dialog.
4. In the Conflict Resolution dialog, for each conflicting rule, in the Resolution field select a resolution from the list.

How to Set the Decision Table Auto Override Conflict Resolution Policy

When you select the Advanced Settings option in a Decision Table, you can select that Decision Table conflicts are automatically resolved using the **auto override** conflict policy (this applies only when it is possible to resolve the conflict using the Oracle Business Rules automatic conflict resolution policies). The automatic override conflict resolution uses a policy where when there is a rule conflict a special case overrides a more general case. For more information, see Understanding Decision Table Conflict Analysis.

To select the auto override policy:

1. Select the rule or Decision Table where you want to use ignore conflict policy.
2. Click the **Show Advanced Settings** button next to the rule or Decision Table name.
3. From the **Conflict Policy** option select **auto override**.

How to Set the Decision Table Ignore Conflicts Policy

When you select the Advanced Settings option in a Decision Table, you can select that the Decision Table conflicts are ignored using the **ignore** conflict policy. The ignore policy tells Oracle Business Rules to ignore conflicts in the Decision Table. For more information, see Understanding Decision Table Conflict Analysis.

To select the ignore conflict policy:

1. Select the rule or Decision Table where you want to use the ignore conflicts policy.
2. Click the **Show Advanced Settings** button next to the rule or Decision Table name.
3. From the **Conflict Policy** option select **ignore**.

Creating and Running an Oracle Business Rules Decision Table Application

The Order Approval application demonstrates the integration of a SOA composite application with Oracle Business Rules and the use of a Decision Table.

In this application a process is modeled that uses the decision component to:

- Process rules from XML inputs including: a credit score and the annual spending of a customer, and the total cost of the incoming order.
- Provide output that determines if an order is approved, rejected, or requires manual processing.

To complete this procedure, you need to:

- Obtain the Source Files for the Order Approval Application
- Create an Application for Order Approval
- Create a Business Rule Service Component for Order Approval
- View Data Model Elements for Order Approval
- Add Value Sets to the Data Model for Order Approval
- Associate Value Sets with Order and CreditScore Properties
- Add a Decision Table for Order Approval
  - Split the Cells in the Decision Table and Add Actions
  - Compact the Decision Table
  - Replace Several Specific Rules with One General Rule
  - Add a General Rule
- Check Dictionary Business Rule Validation Log for Order Approval
- Deploy the Order Approval Application
- Test the Order Approval Application

How to Obtain the Source Files for the Order Approval Application

The source code for Oracle Business Rules-specific samples and SOA samples are available online in the Oracle SOA Suite Samples and Tutorials page.

To work with the Order Approval application, you first need to obtain the `order.xsd` schema file either from the sample project that you obtain online or you can create the schema file and create all the application, project, and other files in Oracle JDeveloper. You can save the schema file provided in the following example locally to make it available to Oracle JDeveloper.

The following example shows the `order.xsd` schema file.

```
<?xml version="1.0" ?>
<schema attributeFormDefault="qualified" elementFormDefault="qualified"
        targetNamespace="http://example.com/ns/customerorder"
        xmlns:tns="http://example.com/ns/customerorder"
        xmlns="http://www.w3.org/2001/XMLSchema">
  <element name="CustomerOrder">
    <complexType>
      <sequence>
        <element name="name" type="string" />
        <element name="creditScore" type="int" />
        <element name="annualSpending" type="double" />
        <element name="value" type="string" />
        <element name="order" type="double" />
      </sequence>
    </complexType>
  </element>
```

```
  <element name="OrderApproval">
    <complexType>
      <sequence>
        <element name="status" type="tns:Status"/>
      </sequence>
    </complexType>
  </element>
  <simpleType name="Status">
    <restriction base="string">
        <enumeration value="manual"/>
        <enumeration value="approved"/>
        <enumeration value="rejected"/>
    </restriction>
  </simpleType>
</schema>
```

How to Create an Application for Order Approval

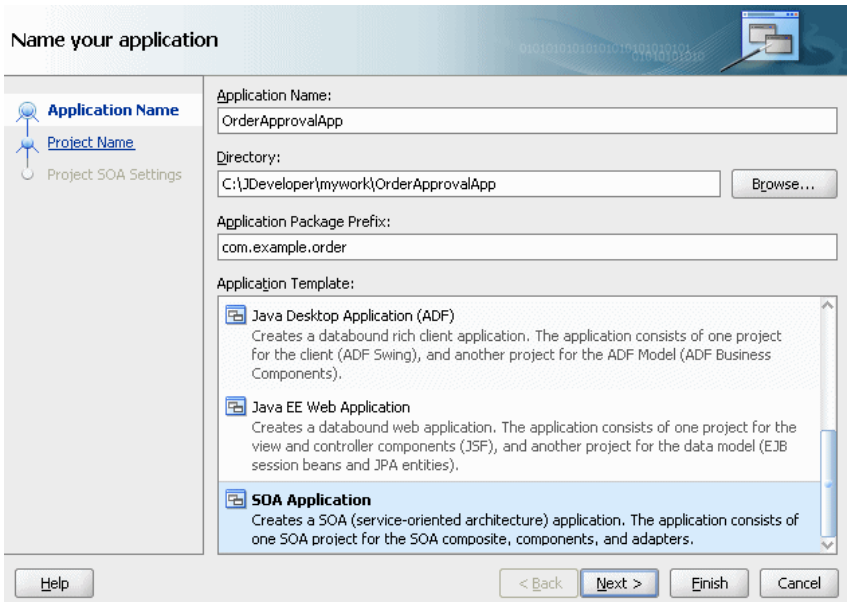
To work with Oracle Business Rules, you first create an application in Oracle JDeveloper.

To create an application for order approval:

1. In the Application Navigator, click **New Application**.
2. In the Name your application dialog, enter the name and location for the new application.
   a. In the **Application Name** field, enter an application name. For example, enter `OrderApprovalApp`.
   b. In the **Directory** field, specify a directory name or accept the default.
   c. In the **Application Package Prefix** field, enter an application package prefix, for example `com.example.order`.

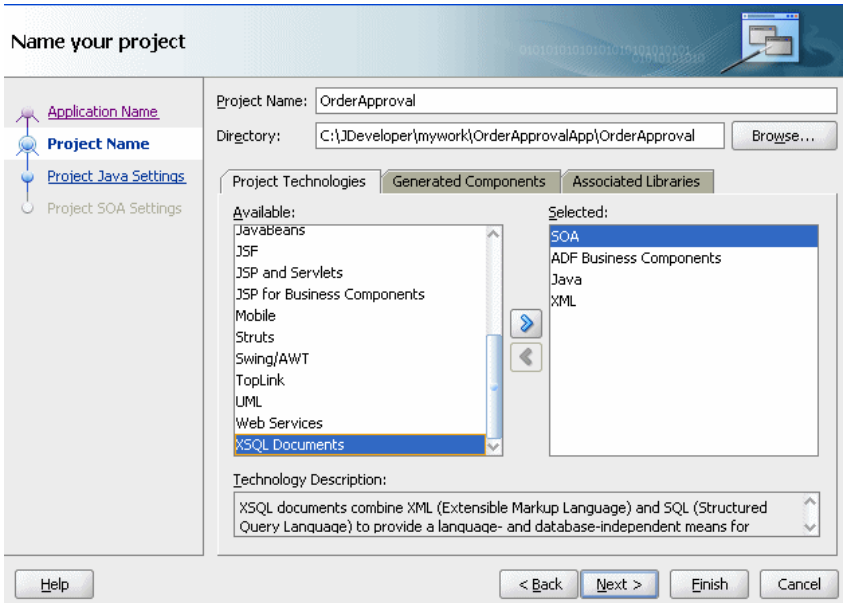
      The prefix, followed by a period, applies to objects created in the initial project of an application.
   d. For a SOA composite with Oracle Business Rules, in the Application Template area select SOA Application for the application template. For example, see Figure 1-57.
   e. Click **Next**.

Figure 1-88 Adding the Order Approval Application



3. In the Name your project page enter the name and location for the project.
   a. In the **Project Name** field, enter a name. For example, enter `OrderApproval`.
   b. Enter or browse for a directory name, or accept the default.
   c. For an Oracle Business Rules project, in the **Project Technologies** area ensure that SOA, ADF Business Components, Java, and XML are in the **Selected** area on the Project Technologies tab, as shown in Figure 1-58. If an item is missing, select it in the **Available** pane and add it to the **Selected** pane using the **Add** button.

Figure 1-89 Adding a Project to an Application



4. Click **Finish**.

How to Create a Business Rule Service Component for Order Approval

After creating a project in Oracle JDeveloper you need to create a Business Rule Service component within the project. When you add a business rule you can create input and output variables to provide input to the service component and to obtain results from the service component.

To use business rules with Oracle JDeveloper, you do the following:

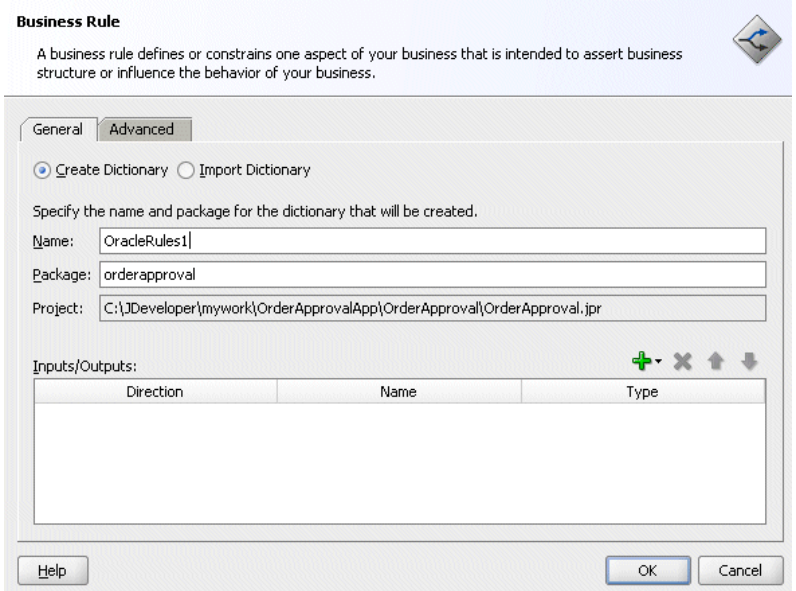
- Add a business rules service component
- Create input and output variables for the service component
- Create an Oracle Business Rules dictionary in the project

To create a business rule service component:

1. In the Application Navigator, in the **OrderApproval** project expand **SOA Content** and double-click `composite.xml` to launch the SOA composite editor (this may already be open after you create the project).
2. From the Component Palette, drag-and-drop a **Business Rule** from the **Service Components** area of the SOA menu to the **Components** lane of the `composite.xml` editor.

   Oracle JDeveloper displays a Create Business Rules page, as shown in Figure 1-59.

Figure 1-90 Adding a Business Rule Dictionary with the Create Business Rules Dialog



3. To add an input, from the list next to the **Add** button select **Input** to enter input for the business rule.
4. In the Type Chooser dialog, click the **Import Schema File**... button. This displays the Import Schema File dialog.

5. In the Import Schema dialog click **Browse Resources** to choose the XML schema elements for the input variable of the process. This displays the SOA Resource Lookup dialog.
6. In the SOA Resource Lookup dialog, navigate to find the `order.xsd` schema file and click **OK**.
7. In the Import Schema File dialog, make sure **Copy to Project** is selected, as shown in Figure 1-60. Click **OK**.

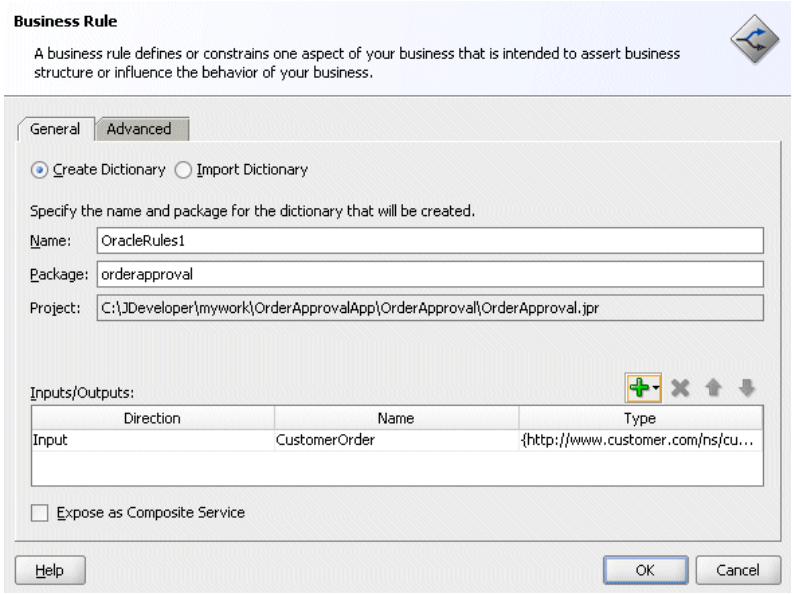
Figure 1-91 Importing the Order.xsd Schema File

8. If the Localize Files dialog displays, click **OK** to copy the schema to the composite process directory.
9. In the Type Chooser, navigate to the Project Schemas Files folder to select the input variable.

   For this example, select `CustomerOrder` as the input variable.

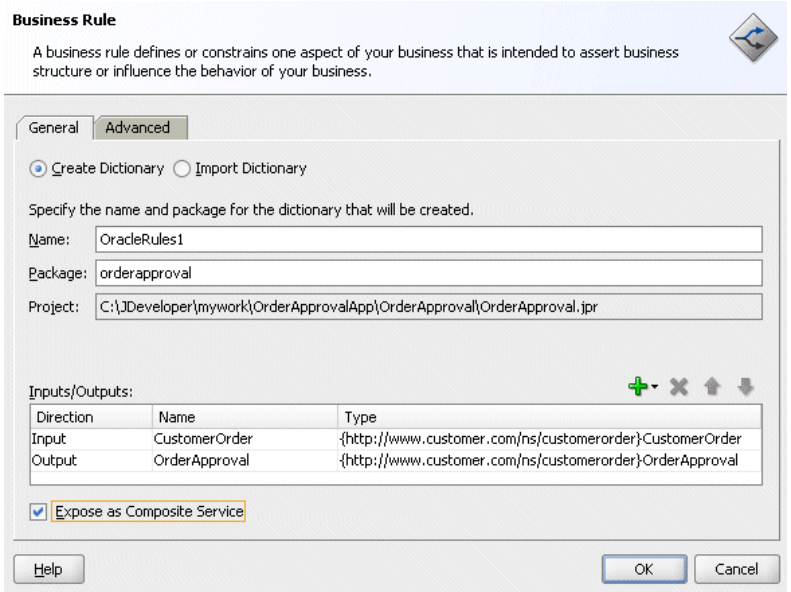
   On the Type Chooser window, click **OK**. This displays the Create Business Rules dialog, as shown in Figure 1-61.

