

Oracle® Policy Automation Developer's Guide

User Assistance and How-To Guide

Browse the Policy Automation Developer's Guide	
What's new	Getting started
Web services	Interviews
Custom development	Extensions
Batch Processor	Language support
Project interchange	Decision reports
Performance	Security
Technical reference	Tutorials and examples

V10.4.6

Copyright © 2009, 2015

Oracle Support

Oracle® Policy Automation Developer's Guide

Release 10.4.6

E64012-01

May 2015

Copyright © 2009, 2015, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate failsafe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

This software or hardware and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

What's new in Oracle Policy Automation V10

Version 10.4

BI Publisher

The version of BI Publisher used by Oracle Policy Automation has been upgraded from v.11.1.1.3 to v.11.1.1.7 (as of release 10.4.5).

Web Determinations

Combo box (filtered dropdown list) search functionality is unsupported in Opera browsers and hence disabled.

Batch Processor

The Batch Processor replaces the Data Source Connector. It allows a large number of cases to be processed in batch and is available in both Java and .NET implementations to enable support for platform specific custom functions. It uses a variety of techniques to maximize throughput for both CSV and database connections. Full details regarding the use of the Batch Processor can be found in the [Batch Processor](#) section. Note that this includes a topic on migrating a Data Source Connector project to a Batch Processor project.

Determinations Server

The assess service will now only return relevant attributes in the decision report for an unknown attribute. For backwards compatibility, all assess services prior to 10.4 will continue to include all related attributes in the decision report. The interview service is not affected by this change.

Three previously disparate topics have been combined to form a single topic [Get an answer from the Determinations Server Assess Request](#). The topics to be removed are: *Process multiple determinations within the same Assess request*, *Get answers from a web service* and *Obtain a decision report from a web service*. Additionally, a new section has been included to describe the use of the Outcome option.

Interview Portlet

A portlet is a pluggable website component that is managed and displayed in a web portal; the Interview Portlet allows interviews to be conducted in such a web portal. Full details regarding customizing and using the Interview Portlet can be found in the [Customize the Interview Portlet](#) section.

Interview Engine API

Some APIs for the Interview Engine have been deprecated. Details can be found in the [Technical Reference](#) section.

Removal of Code Examples

In most instances, code snippets have been removed from the documentation and replaced by references to where the appropriate source code and compiled samples can be found in the Oracle Policy Automation Java or .NET runtime zip file.

Rulebase Listener

Rulebase listeners are custom objects that are created when a rulebase is loaded, and called every time a new session is created. Developers can create a rulebase listener to perform early initialization of new sessions before they are returned to the calling application.

Details can be found in the topics [Initialize a new session with a Rulebase Listener](#), [Rulebase Listeners](#) (in the *Technical Reference*) and [Example: Create a Rulebase Listener to preload reference data](#) (in *Tutorials and Examples*)

Runtime - general information

A new topic has been added as general runtime information relating to how large decimal numbers are handled. For more information, see [Handling of large decimal numbers](#).

Security

Microsoft Team Foundation Server has been tested to work with Oracle Policy Modeling. See *Install Microsoft Team Foundation Server* in the *Oracle Policy Modeling User Guide* for more information. See also, [Secure particular rule sets](#).

Extensions

Rulebase Resolver Plugin

The mechanism used to load and store rulebases and consequently its extension point have undergone a number of changes in 10.4. Prior to this version those wishing to change the default rulebase loading behavior would have to implement the RulebaseService. In doing so, however, this would make them responsible for not only the rulebase sourcing and loading behavior, but the creation of the InterviewRulebase object, the rulebase caching behavior, as well as any dynamic updating behavior.

Since the introduction of modules in this release has necessitated a number of changes in the way rulebases are loaded, created and maintained, it was decided to simplify the extension point such that now people wishing to customize the loading behavior need only implement a method of retrieving rulebase streams. In order to achieve this, the RulebaseService is no longer available as a plugin, instead the loading behavior is customized via the [RulebaseResolverPlugin](#).

Report error messages from plugins

A new topic, [Report error messages from plugins](#) has been added to the extensions section, describing the localization of error messages, the passing of parameters to localized error messages and how to report errors from the various plugins in the Interview Engine and Web Determinations.

New Examples

A number of new examples have been created to assist users with Oracle Policy Automation's features:

- [Add and remove buttons on Entity Collect screens](#)
- [Create a Custom Screen for the Interview Portlet](#)
- [Create a Custom Validator for control validation](#)
- [Create a Custom Validator for screen validation](#)
- [Create a Rulebase Listener to preload reference data](#)
- [Create test cases in the Batch Processor](#)
- [Custom Control – CalendarDateControl Walkthrough Example](#)

- Custom Control - Filtered Dropdown Selection List Walkthrough Example
- Dynamically display an error message for a control
- Encode the Interview Portlet's response
- Extract the username in the Interview Portlet
- Hide eligibility criteria from a decision report depending on benefit applied for
- Pass in parameters for an Interview Portlet across WSRP
- Rulebase Resolver - Sample Code (DerbyRulebaseService) - this replaces *Rulebase Service - Sample Code (DerbyRulebaseService)*
- Run the Batch Processor (InsuranceFraudScore)
- Send external events to other portlets
- Show or Hide an Attribute
- Use the Batch Processor with a Database
- Use the OnInterviewSessionCreatedEvent to pre-seed data into a newly created session

The following no longer appropriate examples, have been removed:

- *Create a mobile device application using Oracle Determinations Engine and Microsoft .NET Compact Framework*
- *Handle rulebase events with an inferencing event listener*
- *Infer entity instances with an inferencing listener*

Version 10.3

Document Generation

The default XSLT/ FOP Document Generation Plugin has been replaced with the BIPublisherDocumentGenerator which uses Oracle Business Intelligence Publisher (BI Publisher) as the basis for its document generation.

Support for BI Publisher has necessitated the following API changes to the Oracle Determinations Interview Engine:

- The DocumentGeneratorPlugin interface has been altered, which will require any third party document generation plugins written prior to this release to be updated accordingly.
- The DocumentGenerationParameters class has been replaced with the DocumentTemplate class which encapsulates all the relevant parameters required to generate a particular document.
- It is no longer possible to register multiple instances of DocumentGeneratorPlugin per session. Instead, third parties should implement a single DocumentGeneratorPlugin which then delegates to different document generation solutions as required.
- The XSLT/FOP based plugins have been removed from the API but have been made available as libraries that can be added as optional plugin to support backwards compatibility. For more information, see [Legacy Document Generation](#).
- Added GoalControlTemplate and DocumentControlTemplate classes to represent Goal and Document controls respectively.

See also:

[Document Generator Plugin Overview](#)

[Use the Default Document Generator](#)

[Legacy Document Generation](#)

Version 10.2

Oracle Determinations Server

The Determinations Server has been completely rewritten to follow a new architecture. This allows the existing functionality provided by the Determinations Server to be extended. It also allows custom services to be written and installed in the Determinations Server.

New WSDL for Assess Service

There is a new version of the WSDL for assess requests and responses. Assess requests and responses now have a hierarchical structure that follows the hierarchical data model provided by Entity Containment. Full backwards compatibility is provided for WSDL used in OPA 10.0 and 10.1.

Interview Service

Version 10.2.0 also includes a new Interview Service. The Interview Service is a web service that can be used by custom applications that conduct interviews. It is provided as an alternative to using the Interview Engine Java or .NET APIs directly.

Version 10.1

Oracle Determinations Engine

The Oracle Determinations Engine has been enhanced with new data types and reasoning features. These features are available throughout the product suite from Oracle Policy Modeling to Oracle Determinations Server and Oracle Web Determinations.

Prevent custom functions making session data changes

It was previously possible (but officially discouraged) to change the session inside a custom function handler or in an inferencing listener after a rulebase event is raised. Since this can cause instability in the engine (at worst) or inconsistent behaviour (at best), this is now explicitly prevented and an error occurs if the custom function handler or inferencing listener attempt to make direct changes to the session in the middle of an inferencing cycle.

Inferencing Listener enhancements

Inferencing listeners are allowed to make changes before or after an inferencing cycle, and this is the recommended alternative. If an inferencing listener needs to make changes in response to a rulebase event, it should set some internal flag and wait until the end of the inferencing cycle to make those changes. The engine will automatically trigger another inferencing cycle if those changes cause other rules to trigger.

See also:

[Customize the inferencing cycle with custom functions and inferencing listeners](#)

[Rulebase Configuration File](#)

Oracle Web Determinations

Oracle Web Determinations has been completely re-architected, and provides improved extensibility, embedability, international support and configurability.

Entity Containment

In Oracle Policy Automation V10.0 the concept of an entity being *collected* was introduced. When an entity is considered collected, the rule engine assumes that it knows the entire set of instances for that entity.

An entity's collection status (whether or not it is considered collected) is of major importance when determining whether or not a relationship is partially known (Partially known relationships).

Under Oracle Policy Automation 10.0 the user was able to directly set the collection status of an entity. However in Oracle Policy Automation 10.1 the collection status of an entity is now determined by the engine through the use of *containment relationships*. A containment relationship is a one-to-many relationship from a parent entity to a child entity. An entity Y is considered to be collected if:

1. A one-to-many containment relationship is defined from some other entity X to entity Y (i.e. *Y is contained by X*) and this is referred to as the *entity Y's containment relationship*
2. Entity Y's containment relationship is set (i.e. it is known) for all instances of entity X
3. Entity X is also considered to be collected.

Note:

- The global entity is always automatically collected. It is not necessary (or possible) to create a containment relationship for the global entity.
- Containment relationships are not supported for singleton entities as they are deprecated. Singleton entities are never considered by the rule engine to be collected.

Rulebase Service Plug-in

An Oracle Web Determinations implementation can now handle more than one rulebase. When there are two or more rulebases, the user is given the choice to select which rulebase to run. Therefore, an Oracle Web Determinations implementation can now list the available rulebases that the user can access, and retrieve a specific rulebase when the user selects one to run. These and other internal rulebase functions are provided by a 'Rulebase Service' object in the Oracle Web Determinations.

The default Oracle Web Determinations Rulebase Service retrieves rulebases as zip files from a pre-defined location/path in the Oracle Web Determinations webapp; for example, for a default installation, the path <OWD webapp>/WEB-INF/classes/rulebases, is read and monitored for rulebases. The user can store/manage the rulebase zip files in the path, and also access them to list or load a specific rulebase.

There are situations where rulebases of an Oracle Web Determinations implementation need to be stored and accessed from a custom datasource. The Rulebase Service Plugin is an Interview Engine plugin that allows usage of custom datasources to store and retrieve rulebases from. Also it allows the implementer to customize the rulebase service functionality.

Localization

The default Oracle Web Determinations user interface is now provided in every one of the 24 languages for which a syntactic or non-syntactic parser is available.