Figure 1-92 Create Business Rules Dialog with CustomerOrder Input



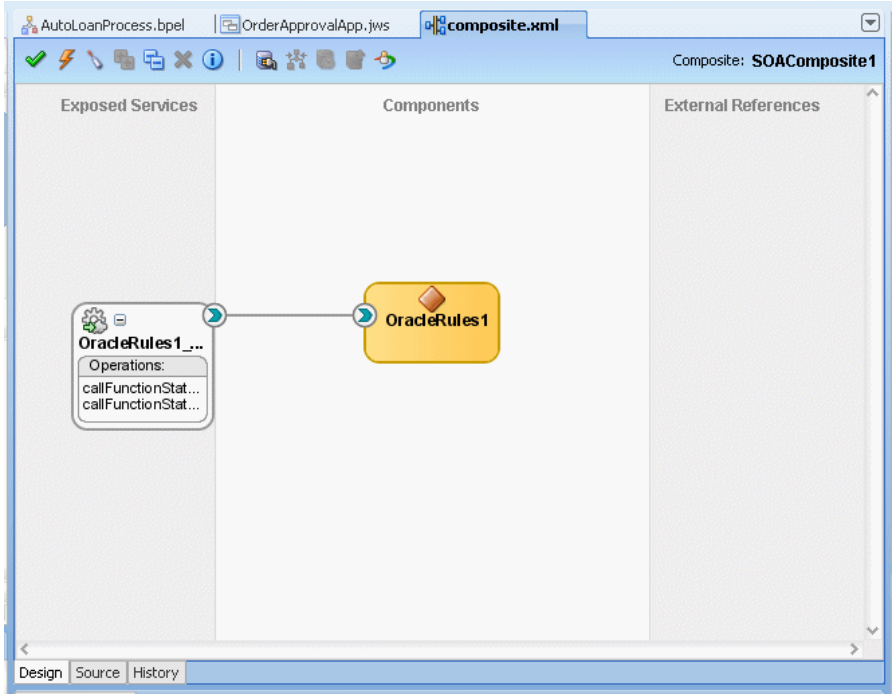
10. In a similar manner, add the output fact type `OrderApproval` from the imported `order.xsd`.
11. In the Create Business Rules dialog, select **Expose as Composite Service**, as shown in Figure 1-62.

Figure 1-93 Create Business Rules Dialog with Input and OrderApproval Output



Click **OK**. This creates the Business Rule component and Oracle JDeveloper shows the Business Rule in the canvas workspace, as shown in Figure 1-63.

Figure 1-94 Business Rules Component in OrderApproval Composite



The business rule service component enables you to integrate your SOA composite application with a business rule. This creates a business rule dictionary and enables you to execute business rules and make business decisions based on the rules.

How to View Data Model Elements for Order Approval

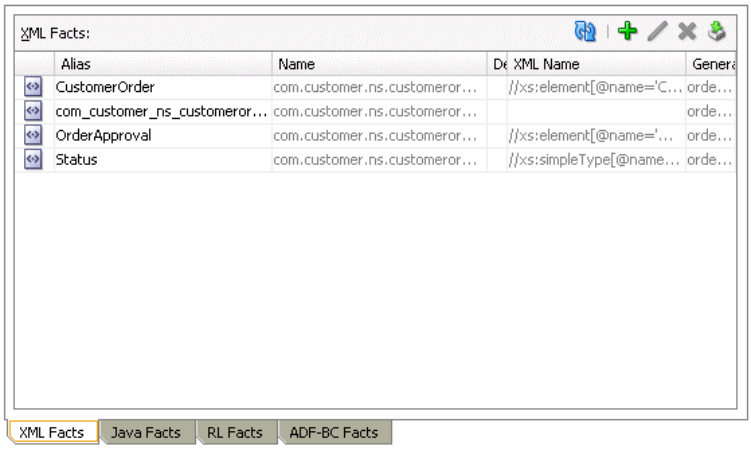
Before adding rules you need to create the Oracle Business Rules data model. The data model contains the business data definitions (types) and definitions for facts that you use to create rules. For example, for this sample the data model includes the XML schema elements from `order.xsd` that you specify when you define inputs and outputs for the business rule activity.

At times when you work with Rules Designer to create a rule or a Decision Table, you may need to create or modify elements in the data model.

To view data model elements for Oracle business rules:

1. Select the composite tab with the value **composite.xml**, and in the Components lane select the business rule (this surrounds the component, **OracleRules1** with a dashed selection box).
2. Double-click the selection box to launch Rules Designer.
3. In Rules Designer select the **Facts** navigation tab.
4. Select **XML Facts** tab in the **Facts** navigation tab as shown in Figure 1-64.

Figure 1-95 Opening a Business Rules Dictionary with Rules Designer



How to Add Value Sets to the Data Model for Order Approval

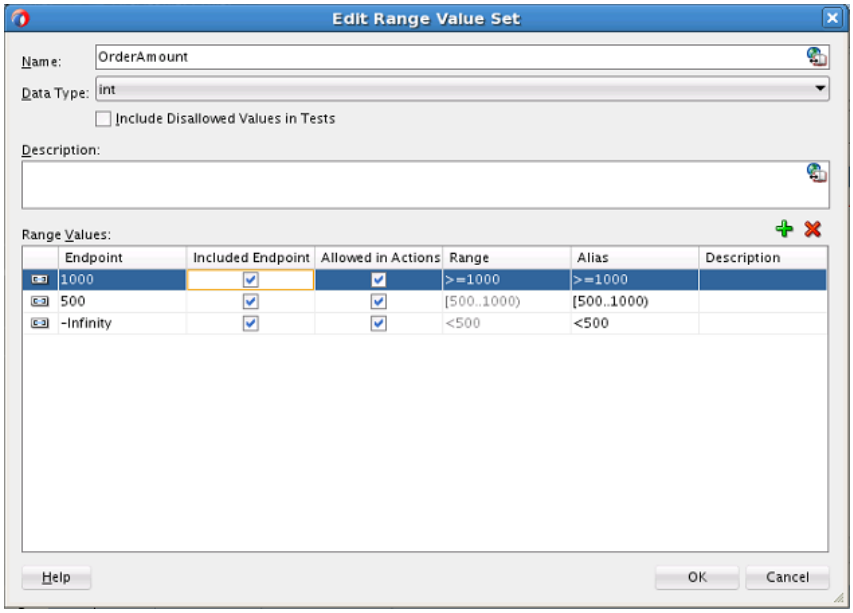
To use a Decision Table you need to define value sets that specify how to draw values for each cell for the conditions that constitute the Decision Table. For this example the value sets are defined with a list of ranges that you define in Rules Designer.

To add OrderAmount value set to the data model:

1. In Rules Designer, select the **Value Sets** navigation tab.
2. From the drop down next to the **Create Value Set...** button, select **Range Value Set**.
3. In the **Name** field, enter `OrderAmount`. Press **Enter** to accept the name.
4. Double-click the **OrderAmount** value set icon to display the **Edit Range Value Set** dialog.

5. Click **Add Value** to add a value.
6. Click **Add Value** again to add another value.
7. In the **Range Values** area, in the top **Endpoint** field enter 1000 for the endpoint value.
8. In the **Range Values** area, for the middle bucket in the **Endpoint** field enter 500 for the endpoint value.
9. In the **Included Endpoint** field for each value set ensure the check box is selected, as shown in Figure 1-65.

Figure 1-96 Adding the OrderAmount Value Set



10. Modify the **Alias** field for each value to **High, Medium, and Low**. Click **OK**.

How to Add CreditScore Value Set to the Data Model

To add CreditScore value set:

1. In Rules Designer select the **Value Sets** navigation tab.
2. From the drop down next to the **Create Valueset...** button, select **List of Ranges**.
3. In the **Name** field, enter `CreditScore`.
4. Double-click the **CreditScore** valueset icon to display the Edit Valueset dialog.
5. Click **Add Value** to add a value.
6. Click **Add Value** again to add another value.
7. In the top valueset, in the **Endpoint** field enter `750`.
8. For the middle valueset, in the **Endpoint** field enter `400`.
9. In the **Included Endpoint** field for each valueset, ensure the check box is selected.
10. Modify the **Alias** field for each endpoint value to **solid** for 750, **avg** for 400, and **risky** for -Infinity. Click **OK**.

How to Associate Value Sets with Order and CreditScore Properties

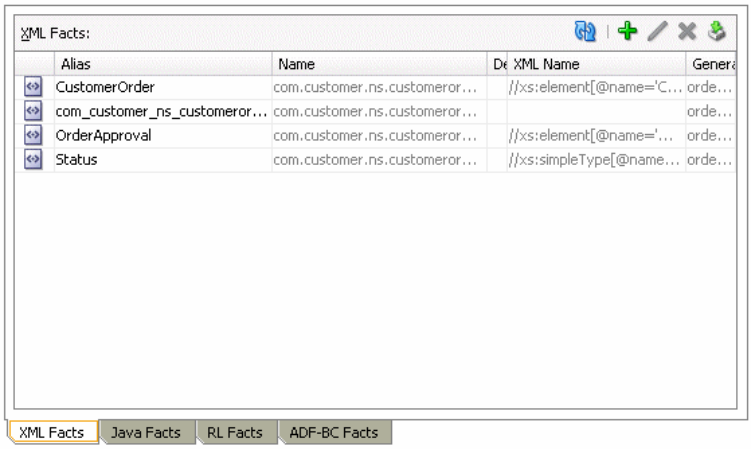
To prepare for creating Decision Tables you can associate a value set with fact properties in the data model. In this way condition cells in a Decision Table **Conditions** area can use the valuesets when you create a Decision Table.

Note that the `OrderApproval.status` property is associated with the `Status` value set when the `OrderApproval` fact type is imported from the XML schema. In the schema, `Status` is a restricted `String` type and is therefore represented as an enum valueset. Rules Designer creates the status value set.

To associate value sets with Order and CreditScore properties:

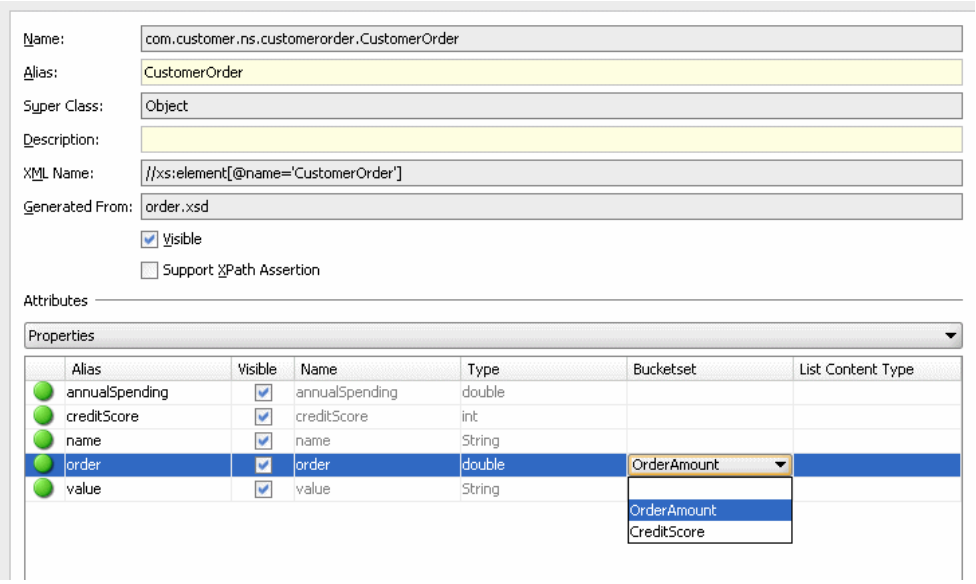
1. In Rules Designer select the **Facts** navigation tab.
2. Select the **XML Facts** tab in the **Facts** navigation tab as shown in Figure 1-66.

Figure 1-97 Opening a Business Rules Dictionary with Rules Designer



3. Select the type you want to modify. For example in the XML Facts table double-click the icon next to the **CustomerOrder** entry. This displays the Edit XML Fact dialog.
4. In the Edit XML Fact dialog, in the **Properties** table in the **Value Set** column select the cell for the appropriate property and from the list select the valueset you want to use. For example, for the property **order** select the **OrderAmount** valueset, as shown in Figure 1-67.

Figure 1-98 Associating the OrderAmount Valueset with CustomerOrder.order



5. In a similar manner, for the property **creditScore** select the **CreditScore** valueset.
6. Click **OK**.

How to Add a Decision Table for Order Approval

You create a Decision Table to process input facts and to produce output facts, or to produce intermediate conclusions that Oracle Business Rules can further process using additional rules or in another Decision Table.

While you work with rules you can use the rule validation features in Rules Designer to assist you. Rules Designer performs dictionary validation when you make any change to the dictionary. To show the validation log window, click the **Validate** button or select **View**>**Log** and select the **Business Rule Validation** tab. If you view the rules validation log you should see warning messages. You remove these warning messages as you create the Decision Table.

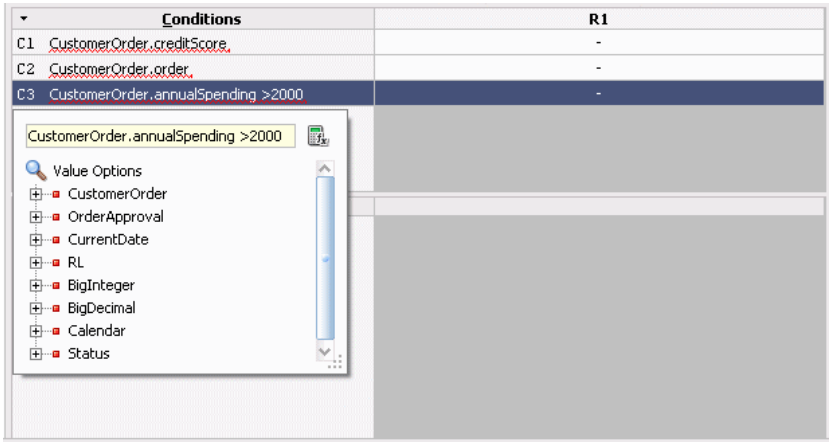
To use a Decision Table for rules in this sample application you work with facts representing a customer spending level and a customer credit risk for a particular customer and a particular order. Then, you use a Decision Table to create rules based on customer spending, the order amount, and the credit risk of the customer.

To add a Decision Table for order approval:

1. In Rules Designer, select **Ruleset_1** under the **Rulesets** navigation tab.
2. Click the **Add** button and from the list and select **Create Decision Table**.
3. In the Decision Table, click the **Add** button and from the list select **Condition**.
4. In the Decision Table, double-click **<edit condition>**. Then, in the navigator expand **CustomerOrder** and select **creditScore**. This enters the expression `CustomerOrder.creditScore` in the **Conditions** column.
5. Again, in the Decision Table, click the **Add** button and from the list select **Condition**.

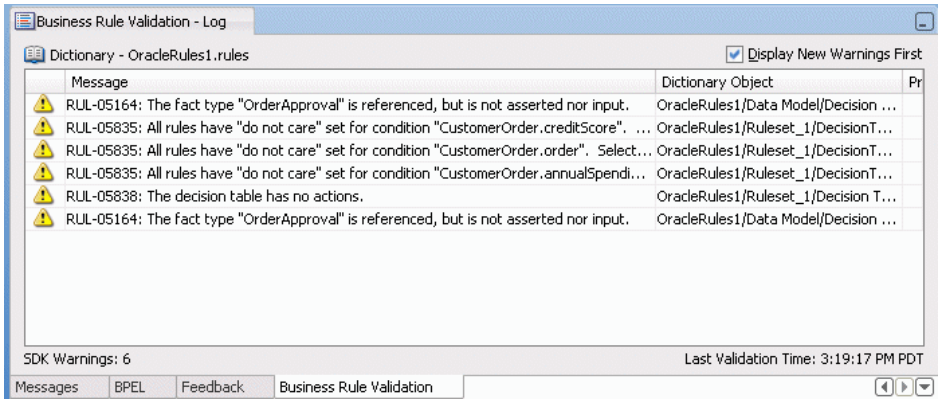
6. In the Decision Table, in the **Conditions** area double-click the **<edit condition>**. Then, in the navigator expand **CustomerOrder** and select **order**. This enters the expression `CustomerOrder.order`.
7. Again, in the Decision Table, click the **Add** button and from the list select **Condition**.
8. In the Decision Table, double-click **<edit condition>**.
9. In the navigator expand **CustomerOrder** and select **annualSpending**. In the text entry area, add `>2000`.

Figure 1-99 Adding the Annual Spending Entry to a Decision Table



10. Type **Enter** to accept the value. If you view the rules validation log you should see the warning messages. You remove these warning messages as you modify the Decision Table in later steps.

Figure 1-100 Adding Conditions to the CustomerOrder Decision Table



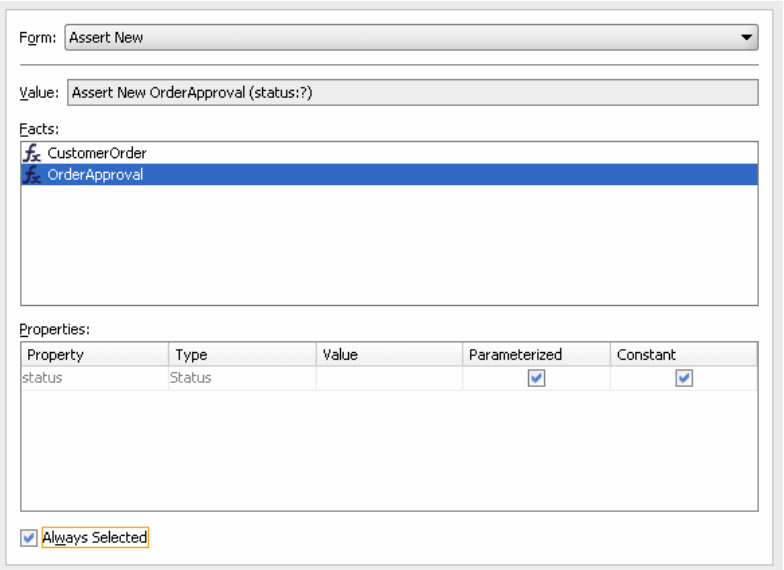
How to Create an action in a Decision Table

To create an action in a Decision Table:

1. In the Decision Table click the **Add** button and from the list select **Action** > **Assert New**.
2. In the **Actions** area, double-click **assert new(**. This displays the Action Editor dialog.

3. In the Action Editor dialog, in the **Facts** area select **OrderApproval**.
4. In the Action Editor dialog, in the Properties table for the property `status` select the **Parameterized** check box and the **Constant** check box. This specifies that each rule independently sets the status.
5. In the Action Editor dialog, select the **Always Selected** check box as shown in Figure 1-70.

Figure 1-101 Adding an Action to a Decision Table with the Action Editor Dialog



6. In the Action Editor dialog, click **OK**.

Next you need to add rules to the Decision Table and specify an action for each rule.

Split the Cells in the Decision Table and Add Actions

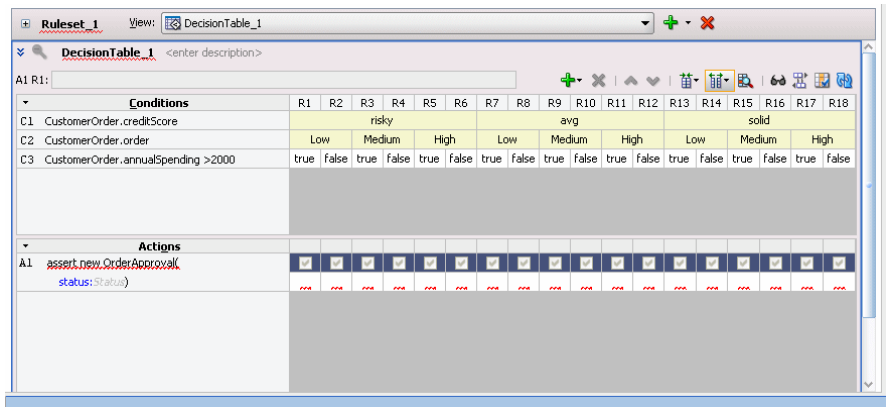
You can use the Decision Table split operation to create rules for the valuesets associated with the condition rows in the Decision Table. This creates one rule for every combination of condition valuesets. There are three order amount valuesets, three credit score valuesets, and two boolean valuesets for the annual spending amount for a total of eighteen rules (3 x 3 x 2 = 18).

To split cells in a decision table:

1. Select the Decision Table.
2. In the Decision Table, click the **Split Table** button and from the list select **Split Table**. The split table operation eliminates the "do not care" cells from the table. The table now shows eighteen rules that cover all ranges as shown in Figure 1-71.

These steps produce validation warnings for action cells with missing expressions. You fix these in later steps.

Figure 1-102 Splitting a Decision Table Using Split Table Operation



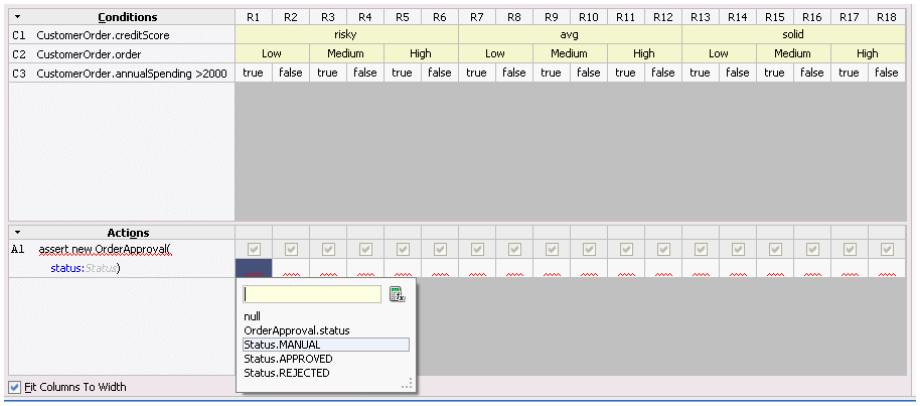
How to Add Actions for Each Rule in the Decision Table

In the Decision Table you specify a value for the status property associated with OrderApproval for each action cell in the **Actions** area. The possible choices are: `Status.MANUAL`, `Status.REJECTED`, or `Status.ACCEPTED`. In this step you fill in a value for status for each of the 18 rules. The values you enter correspond to the conditions that form each rule in the Decision Table.

To add actions for each rule in the decision table:

1. In the **Actions** area, double-click the action cell for the rule you want to work with, as shown in Figure 1-72.

Figure 1-103 Adding Action Cell Values to a Decision Table



2. In the list, select and enter a value for the action cell. For example, enter `Status.MANUAL`.
3. For each action cell, enter the appropriate value as determined by the logic of your application. For this sample application use the values for the Decision Table actions as shown in Table 1-21.
4. Select **Save All** from the **File** main menu to save your work.

Table 1-24 Values for Decision Table Actions

Rule	C1 creditScore	C2 order	C3 annualSpending > 2000	A1 OrderApproval status
R1	risky	Low	true	`Status.MANUAL`
R2	risky	Low	false	`Status.MANUAL`
R3	risky	Medium	true	`Status.MANUAL`
R4	risky	Medium	false	`Status.REJECTED`
R5	risky	High	true	`Status.MANUAL`
R6	risky	High	false	`Status.REJECTED`
R7	avg	Low	true	`Status.APPROVED`
R8	avg	Low	false	`Status.MANUAL`
R9	avg	Medium	true	`Status.APPROVED`
R10	avg	Medium	false	`Status.MANUAL`
R11	avg	High	true	`Status.MANUAL`
R12	avg	High	false	`Status.MANUAL`
R13	solid	Low	true	`Status.APPROVED`
R14	solid	Low	false	`Status.APPROVED`
R15	solid	Medium	true	`Status.APPROVED`
R16	solid	Medium	false	`Status.APPROVED`
R17	solid	High	true	`Status.APPROVED`
R18	solid	High	false	`Status.MANUAL`

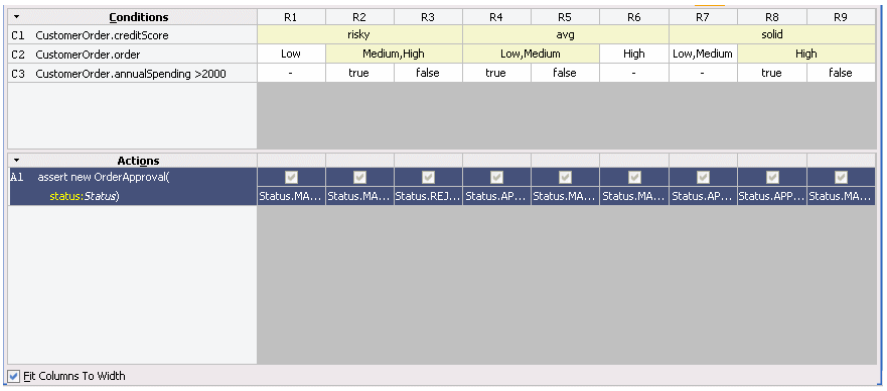
Compact the Decision Table

In this step you compact the rules to merge from eighteen rules to nine rules. This automatically eliminates the rules that are not needed and preserves the no gap, no conflict properties for the Decision Table.

To compact the decision table:

1. Select the Decision Table.
2. Click the **Resize All Columns to Same Width** button.
3. Click the **Compact Table** button and from the list select **Compact Table**. The compact table operation eliminates rules from the Decision Table. The Decision Table now shows nine rules, as shown in Figure 1-73.

Figure 1-104 Compacting a Decision Table Using Compact Table



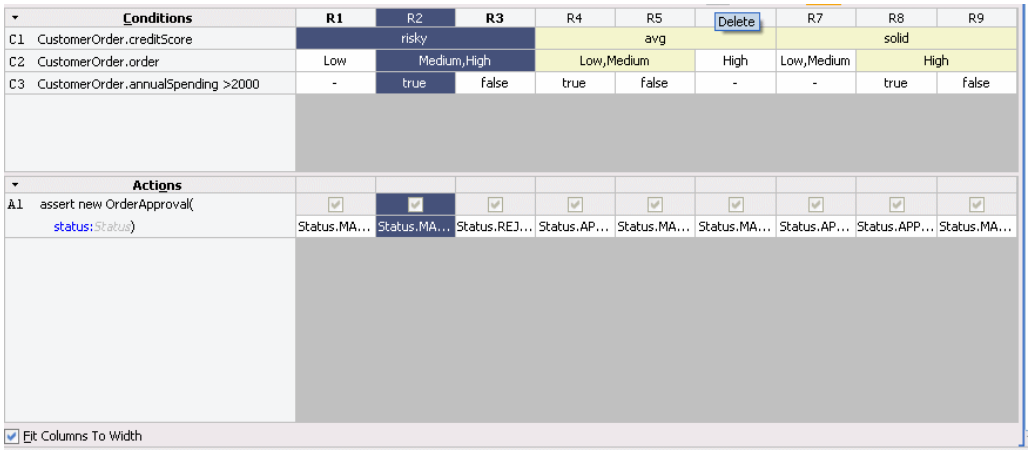
Replace Several Specific Rules with One General Rule

Notice that five of the nine remaining rules result in a manual order approval status. You can reduce the number of rules by deleting these five rules. Note it is often best practice to not do this (that is not replace several specific rules with one general rule). You need to compare the benefits of having fewer rules with the added complexity of managing the conflicts introduced when you reduce the number of rules.

To replace several specific rules with one general rule:

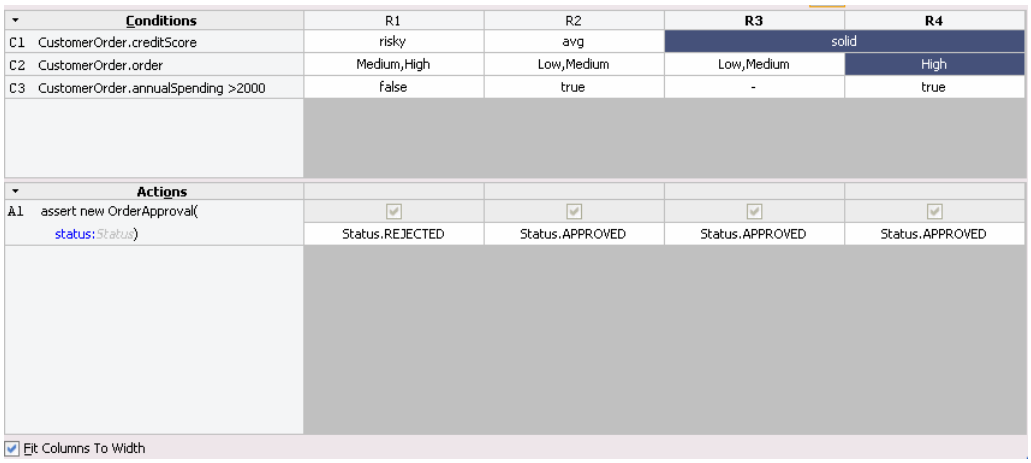
1. Select the Decision Table.
2. In the Decision Table, select a rule with OrderApproval status action set to `Status.MANUAL`. To select a rule, click the column heading. For example, click rule **R2** as shown in Figure 1-74.
3. Click **Delete** to remove a rule in the Decision Table. Be careful to click the delete button in the Decision Table area to delete a rule in the decision table (there is also a delete button shown in the **Ruleset** area that deletes the complete Decision Table).

Figure 1-105 Deleting Rules from a Decision Table



4. Repeat these steps to delete all the rules with action set to `Status.MANUAL`. This should leave the Decision Table with four rules as shown in Figure 1-75.