Version 10.0

[Change the appearance of a web interview](#)

- [Change the behavior of a web interview control](#)
- [Save web interview results](#)
- [Customize resource loading for Web Determinations](#)
- [Add user authentication to Web Determinations](#)

For this release of the Developer's Guide, it may help to be aware of the following product mapping between the Oracle naming and the former Haley naming:

Current Oracle Naming

- Oracle Policy Automation
- Oracle Policy Modeling
- Oracle Policy Automation
- Oracle Determinations Server
- Oracle Determinations Engine
- Oracle Web Determinations

Former Haley Naming

- Haley Office Rules product suite
- Haley Office Rules
- Haley Determination Services
- Haley Determinations Server
- Haley Determinations Engine
- Haley Interactive

Getting started

What do you want to do?

Understand the different components of Oracle Policy Automation

Find the list of platforms supported by Oracle Policy Automation

Design a solution using Oracle Policy Automation components

Upgrade the interview experience from a previous version

Access further resources on Oracle Policy Automation

Understand the different components of Oracle Policy Automation

Oracle Policy Automation comprises five main components:

- Oracle Determinations Engine
- Oracle Determinations Server
- Oracle Web Determinations
- Oracle Interview Portlet
- Oracle Web Determinations Interview Engine

Find the list of platforms supported by Oracle Policy Automation

The following is a list of all platforms currently supported by Oracle Policy Automation:

Java

Runtime

- Java 1.5+

Web Application Servers

- Apache Tomcat 5.5+

.NET

Runtime

- .NET 2.0+
- Visual J# 2.0+

Web Application Servers

- IIS 6.0+

Web Browsers

- Internet Explorer 6.02
- Internet Explorer 7
- Internet Explorer 8
- FireFox 2.0
- FireFox 3.0
- Opera 9.0+
- Safari 2.0+
- Chrome 1.0+

Note: Combo box (filtered dropdown list) search functionality is unsupported in Opera browsers and hence disabled.

Design a solution using Oracle Policy Automation components

When choosing how you want to deploy your rulebase you need to think about how you will be using it. Oracle Policy Automation has different deployment options to suit different uses:

- If you want to create a web-based Web Determinations application you should consider using Oracle Web Determinations.
- If you have a single application that integrates rules in an existing or new application, or you want to use a GUI application written in Java or C# you should consider using the Determinations API directly.
- If you want to manage determinations from a central location and use them as a service in a distributed application, or from many different applications you should consider using the Oracle Determinations Server.

For more a more detailed explanation on whether to use the Determinations Engine API or use the Determinations Server, see the topic, [Embed determinations inside another application](#).

Note: The various API Reference zip files are installed with the application at the following locations:

C:\Program Files\Oracle\Policy Modeling\help\api\determinations-engine-doc.zip

C:\Program Files\Oracle\Policy Modeling\help\api\determinations-engine-net.zip

C:\Program Files\Oracle\Policy Modeling\help\api\masquerade-net.zip

C:\Program Files\Oracle\Policy Modeling\help\api\web-determinations-doc.zip

Also note that the default installation location "C:\Program Files\Oracle\Policy Modeling" may be changed by the user.

Understand the Oracle Determinations Engine

The Oracle Determinations Engine provides high-performance rule execution that can be easily embedded into Java and .NET applications, giving software developers full and direct access to its underlying functions. Organizations can use the Determinations Engine to embed powerful rules-based inferencing directly into their enterprise applications, providing an important alternative to other Oracle deployment offerings.

High performance inferencing

The Oracle Determinations Engine exposes the full power of Oracle's patented Linear Inferencing algorithm, supporting the development of ultra-fast, rules based batch processing applications.

By directly accessing the Determinations Engine, you can maximize the speed at which large data sets can be processed; for example, recalculating insurance premiums across a customer base to deal with a change in policy.

Benchmarks confirm that the Determinations Engine can process millions of records in a matter of hours using cheap, commodity hardware. Excellent scalability enables higher processing speeds with more powerful hardware.

Web Determinations application development

The Oracle Policy Automation Determinations Engine provides full support for the development of custom Web Determinations rules based applications, enabling organizations to create specialized user interfaces or user interfaces that seamlessly integrate with existing enterprise applications.

Oracle's own out-of-the-box user interface, Oracle Policy Automation Web Determinations, is implemented using the Engine.

The Oracle Policy Automation Determinations Engine is the foundation of Oracle's deployment technology, providing all basic services for executing rules based applications across the following four main areas:

Inferencing

- Loading and saving data.
- Application of data to rules to draw conclusions (forward chaining).
- Determining what data is required to draw a specific conclusion (backward chaining).

Metadata

- Retrieving information about the structure of a rule set.
- Retrieving information about objects in the rule set.
- Setting and retrieving application-specific data.

Natural language

- Generating sentence text for data items.
- Personalizing sentence text through data value and pronoun substitution.

Screen handling

- Retrieving and interrogating screen.

The Engine exposes its functionality through a well-defined application programming interface which is available in both Java and .NET versions, allowing it to be used with J2EE server platforms such as WebLogic, WebSphere and Oracle AS, in addition to Microsoft's standard platform; that is, COM+, ASP.NET, IIS.

The Engine is multithreaded, runs in-process and is able to work with as many rule sets as are required. It can simultaneously process multiple concurrent requests across multiple rule sets. It also scales well from single computer installations such as a standalone laptop all the way up to large production environments comprising clustered multiprocessing server farms.

Programming examples

Follow the links below to find reference examples for software engineers using the Determinations Engine API. It is written specifically from a Java perspective and includes code samples given in both Java and C#. Software engineers using the Determinations Engine API for .NET should also read the guide to using the Determinations Engine API for .NET.

A note concerning rounding:

In the Oracle Policy Automation Determinations Engine, numbers and currency are represented by floating point numbers (double) to perform calculations. Because of the way floating point numbers are represented in Java and .NET there can be some differences in calculations.

For example, a number may be calculated as 8.14571428571429 when you test in Oracle Policy Modeling, but the Java Determinations Engine might return 8.145714285714286. These differences will always be extremely small.

A client using the Oracle Policy Automation Determinations Server or Engine can overcome these differences by rounding to some significant value.

The following examples are provided:

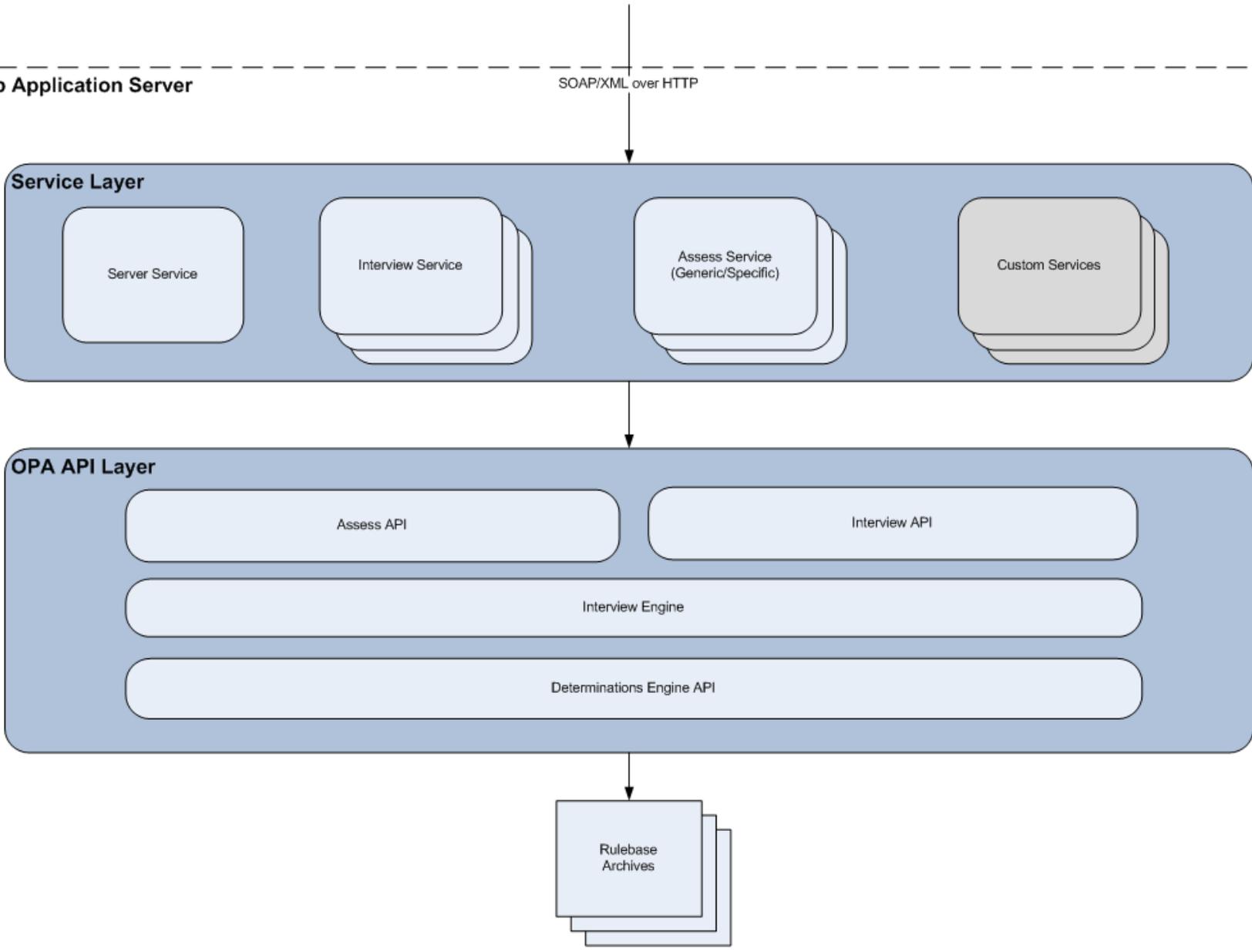
[Retrieve a Decision Report](#)

[Rulebase configuration file](#)

Understand the Oracle Determinations Server

Oracle Determinations Server is a web service that provides the ability for remote client applications to send assessment data, perform inferencing based on the chosen rulebase, and return the outcomes of such inferencing back to the requesting client. It is built upon the Oracle Determinations Engine, the core of Oracle Policy Automation, and provides a simple-to-use interface via the industry standard, XML based SOAP protocol.

See diagram



The above diagram shows the components typically required to run the Oracle Determinations Server. These are:

Rulebase Archives:

These are the deployed rulebases. They include, not just the rulebase itself, but also language and screen files, and possibly custom functions. Rulebase Archive files are a compressed (zip) archive of several files and folders.

Server Service:

The Oracle Determinations Server provides a single Web service for such things as checking the Determinations Server version, listing the deployed rulebases and reloading a changed rulebase.

Generic/Specific Assess Service:

For every rulebase deployed, a Generic and a Specific Web Assess Service is provided. Each of these web services allow assessments to be performed using the related rulebase – data is input, inferencing is completed, and the outcomes of the assessment are returned. These Web Services The Generic and Specific Assess Services each provide the same functions, but vary in the way they are called.

Interview Service:

For every rulebase deployed, an Interview Service is provided. These are stateful web services that can be used to conduct interviews using a rulebase.

Custom Services:

Custom services can be written and deployed to the Determinations Server using the plugin architecture provided. An instance of a custom service will be created for each rulebase that has been deployed to the Determinations Server.

OPA API Layer:

The OPA API Layer consists of the various proprietary APIs upon which the Determinations Server is built. These APIs can be utilized when writing custom services.

Web Application Server:

This is the web application server that is capable of communicating via SOAP, and on which the Determinations Server runs. Examples include IIS for the .NET Determinations Server, or Apache Tomcat or the WebSphere Application Server for the Java Determinations Server.

Note: The Determinations Server for Java relies on the **determinations-server.war** file that contains the web services interface to the Oracle Determinations Engine.

What services does Oracle Determinations Server provide?

The Oracle Determinations Server is a way of utilizing one or more rulebases via HTTP protocols. It is a WS-I compliant Web Service using SOAP-formatted XML envelopes. Its servers are described by a WSDL.

When a rulebase is deployed to Oracle Determinations Server, a series of Web Services are created for that rulebase, each with its own WSDLs and Service End Points.

Oracle Determinations Server provides services split into two distinct areas:

- [Oracle Determinations Service](#)
- [Rulebase Web Services](#)

Oracle Determinations Server is implemented for two frameworks: Java and .NET.

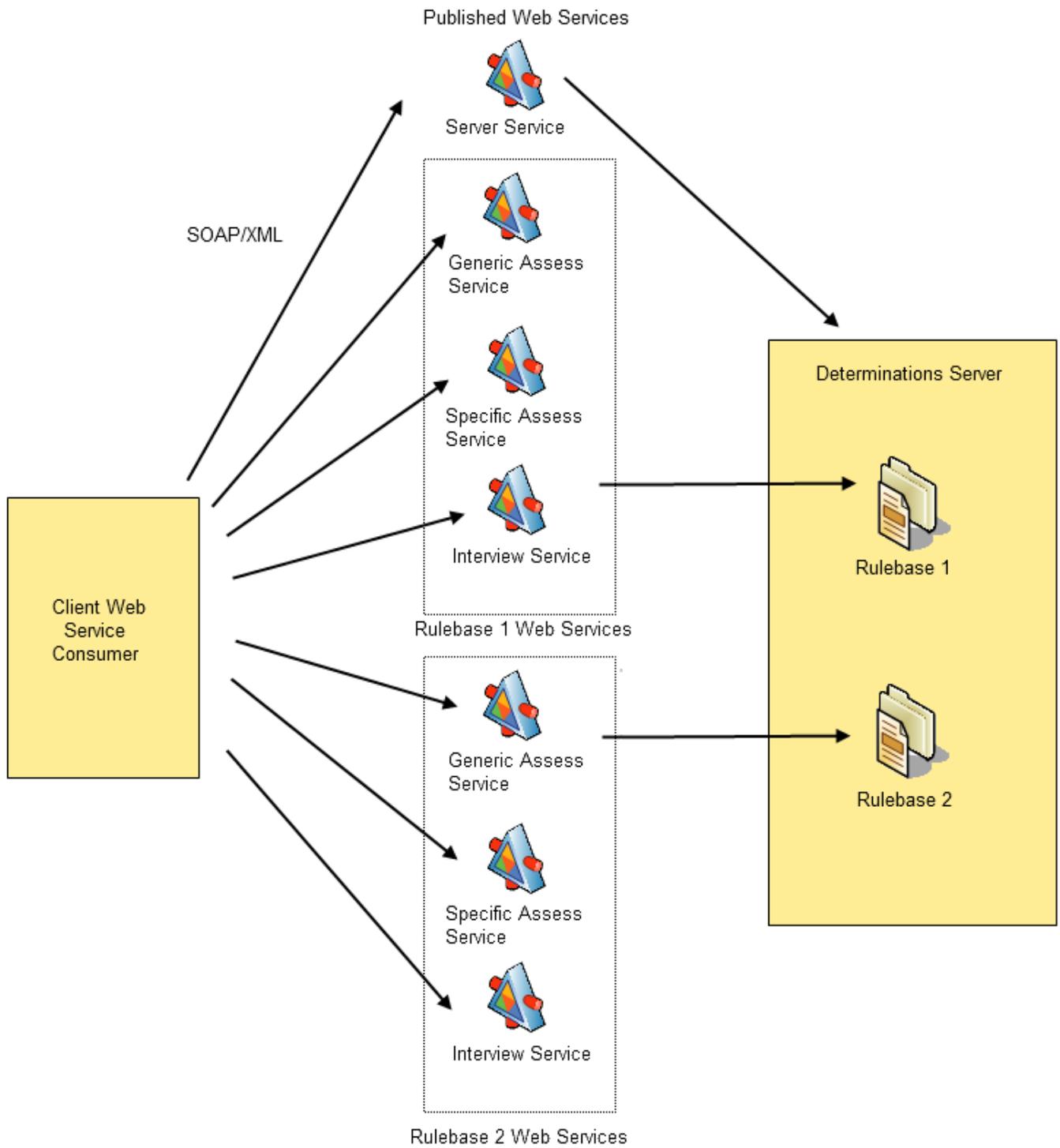
Oracle Determinations Server for Java

Oracle Determinations Server for Java is a standard web application file (.war), and can be run on J2EE Application servers such as Apache Tomcat, BEA WebLogic or IBM Web Sphere.

Oracle Determinations Server for .NET

The Oracle Determinations Server for .NET can be run on Microsoft's Internet Information Server (IIS).

[See diagram](#)



Oracle Determinations Server Directory Structure

The Oracle Determinations Server directories into which messages and rulebase archives are placed (see the diagram below), each of which is described below in the following topics:

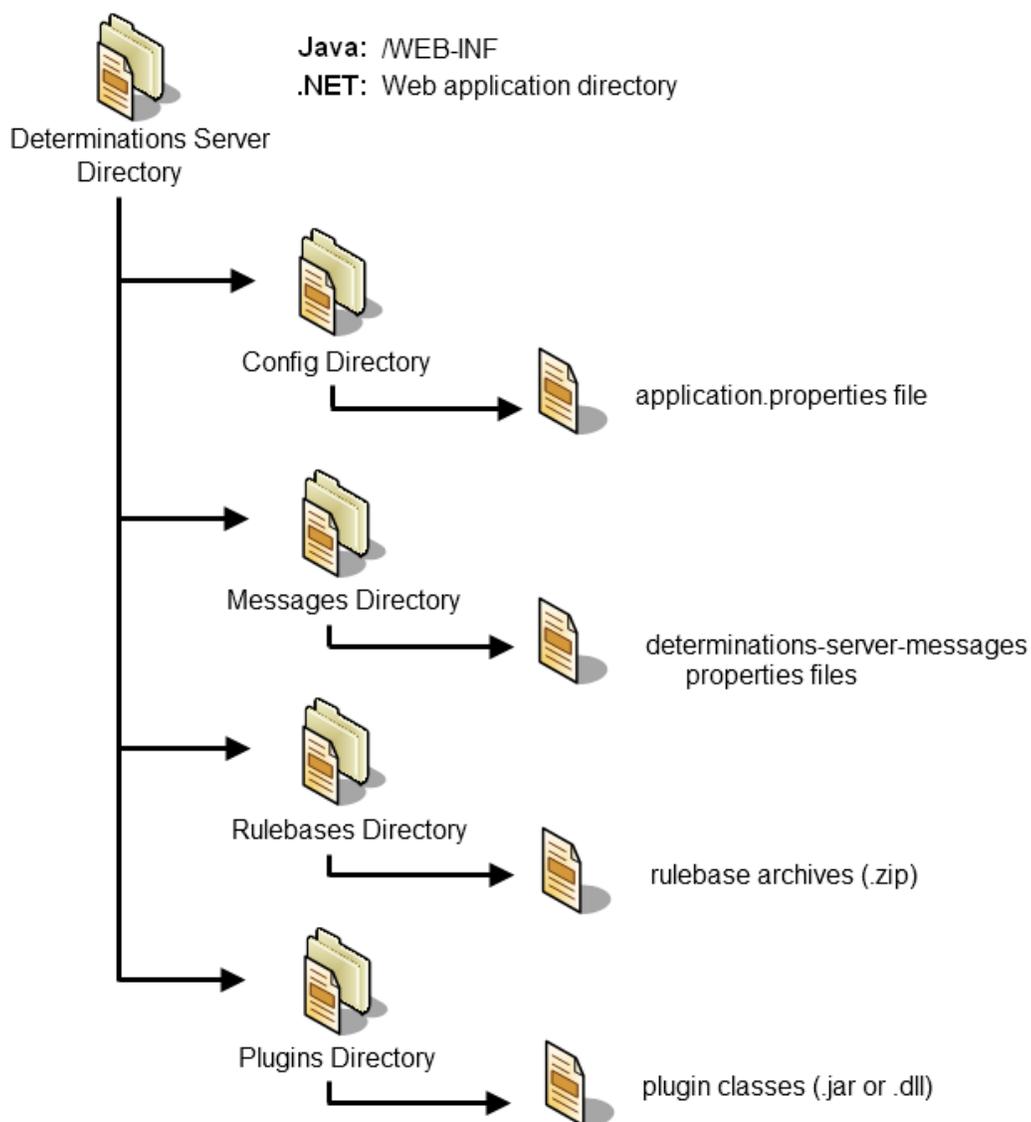
[The Config directory](#)

[The Messages directory](#)

[The Rulebases directory](#)

[The Plugins directory](#)

See diagram



The Config directory

The Config directory contains the applications.properties file which can be edited in order to update the configuration of the Determinations Server.

For Java:

The Config directory can be found in the following location:

[determinations-server/WEB-INF/classes/config](#)

For .NET:

The Config directory defaults to a directory named 'config' in the Web Application directory.

See also:

[Oracle Determinations Server configuration file](#)

The Messages directory

This is where the language specific messages properties files are placed. By default the Oracle Determinations Server only comes with English language configuration (default), but different languages can be supported.

For Java:

The Messages directory can be found in the following location:

[determinations-server/WEB-INF/classes/messages](#)

For .NET:

The Messages directory defaults to a directory named 'messages' in the Web Application directory.

See also:

[Oracle Determinations Server configuration file](#)

The Rulebases directory

The Rulebases directory is where Rulebase Archives are placed to be deployed by default. Rulebase Archives (.zip files generated by Oracle Policy Modeling) placed here will be deployed when the Determinations Server starts and, depending on the configuration settings, while the Determinations Server is running. The Determinations Server can be configured to load its rulebases from another directory.

For Java:

The rulebases directory can be found in the following location:

[determinations-server/WEB-INF/classes/rulebases](#)

For .NET:

The Rulebases Directory defaults to a directory named 'rulebases' in the Web Application directory.