Figure 1-106 Decision Table After Manual Actions Removed



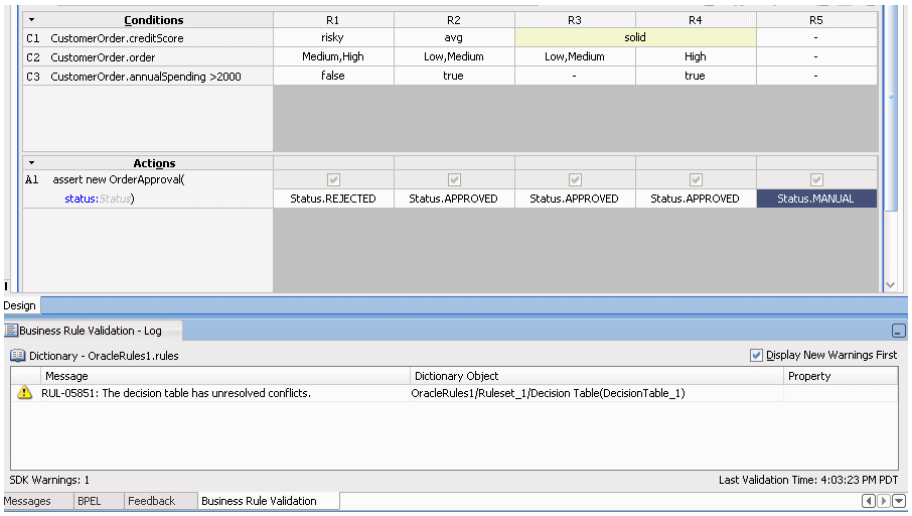
Add a General Rule

Now you can add a single rule to handle the manual case. After adding this rule you set the conflict policy with the option **Conflict Policy auto override** for conflict resolution.

To add a general rule:

1. In the Decision Table, click the **Add** button and from the list select **Rule**.
2. In the **Conditions** area, for the three conditions leave the "-" do not care value for each cell in the rule.
3. In the **Actions** area, enter `Status.MANUAL`, as shown in Figure 1-76. Notice that the **Business Rule Validation** log includes the warning `RUL-05851` for unresolved conflicts.

Figure 1-107 Decision Table with Conflicting Rules



4. Show the conflicting rules by clicking the **Toggle Display of Conflict Resolution** button, as shown in Figure 1-77.

Figure 1-108 Adding a Rule to Handle Status Manual

Conditions	R1	R2	R3	R4	R5
C1 CustomerOrder.creditScore	risky	avg	solid		-
C2 CustomerOrder.order	Medium,High	Low,Medium	Low,Medium	High	-
C3 CustomerOrder.annualSpending >2000	false	true	-	true	-
Conflict Resolution					
Conflict	R5	R5	R5	R5	R1, R2, R3, R4
Actions					
A1 assert new OrderApproval(	☑	☑	☑	☑	☑
status:Status)	Status.REJECTED	Status.APPROVED	Status.APPROVED	Status.APPROVED	Status.MANUAL

Fit Columns To Width

How to Enable the Auto Override Conflict Resolution Policy

To enable the auto override conflict resolution policy:

1. In the Decision Table click **Show Advanced Settings** (next to the Decision Table name).
2. In the Conflict Policy list, select **auto override**. After adding the manual case rule and selecting **auto override**, notice that the conflicts are resolved and special cases override the general case, as shown in Figure 1-78.

Figure 1-109 Adding a Rule to Handle Status Manual with Auto Override Conflict Policy

Conditions	R1	R2	R3	R4	R5
C1 CustomerOrder.creditScore	risky	avg	solid		-
C2 CustomerOrder.order	Medium,High	Low,Medium	Low,Medium	High	-
C3 CustomerOrder.annualSpending >2000	false	true	-	true	-
Conflict Resolution					
Override	R5	R5	R5	R5	
Actions					
A1 assert new OrderApproval(	☑	☑	☑	☑	☑
status:Status)	Status.REJECTED	Status.APPROVED	Status.APPROVED	Status.APPROVED	Status.MANUAL

Fit Columns To Width

How to Check the Business Rule Validation Log for Order Approval

Before you can deploy the application you need to make sure the dictionary validates without warnings. If there are any validation warnings, you need to fix any associated problems. To

validate the dictionary, in the Business Rule Validation Log, check for any validation warnings. If there are warnings, perform appropriate actions to correct the problems.

How to Deploy the Order Approval Application

Business rules created in a SOA application are deployed as part of the SOA composite when you create a deployment profile in Oracle JDeveloper. You deploy a SOA composite application to Oracle WebLogic Server.

To deploy and run the order approval application:

1. If you have not started your application server instance, then start the Oracle WebLogic Server.
2. In the Application Navigator, right-click the **OrderApproval** project and select **Deploy** > **OrderApproval** > to the appropriate server name.

   Then the SOA Deployment Configuration dialog displays. Select your Application connection which you either have created already or you can create it now. The connection contains the authorization and other connection information (server name, port, etc).
3. Click **Next**.
4. In Select Server select or create and then select your application connection.
5. Click **Next, Next** and **Finish.**

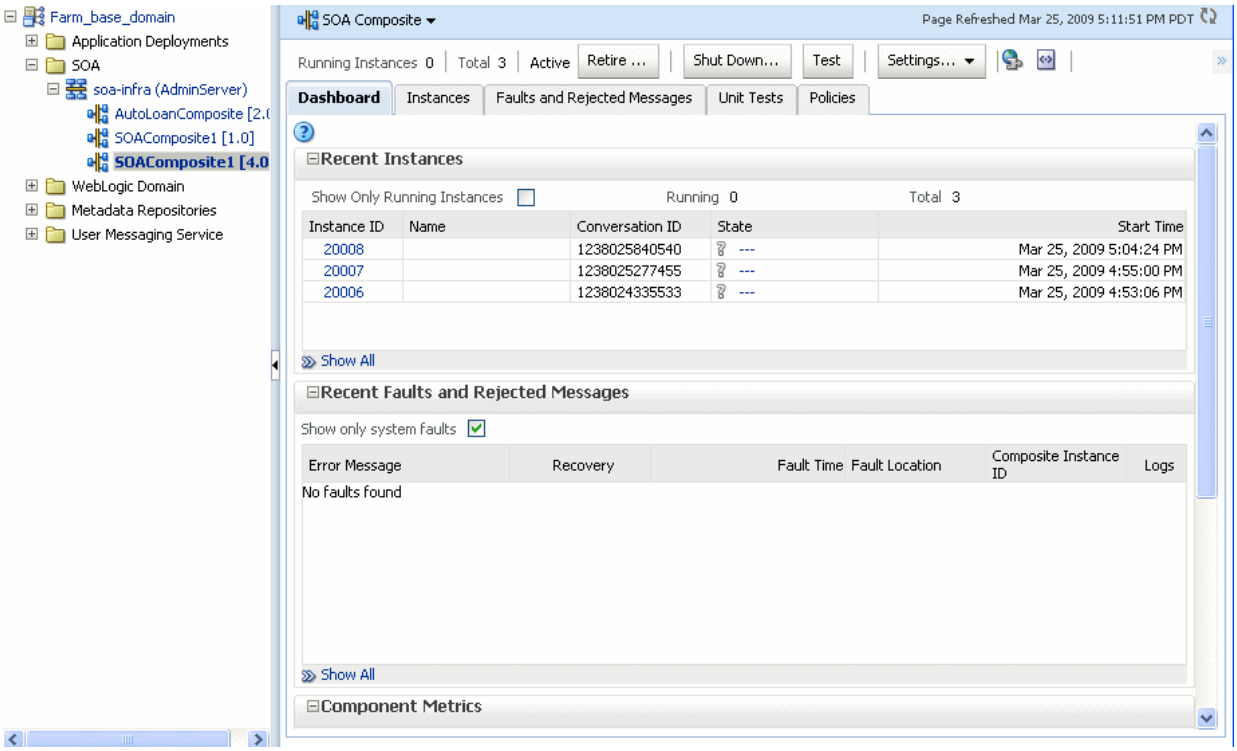
How to Test the Order Approval Application

After deploying the application you can test the Decision Table in the SOA composite application with the Oracle Enterprise Manager Fusion Middleware Control Console.

To test the application:

1. Open the composite application in Oracle Enterprise Manager Fusion Middleware Control Console, as shown in Figure 1-79.

Figure 1-110 Testing the Order Approval Application



2. Click **Test**.
3. In the **Input Arguments** area, select **XML View**. Replace the XML with the contents of the sample input for testing Order Approval application example as shown below.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body xmlns:ns1="http://xmlns.oracle.com/OracleRules1/
OracleRules1_DecisionService_1">
        <ns1:callFunctionStateless name="OracleRules1_DecisionService_1">
                   <ns1:parameterList xmlns:ns3="http://example.com/ns/
customerorder">
                           <ns3:CustomerOrder>
                                   <ns3:name>Gary</ns3:name>
                                       <ns3:creditScore>600</ns3:creditScore>
                                       <ns3:annualSpending>2001.0</
ns3:annualSpending>
                                       <ns3:value>High</ns3:value>
                                       <ns3:order>100.0</ns3:order>
                    </ns3:CustomerOrder>
                   </ns1:parameterList>
           </ns1:callFunctionStateless>
       </soap:Body>
</soap:Envelope>
```

4. Replace the values in the input shown in step 3 as desired for your test.
5. Click **Test Web Service**.
6. In the **Response** tab, view the results. For example, for this input:

```
/OracleRules1_DecisionService_1" xmlns:ns2="http://xmlns.oracle.com/bpel">
      <resultList>
          <OrderApproval:OrderApproval      xmlns:OrderApproval="http://
example.com/ns/customerorder"
```