See also:

[Oracle Determinations Server configuration file](#)

The Plugins directory

The Plugins directory is where compiled determinations server plugins should be placed. From here they will be

loaded and used by the determinations server.

For Java:

The plugins directory can be found in the following location:

[determinations-server/WEB-INF/classes/plugins](#)

.jar files containing determinations server plugins should be placed in this directory.

Note: Loading of classes placed in .jar files in this directory will not work on some application servers. In these situations, the plugin.libraries configuration property should be used in application.properties.

For .NET:

The Plugins directory defaults to a directory named 'plugins' in the Web Application directory.

.dll files containing determinations server plugins should be placed in this directory.

See also:

[Oracle Determinations Server configuration file](#)

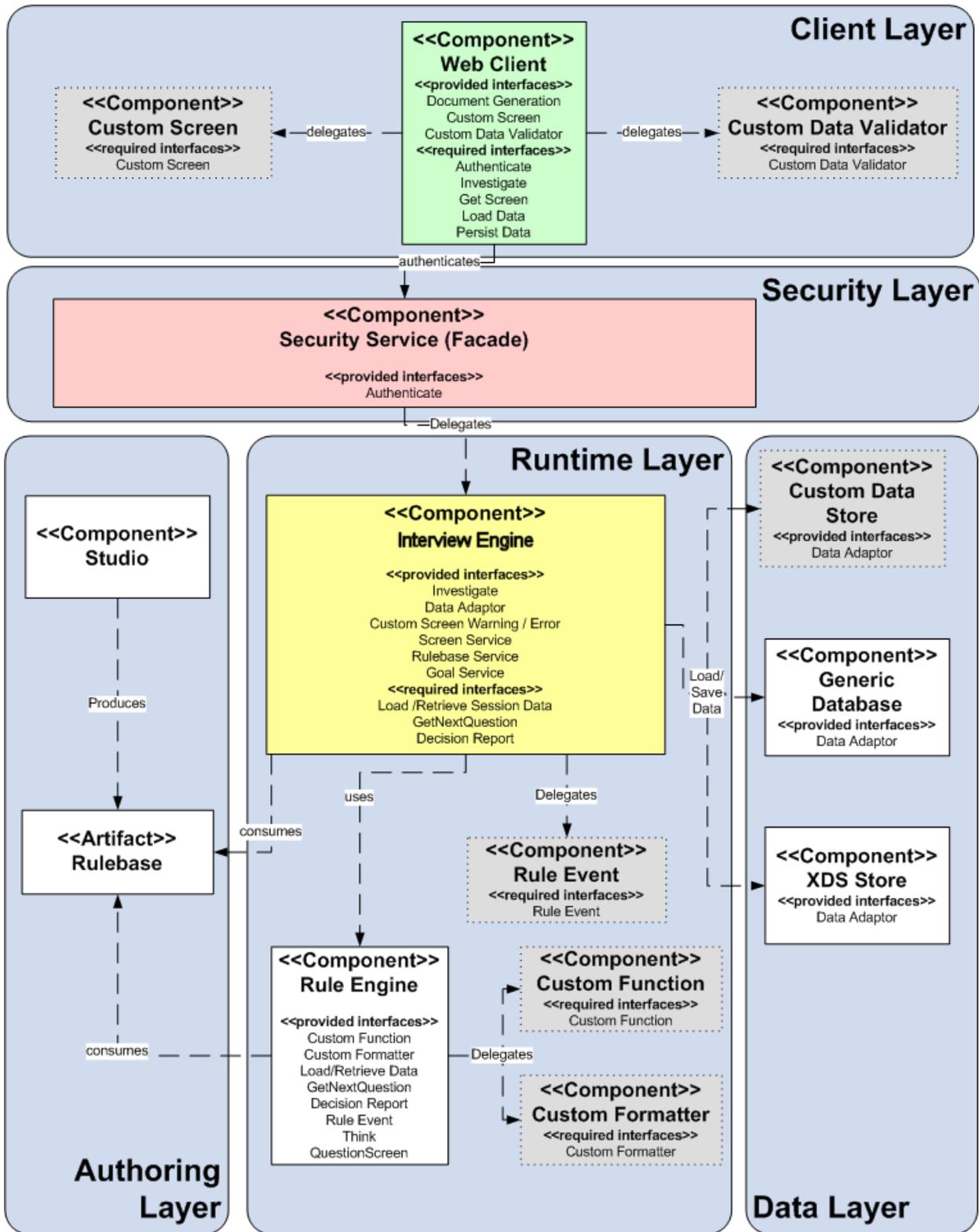
Understand Oracle Web Determinations

The following is a breakdown of what Web Determinations has to offer:

- Separation of layers.
- Clearly defined roles and responsibilities.
- Formal extension points - it is possible for the application to be customized (skinning & theming, custom screens, custom data validators and so on) without editing the source.
- Abstraction and introspection - you don't need to know the intimate details of the component's or even rulebase's implementation in order to be able to use it.
- Separability - write your own clients on top of the Interview Engine with relative ease.

See diagram

The following diagram illustrates the component architecture of Oracle Web Determinations and is followed by a description of each layer:

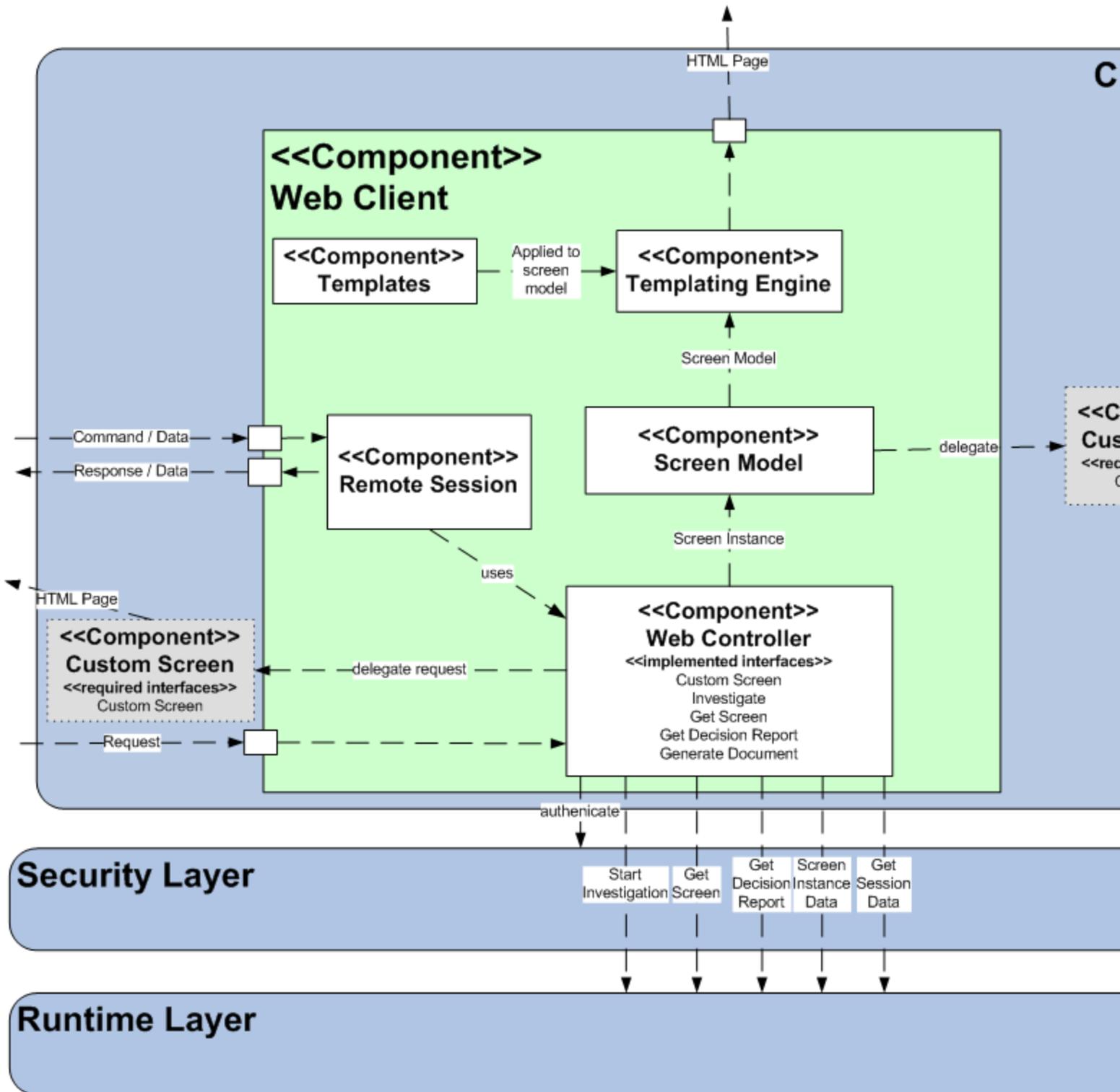


Client layer

The Client layer comprises a Web based client, and is responsible for web-related functionality.

- Render and display HTML screens from Interview Engine.
- Process web user actions/data for Interview Engine.
- Client-side validation.
- Skinning and themeing.
- Localization of the Interview web interface.
- Expose the extension points below:
 - Custom screens
 - Custom Data Validator
 - Document Generator
 - Custom Control.

See diagram



Security layer

The Security layer is responsible for providing authentication functionality for all Web Determinations sessions. It is primarily composed of a security service which is responsible for authenticating user credentials, ensuring the user has the required permissions to carry out the requested action.

Runtime layer

The runtime layer essentially composes two components:

- The Rule (Determinations) Engine which is responsible for providing the core inferencing and determination functionality.
- The [Interview Engine](#) which is responsible for providing the core 'interactive interview' components such as screen instances, screen flow, validation and warnings and transactional support.

Data layer

Provides a way for current interview session data to be saved and loaded via a Data Adaptor. The default Data Adaptor can be overridden to allow connectivity to other data sources.

Authoring layer

The Authoring layer primarily comprises the Oracle Policy Modeling application. Rulebases can be authored in the application, and deployed into Oracle Web Determinations to enable the rulebase interview process through the web.