```
xmlns="http://example.com/ns/customerorder">
                <status>approved</status>
            </OrderApproval:OrderApproval>
        </resultList>
    </callFunctionStatefulDecision>
```

Editing Decision Tables in Microsoft Excel

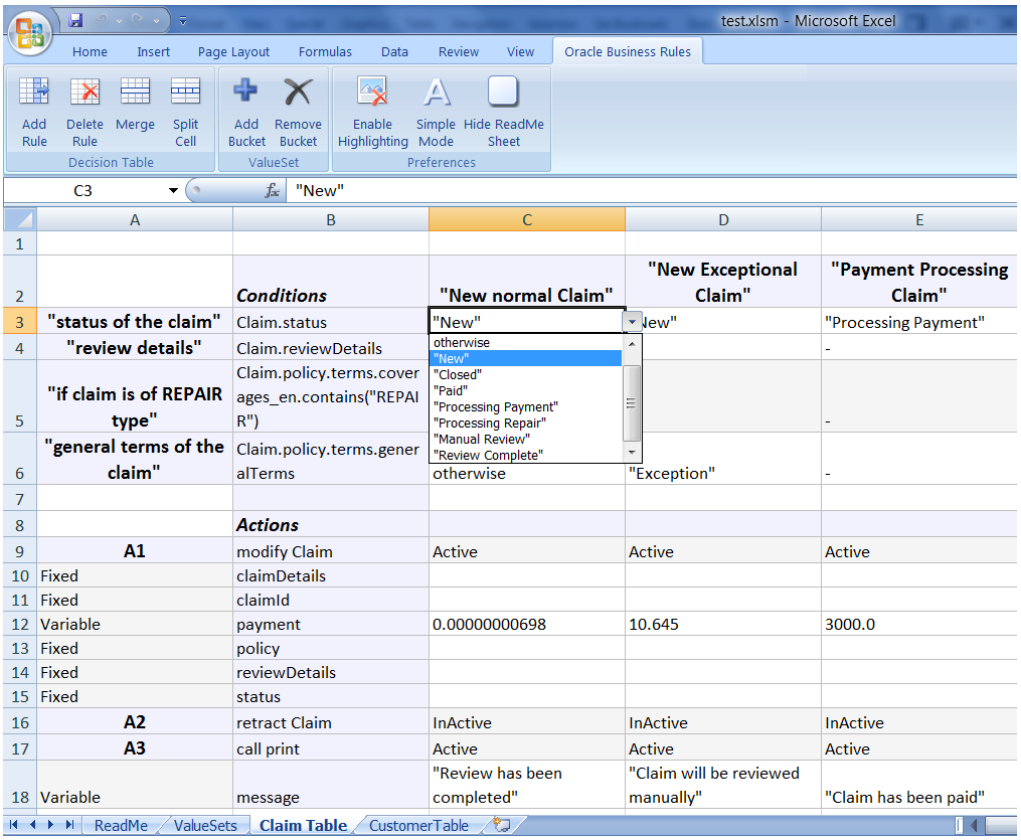
Business users may find that editing Decision Tables is easier to do in Microsoft Excel. New functionality enables both developers and business users to export and edit Decision Tables in Excel and then import the Decision Tables back into the dictionary.

You can export and edit Decision Tables at design-time in Oracle JDeveloper or Business Process Composer. At runtime, you can export and edit in SOA Composer. You can export one or more Decision Tables from a Rule dictionary to the same Excel workbook.

When you import back into the dictionary, you can create a new dictionary, overwrite the existing dictionary, or perform a Diff-Merge. The Diff-Merge enables you to compare dictionaries and accept (merge) or reject any differences.

The Excel workbook structure consists of several worksheets: a Readme sheet, a Value Set sheet, and one sheet for each exported Decision Table, as shown in Figure 1-111. Only Rules and Value Sets can be edited in Excel. You can export to .xlsm (default) or .xls.

Figure 1-111 Microsoft Excel Workbook



When you open the Excel workbook, the macros are disabled by default. If you enable the macros, a new tab called Oracle Business Rules, appears. This tab enables you to add or delete rules, merge or split cells, and add or remove values from value sets. You can also disable or enable highlighting, use a simple or advanced mode and hide or show the Readme worksheet.

You can edit with the macros disabled, though you will not be able to:

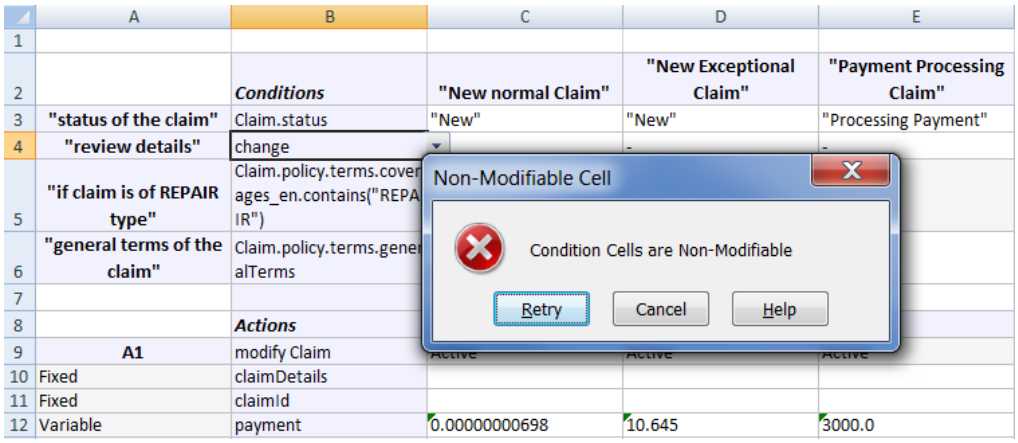
- Choose values from drop lists for restricted cells.
- Edit free form cells.
- Copy and paste a range of cells to add a rule or Value Set.
- Delete a range of cells to delete a rule or Value Set.
- Split or merge cells.
- Create Value Sets automatically.
- Validate the structure of Decision Tables or Value Sets.

Using the predefined macros, you can:

- Add and delete rules.
- Split or merge cells.
- Add or delete Value Sets.
- Editable cells include:
  - Description for Rules, Conditions, Actions.
  - Condition and Action nodes.
  - Action state.
  - Parameterized options for Action parameters.
- Non-editable cells include:
  - Condition expressions.
  - Action expressions.
  - Action parameters.

  If you try to edit these cells, you will get an error message, as shown in Figure 1-112.

Figure 1-112 Non-Modifiable Cell



Understanding What is Exported

In the SDK, there are shared Value Sets that can be associated with multiple conditions across Decision Tables. However, in Excel there are no shared Value Sets--each condition has its own Value Set--so you can only export a Value Set if it is modifiable in Excel. The Value Sets that are non-modifiable include:

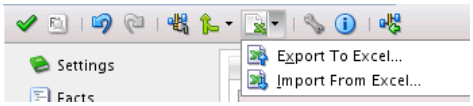
- Linked Dictionary Value Sets.
- Enums.
- Internal Value Sets, for example, boolean Value Sets.

In the worksheet, you can only select values from the drop down for the conditions associated with non-modifiable Value Sets. A highlighting mechanism informs you which conditions are associated with non-modifiable Value Sets.

How to Export Decision Tables

The export and import functionality is invoked using the toolbar button, as shown in Figure 1-113.

Figure 1-113 Export and Import Toolbar Button



To export Decision Table:

1. In Rules Designer, click **Export to Excel**.
2. In the **Export to Excel** dialog box, select the **Format** and browse to the folder where you want to save the workbook.
3. Click **Add** and select the Decision Table(s) to export and click **OK.**
4. Check the **Read Only Value Set** check box to make all of the value sets read-only in Excel. There will not be any Value Sets sheet in the Excel workbook. All conditions will have drop down menus from which values can be selected but no values can be added or removed.
5. Click **Export.** You can now open the workbook and edit the Decision Table.

How to Import Edited Decision Tables Back to the Dictionary

The export and import functionality is invoked using the toolbar button, as shown in Figure 1-113. You can only import Excel workbooks that have been previously exported.

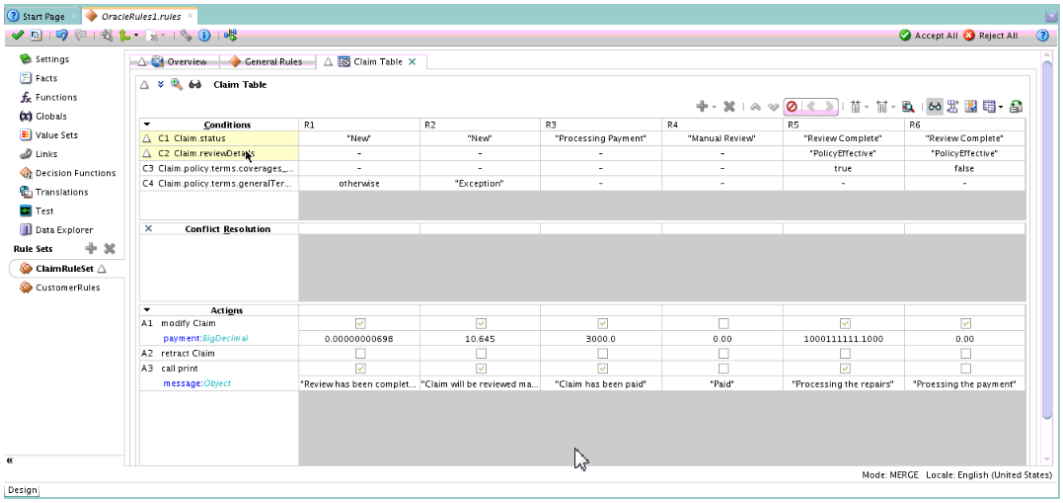
To import edited Decision tables:

1. In Rules Designer, click **Import from Excel**.
2. In the **Import from Excel** dialog box, select the **File** to browse to the folder where you saved the workbook.

3. The **Perform Diff-Merge on Import** check box is selected by default. Browse to the **Base Dictionary** that you want to compare your file to. The base dictionary is required for a 3 way diff-merge.
4. Clear the **Perform Diff-Merge on Import** check box and select **Create New** or **Overwrite.**
5. Click **Import.** The decision table is imported into Rules Designer, where you can accept or reject changes, as shown in Figure 1-114. Each changed artifact is flagged with a change icon. Merging dictionaries should be done with caution.

   For more information about using the Diff-Merge, see *How to Compare or Merge Two or More Dictionaries* in *Designing Business Rules with Oracle Business Process Management*.

Figure 1-114 Perform Diff-Merge on Import



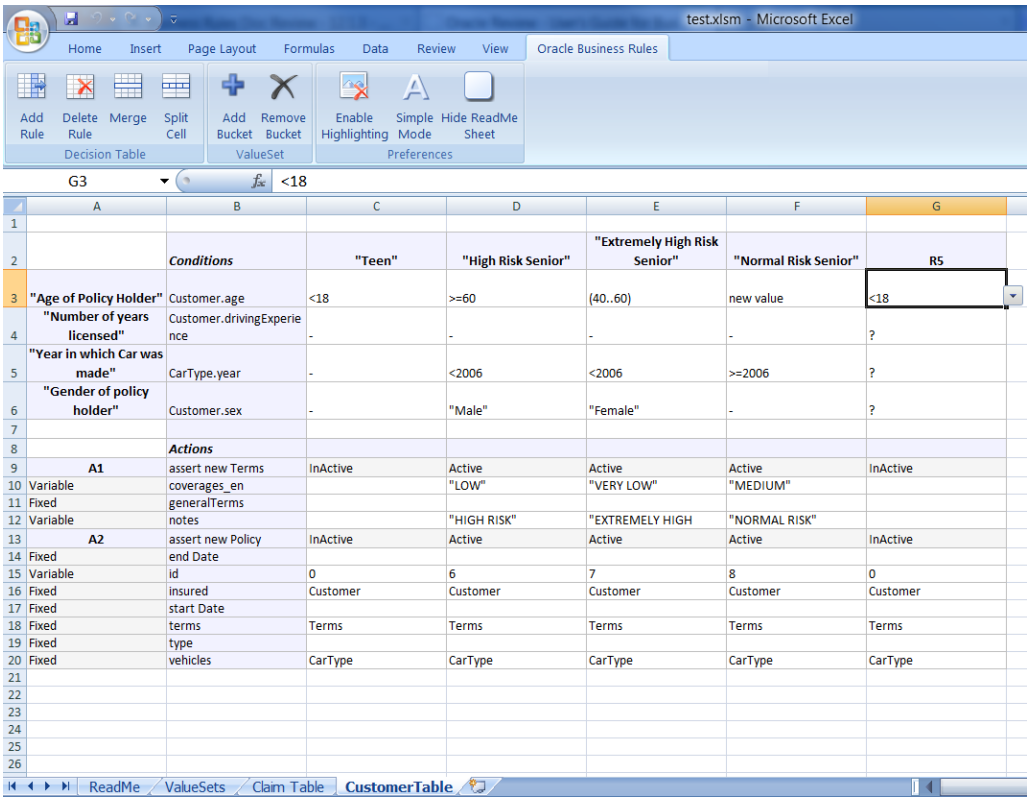
How to Edit Decision Tables in Excel

In Excel, enable the macros to view the Oracle Business Rules tab, which provides you with options to author rules, edit Value Sets, and set preferences.

Adding or Deleting Rules and Merging or Splitting Cells

For each Decision Table worksheet, you can add a rule, as shown in Figure 1-115, delete rules, and merge or split cells.

Figure 1-115 Oracle Business Rules tab in Excel

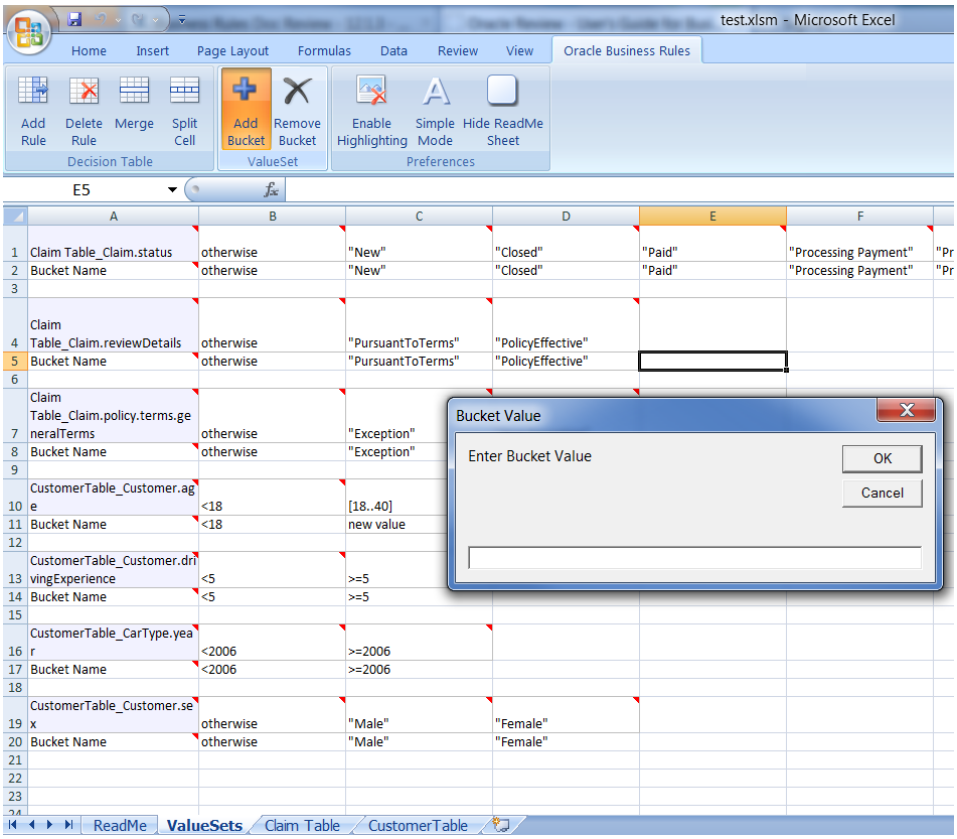


	A	B	C	D	E	F	G
1							
2		***Conditions***	**"Teen"**	**"High Risk Senior"**	**"Extremely High Risk Senior"**	**"Normal Risk Senior"**	**R5**
3	**"Age of Policy Holder"**	Customer.age	<18	>=60	(40..60)	new value	<18
4	**"Number of years licensed"**	Customer.drivingExperience	-	-	-	-	?
5	**"Year in which Car was made"**	CarType.year	-	<2006	<2006	>=2006	?
6	**"Gender of policy holder"**	Customer.sex	-	"Male"	"Female"	-	?
7							
8		***Actions***					
9	**A1**	assert new Terms	InActive	Active	Active	Active	InActive
10	Variable	coverages_en		"LOW"	"VERY LOW"	"MEDIUM"	
11	Fixed	generalTerms					
12	Variable	notes		"HIGH RISK"	"EXTREMELY HIGH	"NORMAL RISK"	
13	**A2**	assert new Policy	InActive	Active	Active	Active	InActive
14	Fixed	end Date					
15	Variable	id	0	6	7	8	0
16	Fixed	insured	Customer	Customer	Customer	Customer	Customer
17	Fixed	start Date					
18	Fixed	terms	Terms	Terms	Terms	Terms	Terms
19	Fixed	type					
20	Fixed	vehicles	CarType	CarType	CarType	CarType	CarType
21							
22							
23							
24							
25							
26							

Adding or Removing Value Sets

In the ValueSets tab, you can add or remove Value Sets, as shown in Figure 1-116. Depending on the cell you click in, your options will vary: endpoints or values.

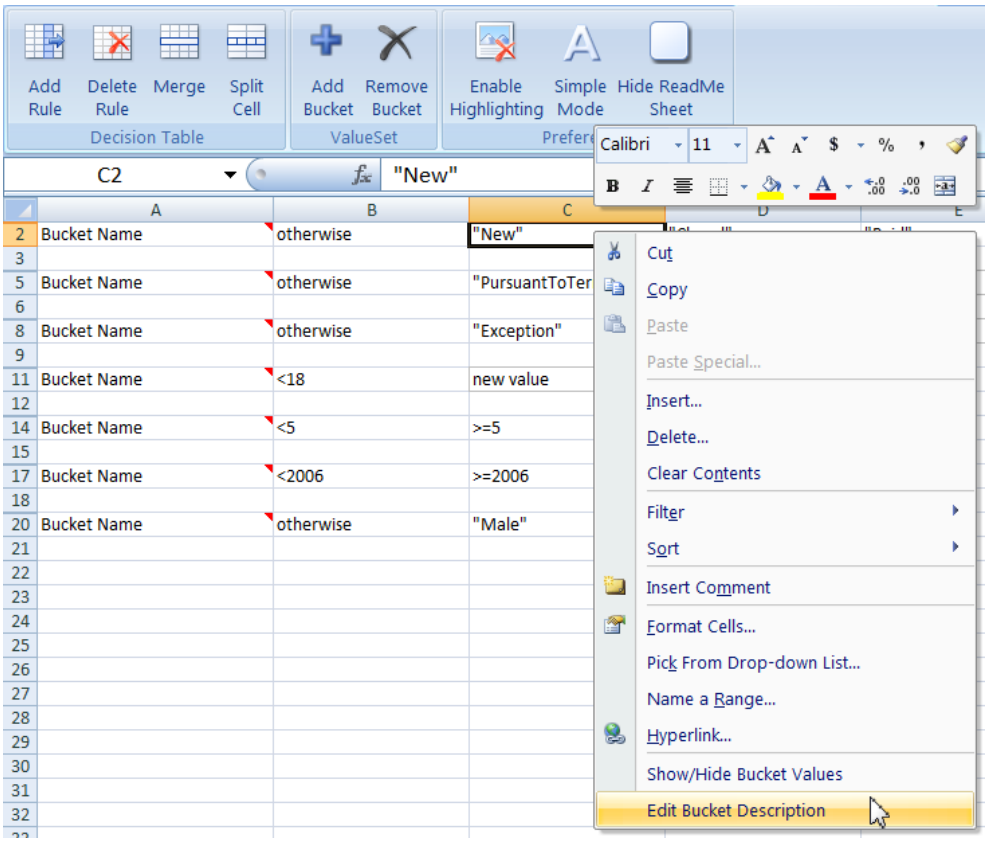
Figure 1-116 Value Sets Worksheet



Showing or Hiding Value Sets and Editing the Description

On the Value Sets worksheet, right click and select **Show/Hide Values** to toggle between viewing or hiding values as shown in Figure 1-117. You can also right click and select **Edit Bucket Description** to change the description.

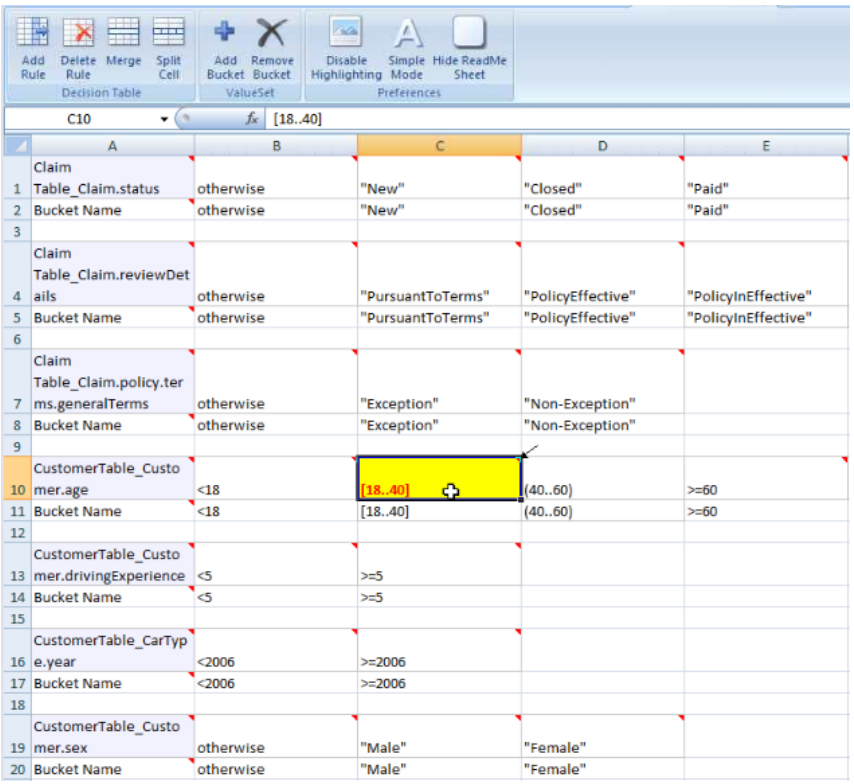
Figure 1-117 Show/Hide Value Sets



Setting Preferences

In the Value Sets tab, click Enable Highlighting or Disable Highlighting, as shown in Figure 1-118.

Figure 1-118 Enabling Highlighting



	A	B	C	D	E
1	Claim Table_Claim.status	otherwise	"New"	"Closed"	"Paid"
2	Bucket Name	otherwise	"New"	"Closed"	"Paid"
3					
4	Claim Table_Claim.reviewDetails	otherwise	"PursuantToTerms"	"PolicyEffective"	"PolicyInEffective"
5	Bucket Name	otherwise	"PursuantToTerms"	"PolicyEffective"	"PolicyInEffective"
6					
7	Claim Table_Claim.policy.terms.generalTerms	otherwise	"Exception"	"Non-Exception"	
8	Bucket Name	otherwise	"Exception"	"Non-Exception"	
9					
10	CustomerTable_Customer.age	<18	[18..40]	(40..60)	>=60
11	Bucket Name	<18	[18..40]	(40..60)	>=60
12					
13	CustomerTable_Customer.drivingExperience	<5	>=5		
14	Bucket Name	<5	>=5		
15					
16	CustomerTable_CarType.year	<2006	>=2006		
17	Bucket Name	<2006	>=2006		
18					
19	CustomerTable_Customer.sex	otherwise	"Male"	"Female"	
20	Bucket Name	otherwise	"Male"	"Female"	

Using Simple or Advanced Mode

In your worksheet, click Simple Mode or Advanced Mode to toggle between the two modes.

Simple mode displays only the descriptions of conditions and actions and not the actual expressions. Also, action parameters are displayed, but you cannot specify them as fixed or variable.

Advanced mode displays both the descriptions and expressions for conditions and actions, as shown in Figure 1-119. Also, you can specify the action parameter type from fixed and variable, which is equivalent to specifying "Parameterized/Constant" in the SDK.

Figure 1-119 Advanced Mode

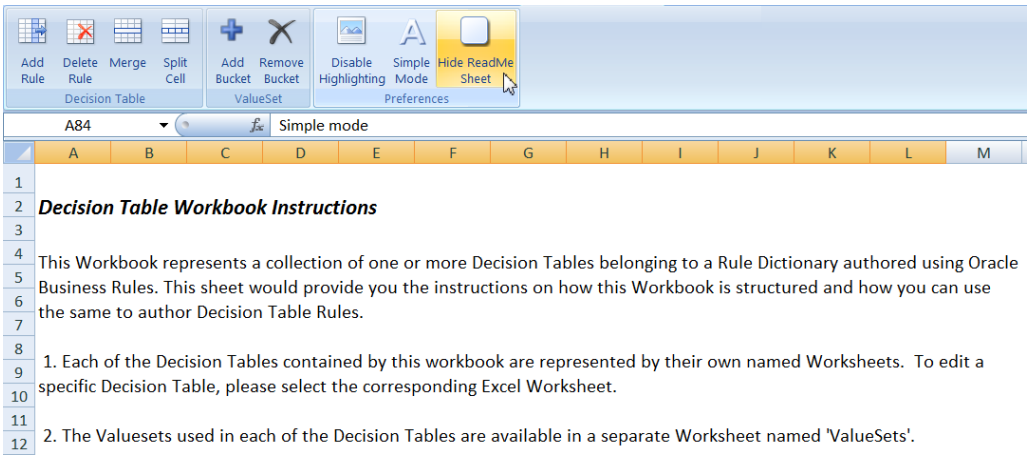
A14 | Fixed

	A	B	C	D	E	F
1						
2		***Conditions***	**"Teen"**	**"High Risk Senior"**	**"Extremely High Risk Senior"**	**"Normal Risk Senior"**
3	**"Age of Policy Holder"**	Customer.age	<18	>=60	(40..60)	new value
4	**"Number of years licensed"**	Customer.drivingExperience	-	-	-	-
5	**"Year in which Car was made"**	CarType.year	-	<2006	<2006	>=2006
6	**"Gender of policy holder"**	Customer.sex	-	"Male"	"Female"	-
7						
8		***Actions***				
9	**A1**	assert new Terms	InActive	Active	Active	Active
10	Variable	coverages_en		"LOW"	"VERY LOW"	"MEDIUM"
11	Fixed	generalTerms				
12	Variable	notes		"HIGH RISK"	"EXTREMELY HIGH	"NORMAL RISK"
13	**A2**	assert new Policy	InActive	Active	Active	Active
14	Fixed	d Date				
15	Variable	id	0	6	7	8
16	Fixed	insured	Customer	Customer	Customer	Customer
17	Fixed	start Date				
18	Fixed	terms	Terms	Terms	Terms	Terms
19	Fixed	type				
20	Fixed	vehicles	CarType	CarType	CarType	CarType

Hiding or Showing the Readme Worksheet

Click Hide or Show ReadMe Sheet to toggle between the modes, as shown in Figure 1-120. The ReadMe worksheet provides helpful information about how to use the features on the Oracle Business Rules tab.

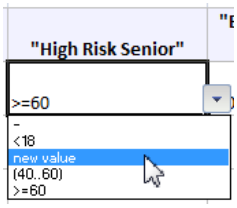
Figure 1-120 Show/Hide Readme



Editing Condition Cells

You can choose from the drop down or use auto-addition to add new values, shown in Figure 1-121. For some of the condition cells, you can only choose values from the drop down menu. These cells have been differentiated by using color code. Any conditions you change between a Value Set or Decision Table are automatically synced.

Figure 1-121 Editing Conditions



Editing Actions

You can select the action state (active/inactive) from the drop down, as shown in Figure 1-122.

Figure 1-122 Editing Action States

8		***Actions***			
9	**A1**	assert new Terms	InActive	Active	Active
10	Variable	coverages_en	Active InActive	DW"	"VERY LOW"
11	Fixed	generalTerms			
12	Variable	notes		"HIGH RISK"	"EXTREMELY HIGH
13	**A2**	assert new Policy	InActive	Active	Active
14	Fixed	end Date			
15	Variable	id	0	6	7
16	Fixed	insured	Customer	Customer	Customer
17	Fixed	start Date			
18	Fixed	terms	Terms	Terms	Terms
19	Fixed	type			
20	Fixed	vehicles	CarType	CarType	CarType

Editing Expressions

You can edit the values of action expression cells. Use care to maintain the validity of these cells when editing.

Editing Action Expression Parameters

You can make action parameters fixed or variable, as shown in Figure 1-123. If the action parameter is fixed, then all the rules will have the same value for that particular parameter. If the action parameter is variable, then different rules can have different values for that particular parameter.

Figure 1-123 Editing Action Expression Parameters

	Actions		
A1	assert new EURentRulesBase.DriverType	Active	InActive
Variable	e	18	25
Fixed Variable	stName	"FirstName"	"FirstName"
Fixed	lastName	null	null
Variable	licenseNumber	"ABCD1234"	"ABCD1234"

Editing Descriptions

You can edit descriptions for actions, conditions, and rules. If the description is not provided for any of the action or condition or rule then it will be defaulted to "A", "C" or "R" followed by a number which denotes its position in the decision table, respectively.

Figure 1-124 Editing Descriptions

	A	B	C	D
1				
2		***Conditions***	**Minor**	**R2**
3	**C1**	EURentRulesBase.DriverType.age	<18	3..60)
4	**C2**	EURentRulesBase.TCaseEvent.milestoneEvent.milestoneEvent	-	-
5	**C3**	EURentRulesBase.DriverType.firstName	"John"	"Carter"
6				
7		***Actions***		
8	**A1**	assert new EURentRulesBase.DriverType	Active	InActive
9	Variable	age	18	25
10	Fixed	firstName	"FirstName"	"FirstName"
11	Fixed	lastName	null	null
12	Variable	licenseNumber	"ABCD1234"	"ABCD1234"

Using the Auto-Addition Feature

You can add values in the value sets in two ways:

1. Go to the specific value set in the value sets worksheet. In the Oracle Business Rules tab, click Add Bucket.
2. Enter a value (in case of LOV valuesets) or end point (in case of Range valuesets) in the condition cell. This is called auto-addition as the value will be automatically added to the corresponding value set, as shown in Figure 1-125.

Figure 1-125 Entering a Value in the Condition Cell

1					
2		***Conditions***	**Minor**	**R2**	**R3**
3	**C1**	EURentRulesBase.DriverType.age	<18	[18..60)	60
4	**C2**	EURentRulesBase.TCaseEvent.milestoneEvent.milestoneEvent	-	- <18 [18..60) >=60	
5	**C3**	EURentRulesBase.DriverType.firstName	"John"	"Carter"	-

The value set above has three values: – 1) <18 , 2) [18..60) , and 3) >=60.

3. To add a new value, for example, [18..30] and (30..60), type 30 in the cell as shown in Figure 1-126 and press Enter.

Figure 1-126 Adding a New Value

	A	B	C	D	E
1					
2		***Conditions***	**Minor**	**R2**	**R3**
3	**C1**	EURentRulesBase.DriverType.age	<18	30	60
4	**C2**	EURentRulesBase.TCaseEvent.milestoneEvent.milestoneEvent	-	-	-

4. After you press enter, the value will be added to the value set and will be shown in the drop-down as shown in Figure 1-127.

Figure 1-127 Value is Auto-Added

	A	B	C	D	E
1					
2		***Conditions***	**Minor**	**R2**	**R3**
3	**C1**	EURentRulesBase.DriverType.age	<18	[18..30]	60
4	**C2**	EURentRulesBase.TCaseEvent.milestoneEvent.milestoneEvent	-	- <18 [18..30] (30..60) >=60	
5	**C3**	EURentRulesBase.DriverType.firstName	"John"	"Carter"	-

Various highlighting techniques are used to inform you about auto-added values in the value set, see the following examples. The comment and the highlighting of the value is removed after you select another value for any other rule for that condition or if a new value is added in the same value set.

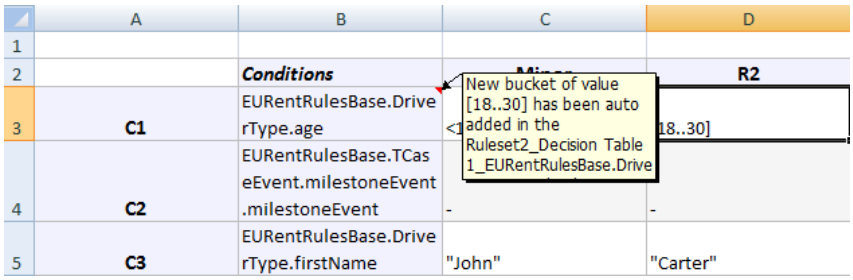
The first is to highlight the newly added value in the value set sheet as shown in Figure 1-128.

Figure 1-128 Highlighted Value Set

	A	B	C	D
10	Decision Table 2_EURentRulesBase.DriverType.age	<21	[21..60)	>=60
11	Bucket Name	<21	[21..60)	>=60
12				
13	Decision Table 2_Float.POSITIVE_INFINITY	<10.8	[10.8..20.0)	[20.0..50.5)
14	Bucket Name	<10.8	[10.8..20.0)	[20.0..50.5)
15				
16	Ruleset2_Decision Table 1_EURentRulesBase.DriverType.age	<18	[18..30]	(30..60)
17	Bucket Name	<18	[18..30]	(30..60)
18				
19	Ruleset2_Decision Table 1_EURentRulesBase.DriverType.firstName	otherwise	"John"	"Carter"

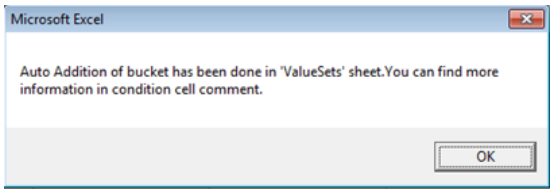
The second is the addition of a comment in the condition cell, as shown in Figure 1-129.

Figure 1-129 Comments in Condition Cells



The third is to print a message box, shown in Figure 1-130. Note that the box is only shown the first time when the value is auto-added.

Figure 1-130 Message Dialog



Aliases of Values in the Value Sets Worksheet