Understand the Interview Engine

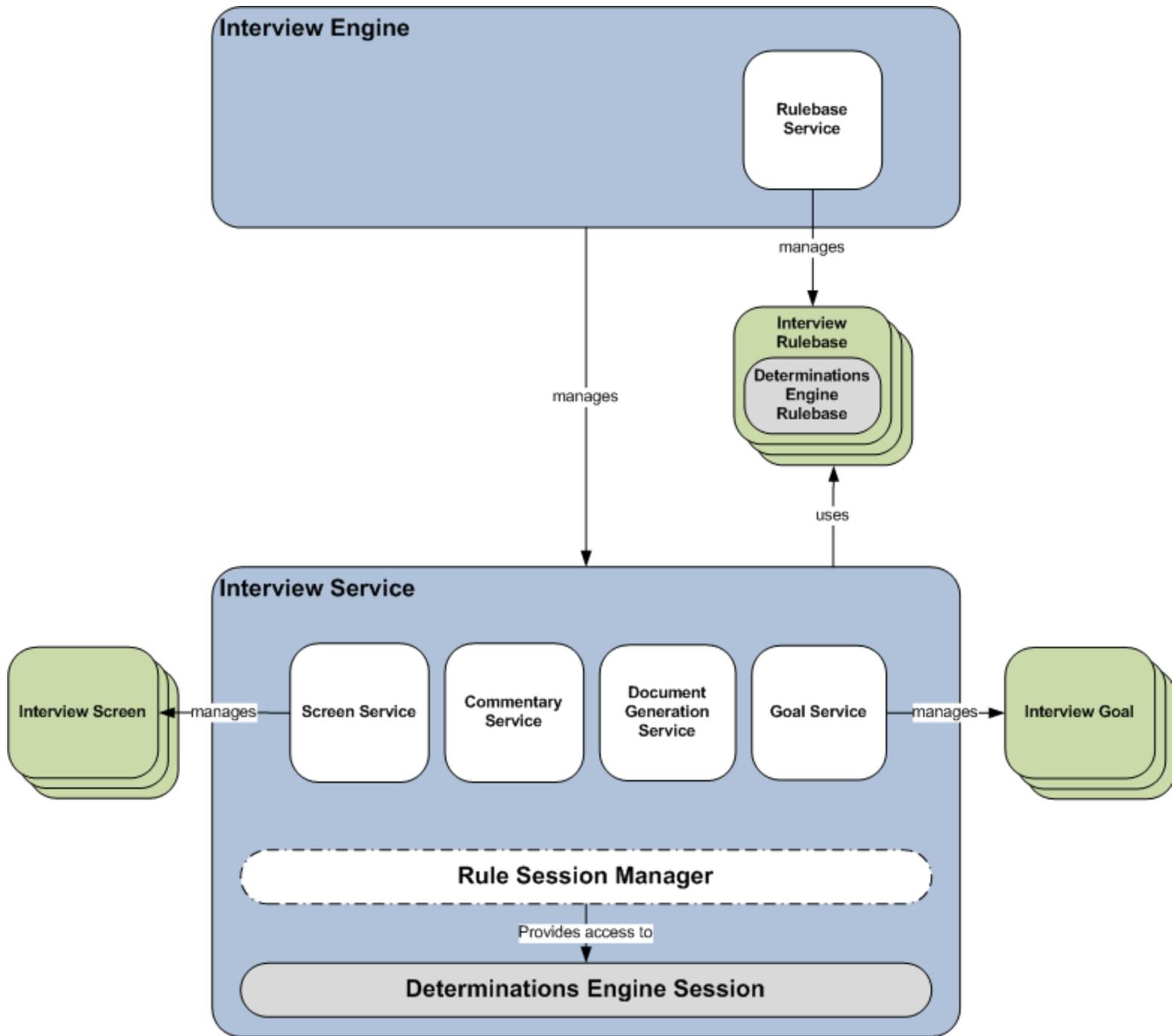
The Interview Engine provides the core functionality required to conduct an interactive investigation. It is responsible for providing the ability for you to:

- conduct an investigation.
- load/save data in/out of a session.
- get a specified screen.
- get a decision report.

To find out how to work with the Interview Engine to create a custom interview client, go to the topic: [Create a custom interview user experience](#).

Components and interfaces

Diagram



Interview Engine

The Interview Engine is responsible for creating Interview Sessions and managing the list of rulebases available to create interviews for. It provides the following services:

- **Rulebase Service** - responsible for managing the available Interview Rulebases and provides the ability to retrieve query the list of available rulebases.
- **Security Service** - responsible for authenticating users.

The Interview Engine engine itself is created by passing an engine configuration object to the Interview Engine Factory. These configuration properties control things such as providing information on how and where rulebases should be loaded from and other parameters controlling how Interview Sessions are created and managed. All interview sessions created through a given instance of the engine will inherit the behavior specified by the engine configuration.

Interview rulebase

The Interview rulebase provides access the abstract definition of screens and flows. It encapsulates a Determinations Engine rulebase, through which the rulebase data model can be accessed. Interview rulebases are required in order to create an Interview Session.

Interview Session

The Interview Session is the core component of the Interview Engine that provides the ability to conduct an interview, get interview screens, add data to a session, retrieve data from a session, generate documents and get commentary. It provides the following services:

- **Screen Service** - responsible for creating and managing all the available Interview Screens in the session. It provides the ability to get question, summary, data review and decision report screens. The screen service can also be queried to provide the list of all available screens currently available in the session.
- **Goal Service** - responsible for creating and managing all the available goals in the rulebase. It provides the ability to retrieve a specific goal as well as providing a list of all top level goals in the investigation.
- **Document Generation Service** - provides the ability to generate documents using the data and outcomes currently held in the session. The document generation itself is provided via the means of a plugin which allows integration with third party document generation solutions. For more information see [Document Generator plugin](#).
- **Commentary Service** - responsible for providing commentary and help for screens and/or questions in the interview. Commentary is provided by means of a plugin which allows third party commentary services to be integrated. For more information see [Commentary plugin](#).

The Rule Session Manager and transaction management

A key feature of the Interview Engine is its transaction support.

In Web Determinations, screen and data submissions are considered a single, atomic transaction. Therefore either all the data on that screen will be submitted or none of it will. This transactional support is provided by the Rule Session Manager which acts as a gateway between the Determinations Engine Session encapsulated in the Interview Session. All access to the Determinations Engine Session, both internally and externally is mediated through this layer.

Although the Rule Session Manager is transparent to clients of the Interview Engine API, it does however, have some impacts that users of the API should be aware of. Firstly, while Interview Session provides the ability to get access to the underlying Determinations Engine Session, clients **must not** modify the data held in the Determinations Engine Session directly but rather use the methods provided by the interview session itself. Secondly, the Web Determinations Session itself is transient and may be destroyed and re-constructed at any point during the life of the Interview Session. This means that references to objects contained in the Determinations Engine Session **cannot be cached**.

Interview Screens

An Interview Screen is the in memory, object representation of a rulebase screen. Unlike the Determinations Engine Screen, an Interview Screen is not an abstract template but rather a specific representation, containing the values, text substitutions etc based on the state of the session. The interview screen is static in sense that it will reflect the data held in the session at the time it was created. This means that if the session data is changed the screen **must** be reconstituted in order for it to reflect those changes.

Interview Goal

The Interview Goal is the object upon which an investigation is performed during an interview. Unlike previous versions, there is no distinction between an attribute goal and a flow goal. There are two ways of accessing goals; either through the goal service or via a goal control on a summary screen.

Retrieving a Decision Report for an Interview Goal

For information about retrieving a decision report from an interview goal, refer to the topic [Retrieve a Decision Report in an interview](#).

Access further resources on Oracle Policy Automation

If you are looking for information that isn't covered in either this help guide, the *Oracle Policy Modeling User's Guide*, or any of the other applicable help guides, then the **Oracle Policy Automation knowledge base** at support.oracle.com may be helpful. You will require Oracle customer details to access this area. Select the Knowledge area in the top menu, browse in the product list to **More Applications | Oracle Haley | Policy Automation**, and select the appropriate program. From here you can browse or search on knowledge base articles, including technical 'how to' instructions, known issues and their workarounds, and product announcements.

You can also visit the **Oracle Policy Automation Discussion Forum**, to search for details of any questions you may have, or ask questions directly if they have not already been discussed on the forum.

Web services

Topics in "Web services"

- Choose a web service to use
- Deploy Determinations Server
- Integrate Determinations Server with a client application
- Get an answer from the Determinations Server Assess request
- Test a rulebase using a web service
- Select the rulebase language a web service uses
- Update a Determinations Server rulebase

Choose a web service to use

What do you want to do?

[Make high performance auditable determinations](#)

[Decide whether to use the generic or specific data model for batch processing](#)

[Embed Interviews via XML API](#)

Make high performance auditable determinations

The Oracle Determinations Server provides a high performance and robust XML/SOAP service. It allows you to manage and deploy your rules to a single centralized service. The XML interface means that a wide range of enterprise applications can use Policy Automation.

The key features of the Oracle Determinations Server are:

- High performance - designed to handle many requests per second.
- Easy to integrate with new or existing enterprise applications
- Easy to scale and expand via Clustering
- Maintain the rules at a single location

The Oracle Determinations Server offers two modes of leveraging the Policy Automation Runtime components:

1. The stateless Assess Service, which is suited to batch style operations; or
2. The stateful Interview Service, which leverages the rulebase to provide guided questionnaires

Decide whether to use the generic or specific data model for batch processing

The Assess Services are provided in two forms, Generic and Specific. Each of these forms have their own WSDL and service endpoints; for example, if I deploy MyRulebase to the Oracle Determinations Server, it will create the following two sets of services:

1. the generic MyRulebase services with an endpoint at <http://my.server.com/determinations-server-/ases/soap/generic/myrulebase> with the WSDL available from <http://my.server.com/determinations-server-/ases/soap/generic/myrulebase?wsdl>
2. the specific MyRulebase services with an endpoint at <http://my.server.com/determinations-server-/ases/soap/specific/myrulebase> with the WSDL available from <http://my.server.com/determinations-server-/ases/soap/specific/myrulebase?wsdl>

The difference between the two services is that the Specific service has a WSDL and Schema specifically generated to suit the rulebase, whereas the generic service uses the same schema for every rulebase.

The Assess and GetScreen services can make use of the Specific Schema. Because the Specific schema is more meaningful, it is recommended that you use Specific services when using Rulebase operations.

The session data for a specific request is different from generic session data. To see the difference between generic and specific, see the following assess request message samples: [Sample Generic and Specific Request](#)

Embed Interviews via XML API

In addition to the native Java and .NET code API's, the Interview Service provides the capacity leverage the Interview technology via an XML API. Like the Assess Service, the Determinations Server provides an instance of the Interview Service for each deployed rulebase.

The endpoint of the Interview Service for the MyRulebase rulebase is:

<http://localhost:8080/det-server-102/interview/soap/MyRulebase>

and the WSDL for that service can be found at

<http://localhost:8080/det-server-102/interview/soap/MyRulebase?wsdl>

More information and examples of the format of the Interview Service messages can be found in the topics in the [Interview Service](#) section.

Deploy Determinations Server

For information on deploying Oracle Determinations Server, refer to the *Oracle Policy Automation Installation Guide*.

Integrate Determinations Server with a client application

What do you want to do?

[Use generic web service session data format to build web services session data xml from scratch](#)

[Call Determinations Server from Siebel](#)

For information on calling Determinations Server from a Java application, see:

[Tutorial: Create and use a JAX-WS web service client for Oracle Determinations Server](#)

For information on calling Determinations Server from a .NET application, see:

[Tutorial: Create and use a C# client for Oracle Determinations Server](#)

Use generic web service session data format to build web services session data xml from scratch

In order to use the Oracle Determinations Server service from your application, you must:

1. Construct a request
2. Send the request to the Determinations Server endpoint
3. Wait for the response
4. Parse the response and extract the information you need.