In the value sets sheet, there are two rows for every value set. The first row denotes the value and the second one denotes the alias of the value. It is the alias of the value that is shown in the drop-down of condition cells.The aliases can be edited. Any change made in aliases will be immediately available in corresponding condition cells.

Syncing Value Sets and Conditions

The value sets and condition cells are always in sync. Any change made in value set is promptly synced with the condition cells whether it is an addition/deletion of any value, or any change in the alias. The sync is always maintained between value set and the corresponding condition cells.

Modifying MDS Configuration Using MBeans

You can use the MBean Browser to perform advanced configuration of MDS parameters.

For more information about configuring MDS using MBeans, see *Changing the MDS Configuration Attributes Using Fusion Middleware Control* in *Administering Oracle Fusion Middleware*.

You must already have deployed an Oracle ADF application and have Enterprise Manager Fusion Middleware Control available to access the application.

To modify MDS configurations using the System MBean Browser:

1. From the navigation pane, expand **Application Deployments**, then click the application that you want to configure.

2. From the Application Deployment menu, choose **MDS Configuration**.
3. Click **Configuration MBean Browser** or **Runtime MBean Browser.**
4. Select the MBean and the attribute you want to view or modify.
5. Change the value and click **Apply**.
6. In the left pane, select the parent ADF MBean **ADFConfig**.
7. In the right pane, click the **Operations** tab and click **save**.

   The new values you have edited are written to MDS after you click **save** from the parent MBean.

ADF Business Components

This section provides a high-level overview of ADF Business Components, including a description of the key features they provide for building your business services. Features described include entity objects, view objects, and application modules.

This section includes the following topics:

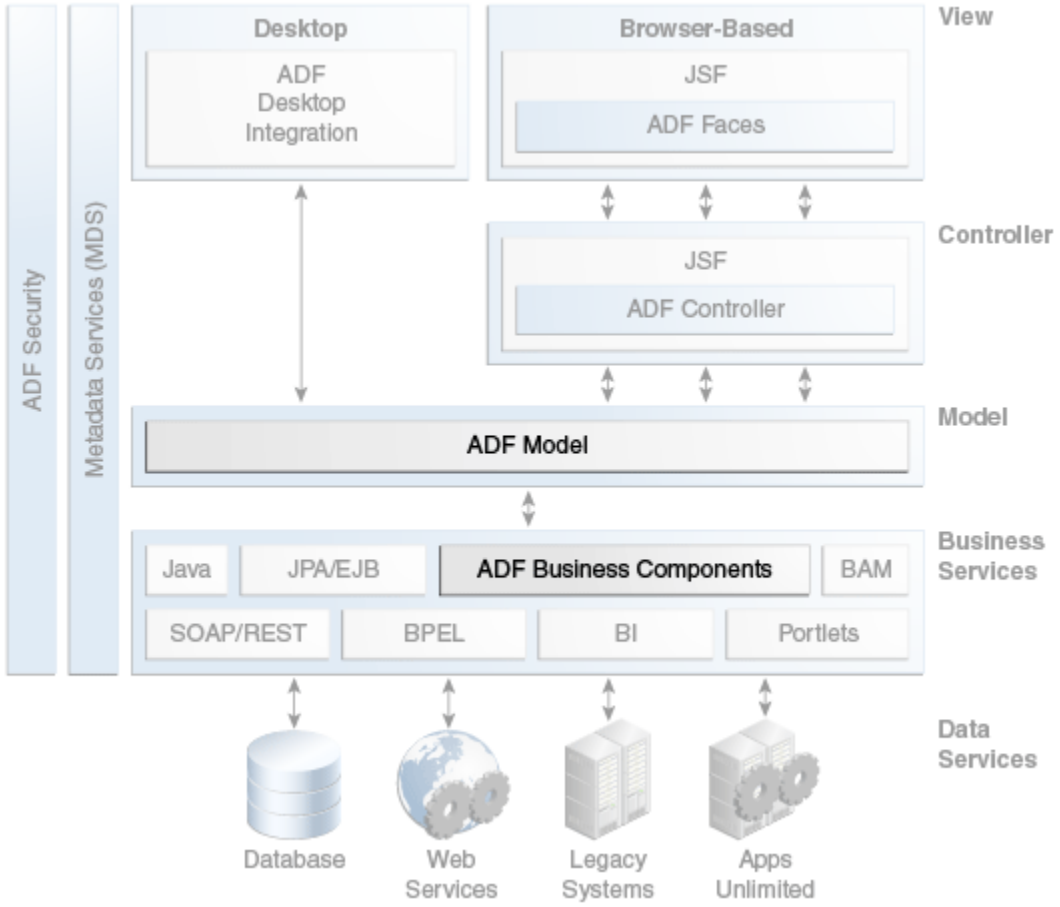
About ADF Business Components

Oracle ADF architecture is based on Model-View-Controller (MVC) design pattern that consists of four layers – Model, View, Controller, and Business Service. The Business Service layer is supported by the ADF Business Components, which is a framework for managing transactions with data sources and is integrated with JDeveloper tools.

ADF Business Components is a technology to create reusable data-aware business services with minimal developer coding. Developers can use wizards and visual editors to create ADF Business Components services without writing any Java code. It is also possible to extend the core ADF Business Components classes to create more advanced functionality. ADF Business Components services are exposed through ADF Model for use by the application's view layer.

Figure 1-131 shows how ADF Business Components fit into the ADF technology stack. Note that ADF Business Components features directly integrate with ADF Model.

Figure 1-131 ADF Architecture with Business Components



In addition, you can expose applications that you create with ADF Business Components as services that can be consumed by other Fusion web applications, composite applications that adhere to the Service Component Architecture (SCA), and other applications via web service calls.

Core Benefits of ADF Business Components

Tight integration with JDeveloper tools simplifies building the business service layer of the Fusion web application. The resulting business objects based on the ADF Business Components framework support managing data transactions at application runtime.

ADF Business Components provides the following benefits for developers of business services:

- Management of database access, including connection, data retrieval, locking of records, and insertion and update of records.
- Ability to create data models that are tailored for specific types of end users, with only the necessary data exposed.
- Creating of data model relationships in addition to those defined by the database.

- Ability to use declarative rules to enforce required fields, primary key uniqueness, data precision-scale, and foreign key references.
- Capturing and enforcing both simple and complex business rules, programmatically or declaratively, with multilevel validation support.
- Implementing end-user Query-by-Example data filtering without code.
- Ability to expose components as services that can be integrated with other Fusion web applications and consumed by SOA composite applications.
- Ability to raise business events to launch business processes and trigger synchronization of external systems.
- A built-in facility for application state management that enables application failover and the handling of user sessions over multiple nodes in clustered and high availability server environments.
- Features geared toward performance optimization, such as shared application modules to handle static data and application module pooling.
- Wizards and visual editors in JDeveloper that generate XML definitions for the components that you can also edit manually.

Key Concepts of ADF Business Components

The ADF Business Components framework supports modeling data sources for use in Fusion web applications based on declarative business objects that define object hierarchies (master-detail relationships) and that shape the data for display to the end user through application-specific views. The framework manages CRUD (create, read, update, delete) transactions at application runtime with a minimum of coding required.

ADF Business Components is an implementation of Java-based business services that directly incorporate ADF Model. This section provides an overview of the role of business services and how ADF Business Components implements business services.

Implementation of Business Services

Business services are behind-the-scenes components that mediate between an MVC application and a data source (usually a database). In general, business services are responsible for the following tasks:

- Retrieving data requested by the rest of the application
- Representing this data as Java objects usable by the rest of the application (object-relational ["O/R"] mapping)
- Persisting changes made by the rest of the application
- Implementing business rules, such as validation logic, calculated attributes, and defaulting logic
- Providing services that can perform large-scale batch operations on data upon request

Business services segregate the persistence and business logic of an application from the logic that governs the application's UI and control flow. Keeping persistence and business logic separate allows you to reuse them in multiple MVC applications.

Based on Standard Java and XML

ADF Business Components is a framework implemented in Java. Base framework classes provide generic, metadata-driven functionality. XML files store metadata that you define to configure each component's runtime behavior. You can also extend the base framework functionality to suit your needs.

Figure 1-132 shows the Applications window in JDeveloper and how it represents the files that comprise ADF Business Components services. For example, the `DeptVO` component is defined with a single XML file that relies entirely on underlying framework classes. On the other hand, the `CustomerVO` definition consists of an XML definition file that provides metadata and three Java classes that extend the underlying framework classes.

Figure 1-132 XML and Java Objects for ADF Business Components

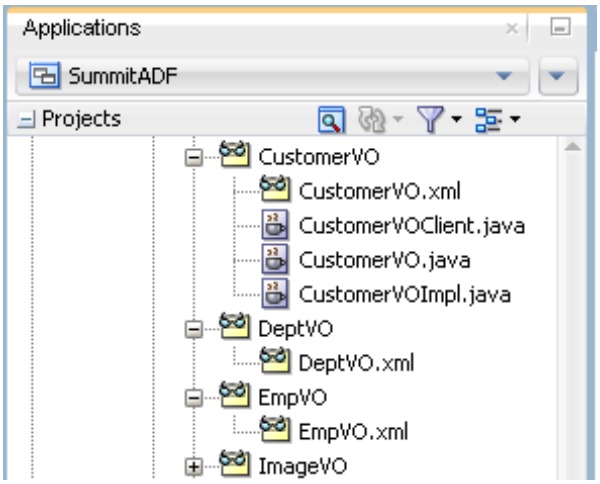


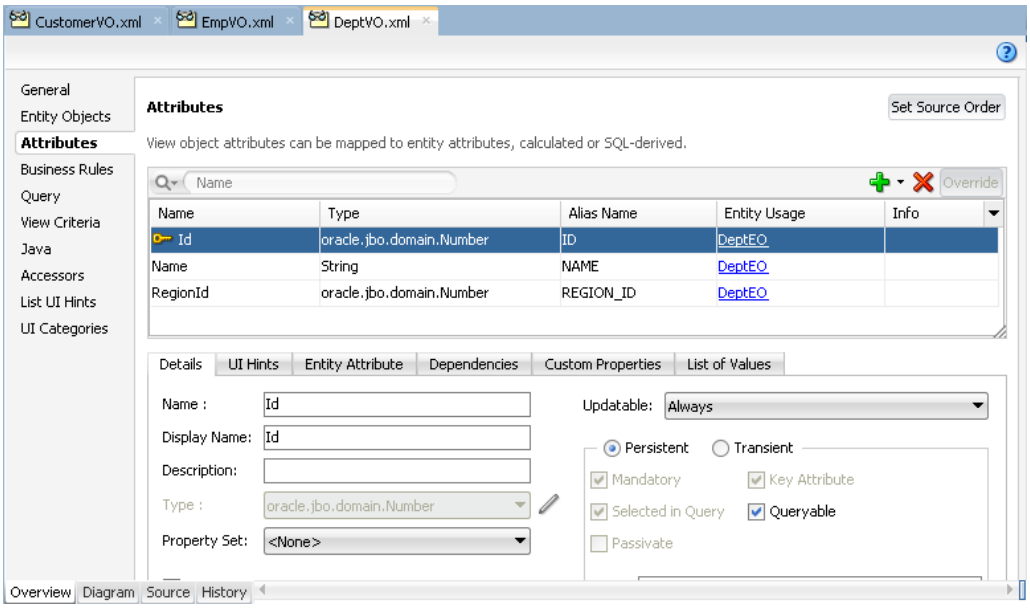
Figure 1-133 shows the source editor for an ADF Business Components view object definition file.

Figure 1-133 Source View for an ADF Business Components Definition File

```
DeptVO.xml
Find
  SelectList="DeptEO.ID,
       DeptEO.NAME,
       DeptEO.REGION_ID"
  FromList="S_DEPT DeptEO"
  BindingStyle="OracleName"
  CustomQuery="false"
  RowClass="oracle.summit.base.SummitViewRowImpl"
  ComponentClass="oracle.summit.base.SummitViewObjectImpl"
  PageIterMode="Full"
  UseGlueCode="false">
  <DesignTime> <Attr N ... ess"/> </DesignTime>
  <EntityUsage
    Name="DeptEO"
    Entity="oracle.summit.model.entities.DeptEO"/>
  <ViewAttribute
    Name="Id"
    IsNotNull="true"
    PrecisionRule="true"
    EntityAttrName="Id"
    EntityUsage="DeptEO"
    AliasName="ID"/>
  <ViewAttribute Name= ... " AliasName="NAME"/>
  <ViewAttribute Name= ... asName="REGION_ID"/>
  <ViewLinkAccessor Na ... Updateable="false"/>
</ViewObject>
```

JDeveloper also provides visual overview editors for ADF Business Components definition files. Figure 1-134 shows the overview editor for the same view object definition file shown in Figure 1-133.

Figure 1-134 Overview Editor for an ADF Business Components Definition File



Application Server and Database Independence

Because ADF Business Components services are implemented using plain Java classes and XML files, applications and services built using ADF Business Components can run on any Java-capable application server, including any Java EE-compliant application server. These applications and services can be run both within and outside of a Java EE server container.

You can use ADF Business Components components with both Oracle and non-Oracle databases. Numerous optimizations are built into ADF Business Components for use with Oracle databases.

Declarative Metadata for Implementation Classes

ADF Business Components objects are based on a set of Java classes that provide built-in runtime functionality that you control through declarative settings. You use an XML component definition file to specify metadata for things like object/relation mapping for database tables, data access methods, and validation rules. At runtime, the metadata is injected into the implementation classes to create instances of the services. For typical use cases, developers do not have to write any Java code to implement the services.

Optional Custom Java Code

It is possible to further configure the behavior of a component by adding custom Java code to the component's definition. When you need to write custom code for a component, for example to augment the component's behavior, you can enable an optional custom Java class for the component in question.

Ability to Expose Services to SOA Applications

After you have developed ADF Business Components services, you can publish them as external services that can be consumed by applications that are based on a service-oriented architecture (SOA). For more information, see Service-enabled Application Modules.

Application State Management

ADF Business Components has a state management facility for application modules that enables you to save the state of a user session, which simplifies recovery and failover scenarios.

For more information on application module state management, see Application State Management.

Key Components of ADF Business Components

The business service layer of the Fusion web application based on ADF Business Components is comprised of entity objects to model the data source (including support for object hierarchies, such as master-detail relationships) and view objects to shape the data for display to the end user through application-specific views. Other components include application modules which support CRUD (create, read, update, delete) transactions on specified view objects at application runtime.

The ADF Business Components architecture consists of the following key components:

- *Entity objects*, which encapsulate individual objects in a data source, such as tables in a database, and which add business logic for working with that data.
- *Entity associations*, which define the relationships between individual entity objects.
- *View objects*, which provide access to data in a form that can be used through ADF Model bindings in a user interface. View objects that allow updating of data are based on entity objects.
- *View links*, which define master-detail hierarchies between view objects.
- *Application modules*, which encapsulates the view objects needed for a logical unit of work related to an end-user task.

Entity Objects

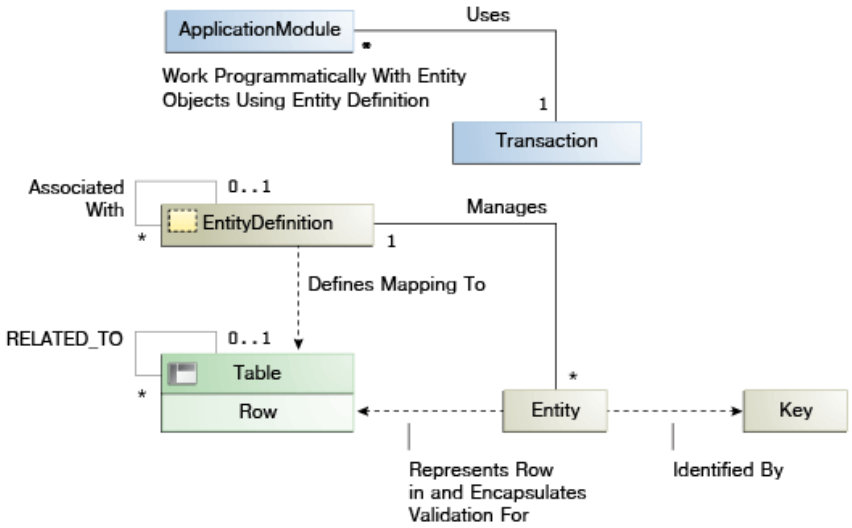
ADF entity objects are business components that encapsulate data, persistence behavior, and business rules for items that are used in your application. For example, entity objects can represent:

- Elements of the logical structure of the business, such as product lines, departments, sales, and regions
- Business documents, such as invoices, change orders, and service requests
- Physical items, such as warehouses, employees, and equipment

Entity objects map to single objects in the data source. In the vast majority of cases, these are tables, views, synonyms, or snapshots in a database. For example, you might create an entity object called `Departments` that represents a database table called `DEPARTMENTS`. Advanced programmers can base entity objects on objects from other data sources, such as spreadsheets, XML files, or flat text files.

Figure 1-135 shows how an entity object fits in with other objects in an ADF Business Components application.

Figure 1-135 Entity Object Within the ADF Business Component Architecture



Entity Object Definition Files

When you use JDeveloper's wizards and visual editors to create and configure an entity object, JDeveloper creates an XML file that contains the declarative metadata that defines the runtime behavior of that entity object, including its O/R mapping, validation rules, UI hints, and other metadata. At runtime, this metadata is injected into an instance of the generic framework class `oracle.jbo.server.EntityImpl`.

It is also possible to add custom functionality to an entity object by writing custom classes that extend ADF Business Components framework classes.

Ways to Configure Entity Objects

Entity objects are part of ADF Business Components implementation of ADF Model. As such, you can add declarative metadata to an entity object definition to configure its behavior. The following are some of the things for which you can set metadata on an entity object:

- UI hints, which are settings that the view layer can use to automatically display the queried information to the user in a consistent, locale-sensitive way.
- Validation rules, which you can set at both the level of entity objects or individual attributes.
- Business events, which you can use to launch business processes and trigger external systems synchronization.

Entity Associations

Relationships between entity object definitions are handled by entity associations, which define a relationship between two entity object definitions based on sets of entity attributes from each. Associations map to relationships between single objects in the

data source. In the vast majority of cases, these are relationships among tables, views, synonyms, and snapshots in a database. Advanced programmers can use associations to represent relationships within other data sources, such as spreadsheets, XML files, or flat text files.

When the data source is a database, associations often map to foreign key relationships between tables in the database. Although there does not need to be a foreign key constraint between tables for you to create a one-to-one or one-to-many association between the corresponding entity objects, there should be an appropriate logical relationship between the tables.

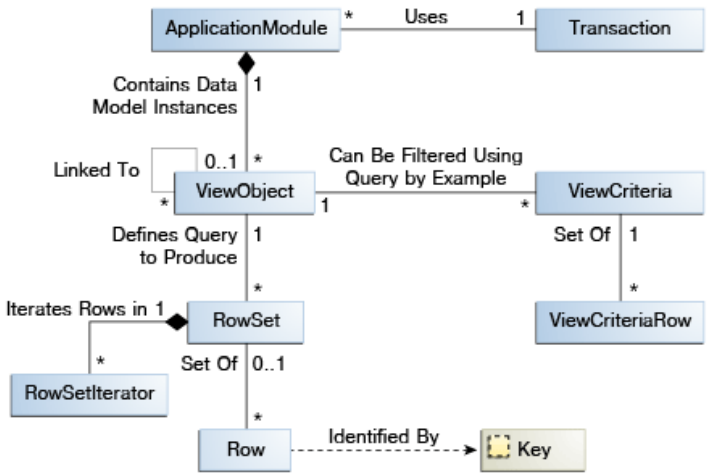
View Objects

ADF view objects are business components that collect data from the data source, shape that data for use by clients, and allow clients to change that data in the ADF Business Components cache. Among other things, a view object definition can gather the information needed to:

- Populate a single table element in a form
- Create and process an insert or edit form
- Create a list of values for populating a dropdown list
- Create a search form with specific search criteria

Once you have created a view object definition and included it in the data model of an application module, you use the Data Controls panel to create UI components based on the collections, attributes, and operations of that view object.

Figure 1-136 View Object Within the ADF Business Component Architecture



View object definitions must have a mechanism for retrieving data from the data source. Usually, the data source is a database, and the mechanism is a SQL query. ADF Business Components can automatically use JDBC to pass a query to the database and receive the result. When view object definitions use a SQL query, query columns map to view attributes in the view object definition. The definitions of these attributes reflect the properties of these columns, such as the columns' data types and precision and scale specifications. When view object definitions use other data sources, view object attributes map to "columns" of data from those data sources, as defined by the programmer.

Typically, when you work with a view object, you work with only a single row set of results at a time. To simplify this use case, the view object contains a default `RowSet`, which, in turn, contains a default `RowSetIterator`. The default `RowSetIterator` allows you to call all of the data-retrieval methods directly on the `ViewObject` component itself, knowing that they will apply automatically to its default row set.

In addition, you can declaratively define view criteria for a view object. With a view criteria, you specify query conditions that augment the `WHERE` clause of the target view object in order to filter the results. You can then use those view criteria to create Query-by-Example search forms, filter row sets or lists-of-values (LOVs) at runtime, or create varying view instances based on a single view object definition.

Types of View Objects

There are two main types of view objects:

- Entity-based view objects, which access data from one or more entity objects and coordinate with those entity objects to update the data source based on user actions.
- Read-only view objects, which have direct access to the data. Because read-only view objects do not require intermediary objects, they access data more quickly than entity-based view objects. Create read-only view objects if you have use cases where you need to access data without modifying it. You might have a read-only view object and an entity-based view object for the same table.

In addition, you can create view objects with other data sources such as:

- Direct SQL queries of the database
- Programmatic sources
- Static data from CSV files

You can also create polymorphic view objects, in which multiple row set types with a common inheritance hierarchy are represented in a single view object.

View Object Definition Files

Similar to working with entity objects, when you use JDeveloper's wizards and visual editors to create and configure a view object definition, JDeveloper creates an XML file that contains the declarative metadata that defines the runtime behavior of that view object and features that are used in the UI, such as UI hints and validation rules. At runtime, this metadata is injected into an instance of the generic framework class `oracle.jbo.server.ViewObjectImpl`.

It is also possible to add custom functionality to a view object by writing custom classes that extend ADF Business Components framework classes.

Ways to Configure View Objects

View objects are part of ADF Business Components implementation of ADF Model. As such, you can add declarative metadata to a view object definition to configure its behavior.

You can define the same declarative metadata for a view object as you can for an entity object (with the exception that you cannot raise business events in view objects). In addition, you can set other types of metadata for a view object, such as the following:

- View criteria, which function as further refined queries and which are represented in the Data Controls panel as named queries, from which you can declaratively create search forms.
- List UI hints, which can be used to guide how lists of values are presented in the user interface.
- UI categories, which can be used for presenting titled groups of attributes in dynamic forms.
- View accessors, which can be used to provide a data source for view instance attributes involved in either list-based attribute validation or lists of values.
- Row finders, which can be used to match view instance rows by non-key attribute values and to initiate row updates either programmatically or through ADF web services.

View Links

Relationships between view objects are handled by view links, which define a relationship between two view objects based on sets of entity attributes from each. Like entity associations, these can range from simple one-to-many relationships based on foreign keys to complex many-to-many relationships.

Individual instances of view objects can also be related by individual instances of view links, which create a master-detail relationship between the query result sets. For example, suppose that you have view object definitions representing a query for department information and a query for employee information, and a view link between the view objects representing the relationship between a department and its employees. If an instance of the former view object definition, `allDepartments`, is related to an instance of the latter, `employeesInDepartment`, by an instance of the view link, those instances will be synchronized: whenever a particular row of `allDepartments` is selected, `employeesInDepartment` will only display details of that row.

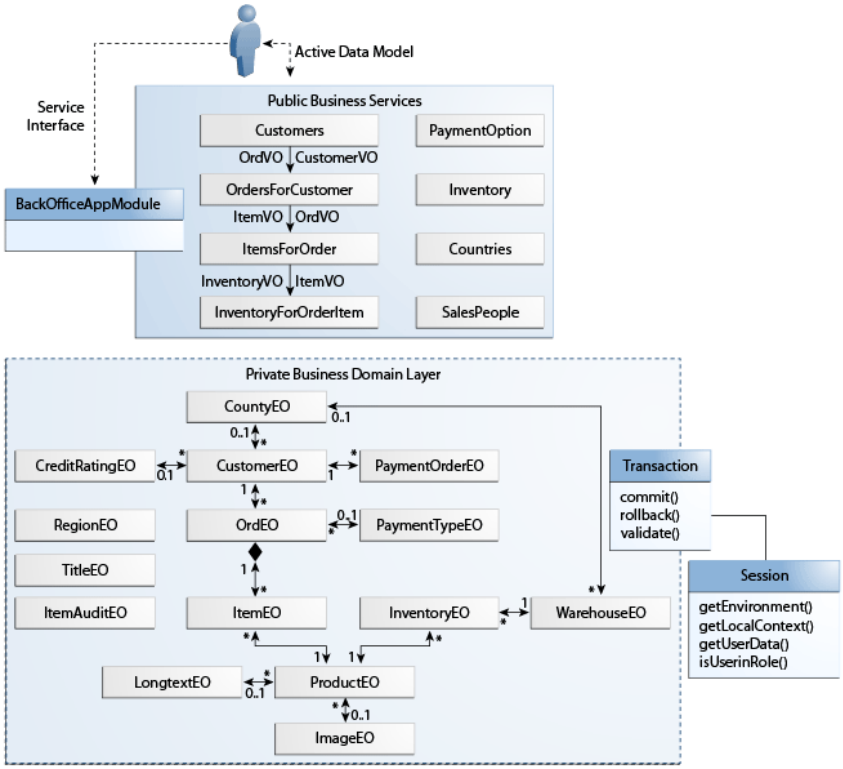
Application Modules

Oracle ADF application modules are the ADF Business Components implementation of ADF Model data controls. Application modules represent particular application tasks. The application module definition provides a data model for the task by aggregating the view object and view link instances required for the task. It also provides services that help the client accomplish the task. For example, an application module can represent and assist with tasks such as:

- Updating customer information
- Creating a new order
- Processing salary increases

Figure 1-137 illustrates how an application module works with other business components.

Figure 1-137 Application Module Within the ADF Business Component Architecture



In addition, application modules have pooling and state management features that simplify making applications scalable, well-performing, and able to handle failover.

Types of Application Modules

You can use application module definitions in the following ways:

- As a service object, in which case each instance of the MVC application has access to one instance of the application module. These root-level application module instances control ADF Business Components transaction objects, which in turn control the entity and view caches.
- As a reusable object for nesting, in which case you can create a data model and service methods on it and then nest one of its instances in other application module definitions. Those application module definitions can, in turn, access the nested module's methods and data model. Nested application modules share the root-level application module's transaction
- As a shared application module, in which data is cached for reuse across sessions and requests. Shared application modules are particularly useful for optimizing performance when you have data that does not change very frequently and needs to be accessed across multiple sessions and requests.

Application Module Definition Files

An application module definition can have one or two parts:

- An XML file, which represents the portion of the application that can be developed declaratively: the view object and view link instances that the application module contains and the way in which they are related. For many application modules, the XML file by itself is sufficient.
- An application module class, which lets you write custom code such as service methods that an MVC application can invoke for batch data handling. Application module classes extend the class `oracle.jbo.server.ApplicationModuleImpl`. If you do not need to write custom service methods, you need not generate an application module class—ADF can use `oracle.jbo.server.ApplicationModuleImpl` directly.

Service-enabled Application Modules

Service-enabled application modules are ADF application modules that you advertise through a service interface to service consumers. There are three scenarios for service consumers to consume a published service-enabled application module:

- web service access
- Service Component Architecture (SCA) composite access
- access by another ADF application module

The Service Component Architecture (SCA) provides an open, technology-neutral model for implementing remotable services that are defined in terms of business functionality and that make middleware functions more accessible to application developers. ADF Business Components supports an SCA-compliant solution through application modules you can publish with a service interface. The service interface is described for Fusion web application clients in a language-neutral way by the combination of WSDL and XSD.

When you service-enable your application module, JDeveloper generates the artifacts, which comprise the following files:

- the Java interface defining the service
- an EJB session bean that implements this Java interface
- a WSDL file that describes the service's operations
- an XML Schema Document (XSD) that defines the service's data structures

SCA defines two kinds of service:

- Remotable services, typically coarse-grained and designed to be published remotely in a loosely coupled SOA architecture
- Local services, typically fine-grained and designed to be used locally by other implementations that are deployed concurrently in a tightly coupled architecture

ADF Business Components services fall into the first category, and should only be used as remotable services.

ADF Business Components services, including data access and method calls, defined by the remote application modules are interoperable with any other application module. This means the same application module can support interactive web user interfaces using ADF data controls and web service clients.

Any development team can publish a service-enabled application module to contribute to the Fusion web application. The Fusion web application assembled from remote services also does not require the participating services to run on a single application server.

Although the web applications may run on separate application servers, the appearance that SCA provides is one of a unified application. Consuming client projects use the ADF service factory lookup mechanism to access the data and any business methods encapsulated by the service-enabled application module. At runtime, the calling client and the ADF web service may or may not participate in the same transaction, depending on the protocol used to invoke the service (either SOAP or RMI). Only the RMI protocol and a Java Transaction API (JTA) managed transaction support the option to call the service in the same transaction as the calling client. By default, to support the RMI protocol, the ADF web service is configured to participate in the same transaction.

Application Module Pooling

Applications you build that leverage an application module as their business service take advantage of an automatic application module pooling feature. This facility manages a configurable set of application module instances that grows and shrinks as the end-user load on your application changes. Due to the natural "think time" inherent in the end user's interaction with your application user interface, the number of application module instances in the pool can be smaller than the overall number of active users using the system. As a given end user visits multiple pages in your application to accomplish a logical task, an application module instance in the pool is acquired automatically from the pool for the lifetime of each request. At the end of the request, the instance is automatically returned to the pool for use by another user session.

To optimize your application's performance, you can tune application module pooling properties, such as initial and maximum pool size and the amount of time application module instances must be inactive before they can be removed from the pool.

Application State Management

You can use application module components to implement completely stateless applications or to support a unit of work that spans multiple browser pages. An application module supports *passivating* (storing) its pending transaction state to an XML document, which is stored in the database in a single, generic table, keyed by a unique passivation snapshot ID. It also supports the reverse operation of activating pending transaction state from one of these saved XML snapshots. This passivation and activation is performed automatically by the application module pool when needed. Activation can be triggered by server failover or simply because a user session spans multiple instances in the application module pool before it is completed.

Overview of the ADF Business Components Process Flow

Modelling a data source to create the business service layer of the Fusion web application that you develop with Oracle ADF Business Components follows a step-by-step process that is supported by JDeveloper tools.

Creating a business service layer based on ADF Business Components consists of the following general steps:

1. In JDeveloper, create an application workspace for the application.
2. Create custom classes that extend the base framework classes and then configure the model project to base any business components that you create on these custom classes. These classes provide a mechanism to later change base

framework behavior and have those changes apply to all of the business components you have created in the application.

3. Using wizards in JDeveloper's New Gallery, create a combination of the following objects:
   - Entity objects
   - Entity associations
   - View objects based on the entity objects
   - Optionally, view objects based on queries directly to the database
   - View links between view objects to establish master-detail relationships
   - Create application modules and include the appropriate view objects and view links within them to establish your data model
4. Optionally, use JDeveloper's visual editors to declaratively specify business rules for the entity objects and view objects.
5. Use the ADF Model Tester to test the data model's business logic.
6. Tune the application modules for performance.
7. If participating in a SOA application, publish the services so that they can be consumed by an external application.
8. Using the Data Controls panel and various binding editors, create databound components in the view layer.