There are many tools and utilities to assist you to construct requests, parse responses and make the web service calls. They all essentially accomplish the same thing, helping build a SOAP request containing XML, sending the request with the correct HTTP headers to a specified endpoint, and then parsing the SOAP response containing xml.

Two tutorials are provided that will walk you through using a standard Web Service client for Java and C#.

Go to:

[Tutorial: Creating and using a C# client for Oracle Determinations Server](#)

[Tutorial: Create and use a JAX-WS web service client for Oracle Determinations Server](#)

See also:

[Example: Assess Request xml](#)

Call Determinations Server from Siebel

For information relating to calling Oracle Determinations Server from Siebel, refer to the *Oracle Policy Automation Connector for Siebel Developer Help*.

Get an answer from the Determinations Server Assess request

What do you want to do?

Get an attribute as an answer

Get a relationship as an answer

Process multiple determinations within the same Assess request

Handle errors or warnings returned by the Determinations Server

Obtain a decision report from a web service

Specify outcomes for all instances of a given entity using the Outcome option

Get an attribute as an answer

You can get the answer to an attribute from the Oracle Determinations Server by using an Assess operation. Typically this attribute would be an inferred goal of a rulebase. In order to get an answer for this attribute goal you will need to provide sufficient information in the Assess operation for the answer to be inferred.

In both the generic and specific interfaces, you can ask for the value of the attribute by specifying an outcome style (outcome-style) instead of supplying a value. An outcome style tells the assess operation that you are asking for a value rather than setting one.

See also: [Important Note: do not set value and specify outcome style for same attribute.](#)

Example generic request

```
<entity id="client">
  <instance id="client-100064">
    <attribute id="client_is_eligible" outcome-style="value-only" />
    ...
  </instance>
</entity>
```

In the example above, the attribute element with an outcome-style is inside the entity element for client-100064. This means that I am requesting the value of the attribute client_is_eligible for the client identified in my assess request by the id client-100064. The outcome-style has been specified to be value-only, meaning I am only interested in the value. Alternatively you could specify an outcome-style of decision-report or base-attributes if you wanted the answer reported with a decision report.

In the assess response, you will find all attribute elements with an outcome-style now contain the inferred value of that attribute.

As an alternative to supplying the outcome style, we can supply both the known-outcome-style and unknown-outcome-style. This allows you to control the content of the response depending on whether the inferred attribute is known or unknown. In the example below we are again requesting the value of client_is_eligible for client instance client-100064. If the value of the attribute is unknown, we want a decision report. If the value of the attribute is known, we are only interested in the value:

Example generic request - known and unknown outcome styles

```
<entity id="client">
  <instance id="client-100064">
    <attribute id="client_is_eligible" known-outcome-style="value-only" unknown-outcome-style="decision-report" />
    ...
  </instance>
</entity>
```

```
</instance>
</entity>
```

In the example response, we can see that the value returned is true (the client is eligible):

Example attribute in response

```
<entity id="client">
  <instance id="client-100064">
    <attribute id="client_is_eligible" type="boolean" inferred="true">
      <typ:boolean-val>true</typ:boolean-val>
    </attribute>
    ...
  </instance>
</entity>
```

Use of the specific format is very similar. The outcome-style (or both known-outcome-style and unknown-outcome-style) is specified for the element representing the attribute that we are interested in to indicate that the Determinations Server should put the value in the response:

Example specific request

```
<list-client>
  <client id="client-100064">
    <client_is_eligible outcome-style="value-only" />
    ...
  </client>
</list-client>
```

Example attribute in specific response

```
<list-client>
  <client id="client-100064">
    <client_is_eligible type="boolean" inferred="true">
      <typ:boolean-val>true</typ:boolean-val>
    </client_is_eligible>
    ...
  </client>
</list-client>
```

Get a relationship as an answer

In the same way that you can ask for the value of an attribute as answer, you can also ask for the value of an inferred relationship. This is also done through the Assess operation, by specifying an outcome-style (or known-outcome-style and unknown-outcome-

style) in the request.

Example generic request

```
<entity id="client">
  <instance id="client-100064">
    ...
    <relationship id="clients_eligible_children"
      outcome-style="value-only" />
    ...
  </instance>
</entity>
```

As with attributes (see above), you can specify the outcome style to be the value (in this case all targets of the inferred relationship) or the value and a decision report.

In the assess response, you will find all relationship elements with outcome styles specified populated with all the targets of that relationship.

Example relationship in response

```
<entity id="client">
  <instance id="client-100064">
    ...
    <relationship name="clients_eligible_children"
      state="known" inferred="true">
      <target instance-id="child_100064-1"/>
      <target instance-id="child_100064-2"/>
    </relationship>
    ...
  </entity>
</list-entity>
```

In the example response, we can see that the inferred relationship is known for the client, and that there are two eligible targets, identified by their id's.

Use of the specific format is very similar. The outcome-style (or both known-outcome-style and unknown-outcome-style) is specified for the element representing the relationship that we are interested in to indicate that the Determinations Server should put the targets in the response:

Example specific request

```
<list-client>
  <client id="client-100064">
    ...
    <relationships>
```

```
        <clients_eligible_children outcome-style="value-only" />
    </relationships>
</client>
</list-client>
```

Example relationship in specific response

```
<list-client>
  <client id="client-100064">
    ...
    <relationships>
      <clients_eligible_children state="known" inferred="true">
        <target instance-id="child_100064-1"/>
        <target instance-id="child_100064-2"/>
      </clients_eligible_children>
    </relationships>
    ...
  </client>
</list-client>
```

Process multiple determinations within the same Assess request

The Determinations Server is able to process multiple determinations within the same Assess request. To do this you must:

1. Design your rulebase to concurrently determine multiple outcomes.
2. At runtime, construct your request to send all the entities you want to reason on, and all the outcomes that you want.

Rulebase Design

You can design a rulebase in such a way that it can make multiple determinations concurrently if the determinations are attributes or, less commonly, inferred relationships belonging to a non-singleton (and not the global) entity.

Once the outcomes belong to an entity, it is possible to create multiple entities and infer the values for all entities at once.

Example

In a Human Resources department, we want to determine the number of days of leave per year an employee is eligible for. We also want to determine if that person is eligible for long service leave. Both these outcomes are based on the number of years the employee has served in the company.

This rulebase can be found in the OPA runtime directory at: [examples\rulebases\compiled\EmployeeLeave.zip](#)

Employee Attributes

Attribute	Type
Date joined company	Base level (input)

Attribute	Type
The employee is eligible for long service leave	Inferred (outcome)
The number of days of leave per year	Inferred (outcome)

If we decide that the employee is the global entity, and the attributes are created against this entity, then we can only infer one employee's outcomes per session, as we can only have one global entity.

However, if we create an entity "the employee" we can then have many employees in the rulebase, and we can effectively run a batch job, getting many outcomes within in the same session and think cycle.

Assess request at runtime

If the rulebase has been designed along the principles above, we can use a single Assess request to get multiple outcomes.

This can be done with the Determinations Server, and it is also possible to do this against the Determinations Engine API directly. In both cases the principle is:

1. Create a session with multiple entity instances
2. Provide all the input data necessary to determine the outcomes
3. Retrieve outcomes.

Example Assess Request and Response

In the following example request, we are sending an Assess request with three employee entities. For each of these entities we are asking for the outcomes for the number of days of annual leave each employee is entitled to, and also whether the employee is eligible for long service leave. Because these outcomes are on the employee entity we can ask for outcomes for multiple employee entities, effectively doing a batch request.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:assess-request>
      <typ:config>
        <typ:outcome>
          <typ:entity id="employee">
            <typ:attribute-outcome id="employee_days_leave_per_year" outcome-style="value-only"/>
            <typ:attribute-outcome id="employee_long_service_leave" outcome-style="value-only"/>
          </typ:entity>
        </typ:outcome>
      </typ:config>
      <typ:global-instance>
        <typ:entity id="employee">
          <typ:instance id="employee1">
            <typ:attribute id="employee_date_joined_company">
              <typ:date-val>1986-02-16</typ:date-val>
            </typ:attribute>
          </typ:instance>
        </typ:entity>
      </typ:global-instance>
    </typ:assess-request>
  </soapenv:Body>
</soapenv:Envelope>
```

```

<typ:instance id="employee2">
  <typ:attribute id="employee_date_joined_company">
    <typ:date-val>2001-01-01</typ:date-val>
  </typ:attribute>
</typ:instance>
<typ:instance id="employee3">
  <typ:attribute id="employee_date_joined_company">
    <typ:date-val>1992-12-16</typ:date-val>
  </typ:attribute>
</typ:instance>
</typ:entity>
</typ:global-instance>
</typ:assess-request>
</soapenv:Body>
</soapenv:Envelope>

```

In the following response, we can see that for each employee we have an answer for the two outcomes we requested. We can parse the returned XML and get the outcomes for each employee.

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:assess-response>
      <typ:global-instance>
        <typ:entity id="employee" inferred="false">
          <typ:instance id="employee1">
            <typ:attribute id="employee_days_leave_per_year" type="number" inferred="true">
              <typ:number-val>27.5</typ:number-val>
            </typ:attribute>
            <typ:attribute id="employee_long_service_leave" type="boolean" inferred="true">
              <typ:boolean-val>true</typ:boolean-val>
            </typ:attribute>
            <typ:attribute id="employee_date_joined_company" type="date">
              <typ:date-val>1986-02-16</typ:date-val>
            </typ:attribute>
          </typ:instance>
          <typ:instance id="employee2">
            <typ:attribute id="employee_days_leave_per_year" type="number" inferred="true">
              <typ:number-val>20.5</typ:number-val>

```

```

    </typ:attribute>
    <typ:attribute id="employee_long_service_leave" type="boolean" inferred="true">
      <typ:boolean-val>true</typ:boolean-val>
    </typ:attribute>
    <typ:attribute id="employee_date_joined_company" type="date">
      <typ:date-val>2001-01-01</typ:date-val>
    </typ:attribute>
  </typ:instance>
  <typ:instance id="employee3">
    <typ:attribute id="employee_days_leave_per_year" type="number" inferred="true">
      <typ:number-val>24.5</typ:number-val>
    </typ:attribute>
    <typ:attribute id="employee_long_service_leave" type="boolean" inferred="true">
      <typ:boolean-val>true</typ:boolean-val>
    </typ:attribute>
    <typ:attribute id="employee_date_joined_company" type="date">
      <typ:date-val>1992-12-16</typ:date-val>
    </typ:attribute>
  </typ:instance>
</typ:entity>
</typ:global-instance>
</typ:assess-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

See also:

[Assess Request and Response Elements](#)

[Example: Assess Request xml](#)

[Example: Assess Response xml](#)

Handle errors or warnings returned by the Determinations Server

The Determinations Server will return Error and Warning events that may happen during an assess operation.

For Warnings, you must specify in the assess-request that you want any warning returned. You can do this by setting the optional "show-events" attribute of the assess-request "config" element. This is the same for both the Specific and Generic service.

Example

```

<assess-request>
  <config>
    <show-events>true</showevents>
  </config>
  <!-- generic or specific session data follows -->
  ...
</assess-request>

```

When show events has been set to true, any warning events will display at the top of the assess-response.

Example

```
<event entity-id="child" instance-id="child_1" name="warning">
  <message>"the child's age might be incorrect"</message>
  <parameters>
    <value>"the child's age might be incorrect"</value>
  </parameters>
  <decision-report report-style="base-attributes">
    <attribute-node id="dn:1" entity-id="global" instance-id="global" hypothetical-instance="false" attribute-id="child_
age"
    type="text" text=" The child's age is 200.0." inferred="false">
      <number-val>200.0</number-val>
    </attribute-node>
  </decision-report>
</event>
```

A warning event is always returned with the entity and a decision report attached.

For error events, no configuration is needed. Error events will always return a SOAP Fault if an error is raised in the inferencing.

Example

```
<SOAP-ENV:Fault>
  <faultcode>SOAP-ENV:Client</faultcode>
  <faultstring>The Rulebase generated an error event</faultstring>
  <detail>
    <typ:error-response>
      <typ:code>assess.request.event.error</typ:code>
      <typ:message>The Rulebase generated an error event</typ:message>
    </typ:error-response>
  </detail>
</SOAP-ENV:Fault>
```

Obtain a decision report from a web service

Get a decision report showing the rules used to reach a decision

When you request an outcome from the Determinations Server, you have control over what information is returned:

- You can ask for just the value
- You can ask for the value and a decision report

You can also ask for different information to be returned depending on whether the outcome is known or unknown; for example, if the outcome is known, you may just want the value, but, if the outcome is unknown, you may want a full decision report.

When you request an attribute or relationship outcome you can specify what information you want displayed by setting the following XML attributes on the attribute or relationship element in the Assess request.

outcome-style:

this attribute controls the style of the outcome generally (regardless of when it is known or unknown).

known-outcome-style:

this attribute controls the style of the outcome when the outcome is known.

unknown-outcome-style:

this attribute controls the style of the outcome when the outcome is known.

For every outcome, you must specify either the *outcome-style*, or **both** the *known-outcome-style* and *unknown-outcome-style*.

The valid values for outcome styles are *value-only*, *decision-report* and *base-attributes*.

value-only

will return only the value for the outcome.

decision-report

will return a full decision report including all inferred attributes and relationships relationships that are relevant to the outcome.

base-attributes

will return a decision report, but only base level (non-inferred) attributes and relationships that are relevant to the outcome.

Request - Attribute outcome with decision report

```
<assess-request>
  <global-instance>
    ...
    <attribute id="eligible_teenage_child_allowance" known-outcome-style="value-only" unknown-outcome-style="decision-report"/>
    ...
  </global-instance>
</assess-request>
```

Request - Relationship outcome with decision report

```
<assess-request>
  <global-instance>
    ...
    <relationship id="eligible_children" known-outcome-style="value-only" unknown-outcome-style="decision-report"/>
    ...
  </global-instance>
</assess-request>
```

```
...
  </global-instance>
</assess-request>
```

Response with decision report

In the example response below, you can see that the requested attribute *eligible_teenage_child_allowance* is unknown. The decision report explains that the contributing entities, attributes and relationships are:

- The relationship *claimantschildren*, from the global to child entities.
- The inferred attribute the child is a teenager for three child entity instances
- The base level attribute the child's age for the three children.

By examining the decision report you can see that the reason that the attribute is unknown is because the child's age is unknown for *child_3*.

Example

```
<typ:assess-response>
  <typ:global-instance>
    ...
    <typ:attribute id="eligible_teenage_allowance" type="boolean" inferred="false">
      <typ:unknown-val/>
      <typ:decision-report report-style="decision-report">
        <typ:attribute-node id="dn:0" entity-id="global" instance-id="global" hypothetical-instance="false" attribute-id="eligible_
teenage_
teenage_allowance" type="boolean" text="Is the claimant eligible for the teenage child allowance?" inferred-
d="false">
          <typ:unknown-val/>
          <typ:relationship-node id="dn:1" source-entity-id="global" source-instance-id="global" hypothetical-instance-
e="false"
target-entity-id="child" relationship-id="claimantschildren" state="known" inferred="false">
            <typ:target instance-id="child1"/>
            <typ:target instance-id="child2"/>
            <typ:target instance-id="child3"/>
          </typ:relationship-node>
          <typ:attribute-node id="dn:2" entity-id="child" instance-id="child1" hypothetical-instance="false" attribute-
id="child_
teenager" type="boolean" text="The child is not a teenager." inferred="true">
            <typ:boolean-val>false</typ:boolean-val>
            <typ:attribute-node id="dn:3" entity-id="child" instance-id="child1" hypothetical-instance="false" attribute-
id="child_age"
type="number" text="The child's age is 8." inferred="false">
              <typ:number-val>8.0</typ:number-val>
            </typ:attribute-node>
```

```

        </typ:attribute-node>
        <typ:attribute-node id="dn:4" entity-id="child" instance-id="child2" hypothetical-instance="false" attribute-
id="child_
        teenager" type="boolean" text="The child is not a teenager." inferred="true">
        <typ:boolean-val>>false</typ:boolean-val>
        <typ:attribute-node id="dn:5" entity-id="child" instance-id="child2" hypothetical-instance="false" attribute-
id="child_age"
        type="number" text="The child's age is 11." inferred="false">
        <typ:number-val>11.0</typ:number-val>
        </typ:attribute-node>
        </typ:attribute-node>
        <typ:attribute-node id="dn:6" entity-id="child" instance-id="child3" hypothetical-instance="false" attribute-
id="child_
        teenager" type="boolean" text="Is the child a teenager?" inferred="false">
        <typ:unknown-val/>
        <typ:attribute-node id="dn:7" entity-id="child" instance-id="child3" hypothetical-instance="false" attribute-
id="child_age"
        type="number" text="The child's age is unknown." inferred="false">
        <typ:unknown-val/>
        </typ:attribute-node>
        </typ:attribute-node>
        </typ:attribute-node>
        </typ:decision-report>
    </typ:attribute>
    ...
</typ:global-instance>
</typ:assess-response>

```

Control the information included in a decision report

There are several ways in which you can control information included in a decision report. The first way is to set the silent and invisible options on the attribute (see [authoring > attributes > silent and invisible](#)).

In addition to controlling the attributes that appear through their silent/invisible properties when authoring the rulebase, for the Determinations Server you can also set a decision report to base-attributes. This is specified when you request an attribute or relationship outcome in an Assess operation. When this value is set for the decision report style no inferred attributes will be returned, only base level attributes.

See [Assess Operation Request and Response Elements](#) for more information on outcome-styles.

Example - generic

```

<attribute id="eligible_teenage_child_allowance" outcome-style="base-attributes"/>
<relationship id="eligible_children" outcome-style="base-attributes"/>

```

Example - specific

```

<eligible_teenage_child_allowance outcome-style="base-attributes"/>

```

```
<eligible_children outcome-style="base-attributes"/>
```

Specify outcomes for all instances of a given entity using the Outcome option

In the example that follows, we specify the attribute outcomes "B.Benefit_Amount" and "monthly_benefit_timeline" as well as the relationship outcome "the_payments" for every single instance of the entity "the_benefit". The **outcome-style** is used to specify the level of detail that will be provided in the decision report; a full decision report will returned.

Example:

```
<typ:assess-request>
  <typ:config>
    <typ:outcome>
      <typ:entity id="the_benefit">
        <typ:attribute-outcome id="B.Benefit_Amount" outcome-style-
e="decision-report"/>
        <typ:attribute-outcome id="monthly_benefit_timeline" outcome-style-
e="decision-report"/>
        <typ:relationship-outcome id="the_payments" outcome-style-
e="decision-report"/>
      </typ:entity>
      <typ:entity id="the_payment">
        <typ:attribute-outcome id="P.Payment_Amt" outcome-style-
e="decision-report"/>
      </typ:entity>
    </typ:outcome>
  </typ:config>
```

The following describes each of the elements used in the above example (more information can be found in [Assess Operation Request and Response Elements](#) in the Technical Reference section):

outcome

Specifies the assess outcomes.

entity id

Specifies the public name of the entity.

attribute-outcome id

Corresponds to the public name of the attribute.

relationship-outcome id

Corresponds to the public name of the relationship.

outcome-style

This is the default outcome-style to use whether known or unknown. The outcome style attribute can be "value-only" (no decision report, just the value), "base-attributes" (a decision reports showing the relevant attributes, but only the base level ones), and "decision-report" a full decision report.

Test a rulebase using a web service

What do you want to do?

Find out the installed version of Determinations Server

Find out what rulebases are deployed on the Determinations Server

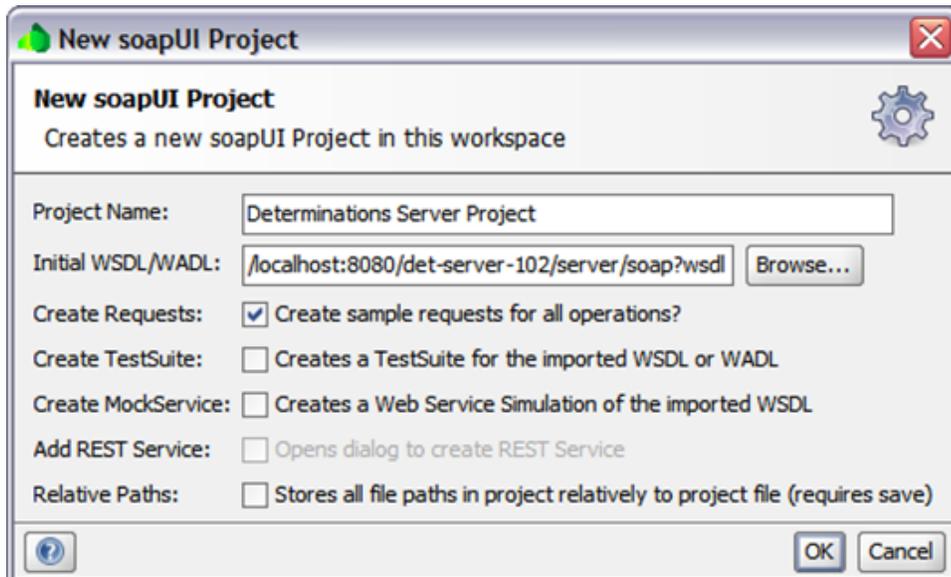
Use soapUI to test a rulebase

Create and execute an Assess operation

- soapUI is an application specifically designed to test Web Services. A free version of soapUI is available at <http://www.soapui.org/>.
- soapUI can be used to test your rulebases deployed on the Determinations Server, allowing you to construct requests, send them to the Determinations Server and examine the responses.
- soapUI is a Java application, but it can be used to test both the Java and .NET Determinations Server services.

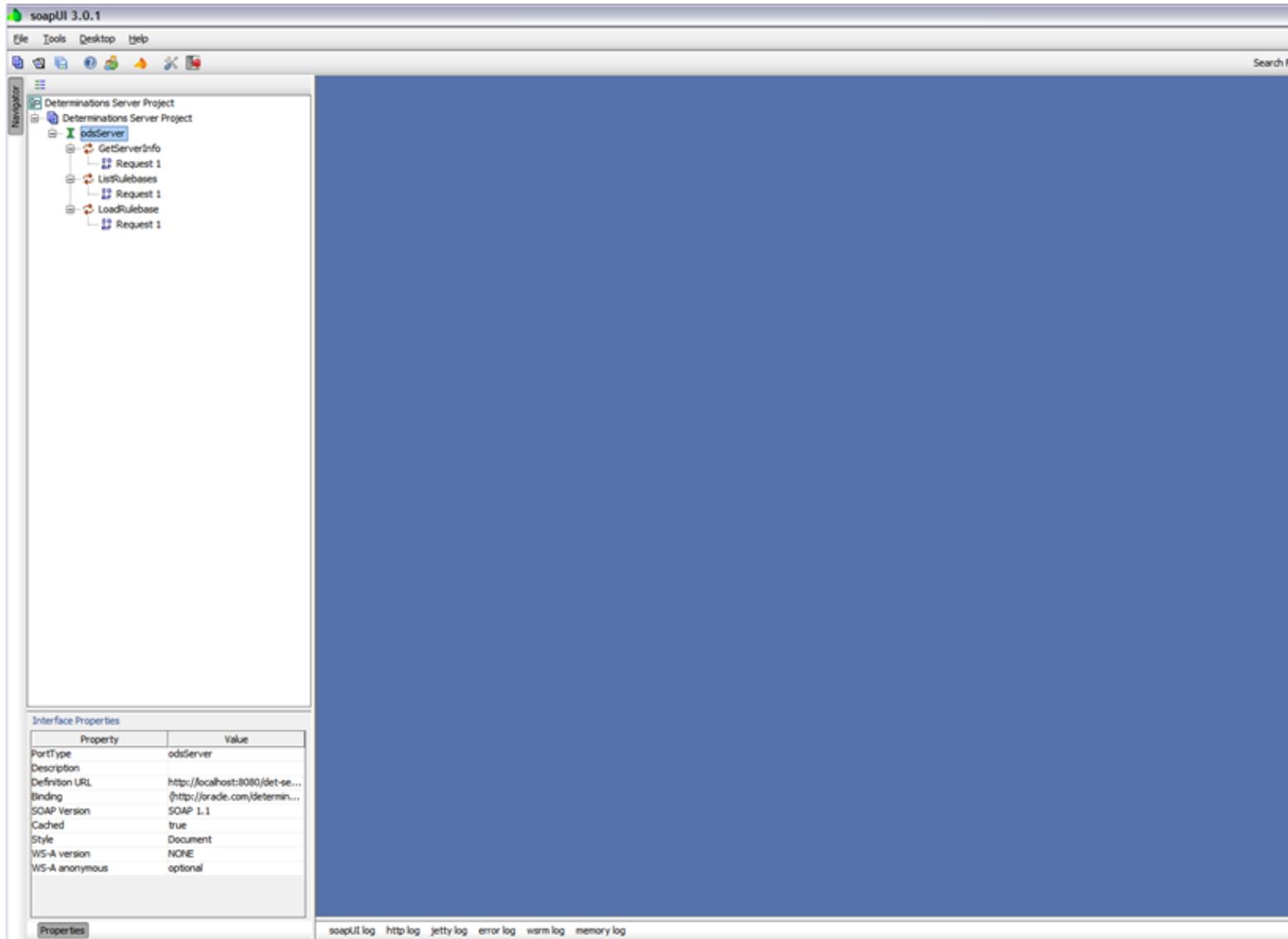
To import the Determinations Server WSDL into soapUI, do the following:

1. Make sure that the Determinations Server is running
2. Start soapUI
3. Choose **File > New soapUI project** and give the project a name of your choice, but for the initial WSDL/WADL, put the WSDL for the Determinations Server. For java this will be <http://<server>:<port>/determinations-server-server/soap?wsdl> and for .NET this will be <http://<server>:<port>/determinations-server/server/soap.asmx?wsdl>
4. Click **OK**



soapUI will create a new project and import the WSDL into that project giving you a service definition named *odsServer*. If you left the option *Create sample requests for all operations on*, you will have a sample operation for the three Determinations Server Service Operations: *GetServerInfo*, *ListRulebases* and *LoadRulebase*.

If an error occurred, it is probably because the Initial WSDL URL is incorrect.



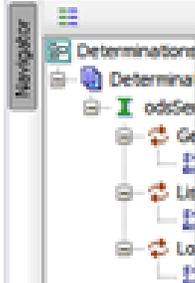
Find out the installed version of Determinations Server

1. Make sure that the Determinations Server is running
2. On the left hand side of soapUI, go to the *GetServerInfo* operation and expand it so you can see *Request 1*. Double-click on that request to open it. Notice that the request has been created by soapUI. Because there are no parameters needed for a *GetServerInfo* operation, you can run this operation without modification. To run this operation, click on the green arrow in the top lefthand corner of the request panel.

3. The response should appear in the right hand panel. The GetServerInfo information returns versions for: the Determinations Server, the Determinations Engine, the Interview Engine, and the time zone that Determinations Server is using.

Find out what rulebases are deployed on the Determinations Server

1. Make sure that the Determinations Server is running
2. On the left hand side of soapUI, go to the ListRulebases operation and expand it so you can see Request 1. Double-click on that request to open it. Notice that the request has been created by soapUI. Because there are no parameters needed for a ListRulebases operation, you can run this operation without modification. To run this operation, click on the green arrow in the top lefthand corner of the request panel.
3. The response should appear in the right hand panel. In the screenshot below, you should be able to see that there is one rulebase, "SimpleBenefits", deployed to this Determinations Server along with the list of language(s) supported by that rulebase and the WSDL URLs for all services available for that rulebase.



Request 1

http://localhost:8080/det-server-102/server/soap/10.2

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">  
  <soapenv:Header/>  
  <soapenv:Body>  
    <typ:list-rulebases-request/>  
  </soapenv:Body>  
</soapenv:Envelope>
```

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">  
  <SOAP-ENV:Header/>  
  <!10n:international>  
    <!10n:locale>en_US</!10n:locale>  
    <!10n:tz>GMT+1100</!10n:tz>  
  </!10n:international>  
</SOAP-ENV:Header>  
<SOAP-ENV:Body>  
  <typ:list-rulebases-response>  
    <typ:rulebase name="SimpleBenefits">  
      <typ:available-languages>  
        <typ:language>en_US</typ:language>  
      </typ:available-languages>  
      <typ:available-services>  
        <typ:service name="odsInterviewService">http://localhost:8080/det-server-102/server/soap/10.2</typ:service>  
        <typ:service name="odsInterviewService102">http://localhost:8080/det-server-102/server/soap/10.2</typ:service>  
        <typ:service name="odsAssessServiceSpecific">http://localhost:8080/det-server-102/server/soap/10.2</typ:service>  
        <typ:service name="odsAssessServiceGeneric">http://localhost:8080/det-server-102/server/soap/10.2</typ:service>  
        <typ:service name="odsAssessServiceSpecific102">http://localhost:8080/det-server-102/server/soap/10.2</typ:service>  
        <typ:service name="odsAssessServiceGeneric102">http://localhost:8080/det-server-102/server/soap/10.2</typ:service>  
        <typ:service name="odsAssessServiceSpecific10">http://localhost:8080/det-server-102/server/soap/10.2</typ:service>  
        <typ:service name="odsAssessServiceGeneric10">http://localhost:8080/det-server-102/server/soap/10.2</typ:service>  
      </typ:available-services>  
    </typ:rulebase>  
  </typ:list-rulebases-response>  
</SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

Request Properties

Prop...	Value
Name	Re...
Des...	
Me...	265
Enc...	UTF-8
End...	htt...
En...	
Pol...	false
Use...	
Pas...	
Do...	
WS...	
WS...	

Aut Headers ... Attachments... WS-A WS-RM

Headers (7) Attachments (0) SSL Info WSS (0)

response time: 1952ms (1879 bytes)

soapUI log http log jetty log error log warn log memory log

Use soapUI to test a rulebase

You can use soapUI to test a rulebase deployed in the Determinations Server

To import the rulebase WSDL, do the following:

1. Make sure that the Determinations Server is running
2. Follow the steps in "Find out what rulebases are deployed on the Determinations Server" to get the Generic or Specific WSDLs for your deployed rulebase.
3. In the left hand pane, right-click on the soapUI project and choose "Add WSDL", enter the WSDL for the rulebase (for example, <http://localhost:8080/det-server-102/assess/soap/generic/SimpleBenefits?wsdl>). It is probably a good idea to turn off "Create sample requests for all operations". In the figure below, you can see soapUI with the new rulebase WSDL on the left hand pane. There are two operations: Assess, and ListGoals.



Navigator

- Determinations Server Project
 - Determinations Server Project
 - odsServer
 - GetServerInfo
 - Request 1
 - ListRulebases
 - Request 1
 - LoadRulebase
 - Request 1
 - odsAssessServiceGeneric_SimpleBenefits
 - Assess
 - ListGoals

Request 1

http://localhost:8080/det-server-102/server/soap/10.2

```

<soapenv:Envelope xmlns:
  <soapenv:Header/>
  <soapenv:Body>
    <typ:list-rulebase
  </soapenv:Body>
</soapenv:Envelope>
    
```

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.
  <SOAP-ENV:Header>
    <!10n:international>
    <!10n:locale>en_US</!10n:locale>
    <!10n:tz>GMT+1100</!10n:tz>
    </!10n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:list-rulebases-response>
      <typ:rulebase name="SimpleBenefits">
        <typ:available-languages>
          <typ:language>en_US</typ:language>
        </typ:available-languages>
        <typ:available-services>
          <typ:service name="odsInterviewService">htt
          <typ:service name="odsInterviewService102">
          <typ:service name="odsAssessServiceSpecific
          <typ:service name="odsAssessServiceGeneric"
          <typ:service name="odsAssessServiceSpecific
          <typ:service name="odsAssessServiceGeneric
          <typ:service name="odsAssessServiceSpecific
          <typ:service name="odsAssessServiceGeneric
        </typ:available-services>
      </typ:rulebase>
    </typ:list-rulebases-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
    
```

Interface Properties

Property	Value
Port type	odsAssessServiceGe...
Description	
Definition URL	http://localhost:808...
Binding	{http://oracle.com/d...
SOAP Version	SOAP 1.1
Cached	true
Style	Document
WS-A version	NONE
WS-A anonymous	optional

Create and execute an Assess operation

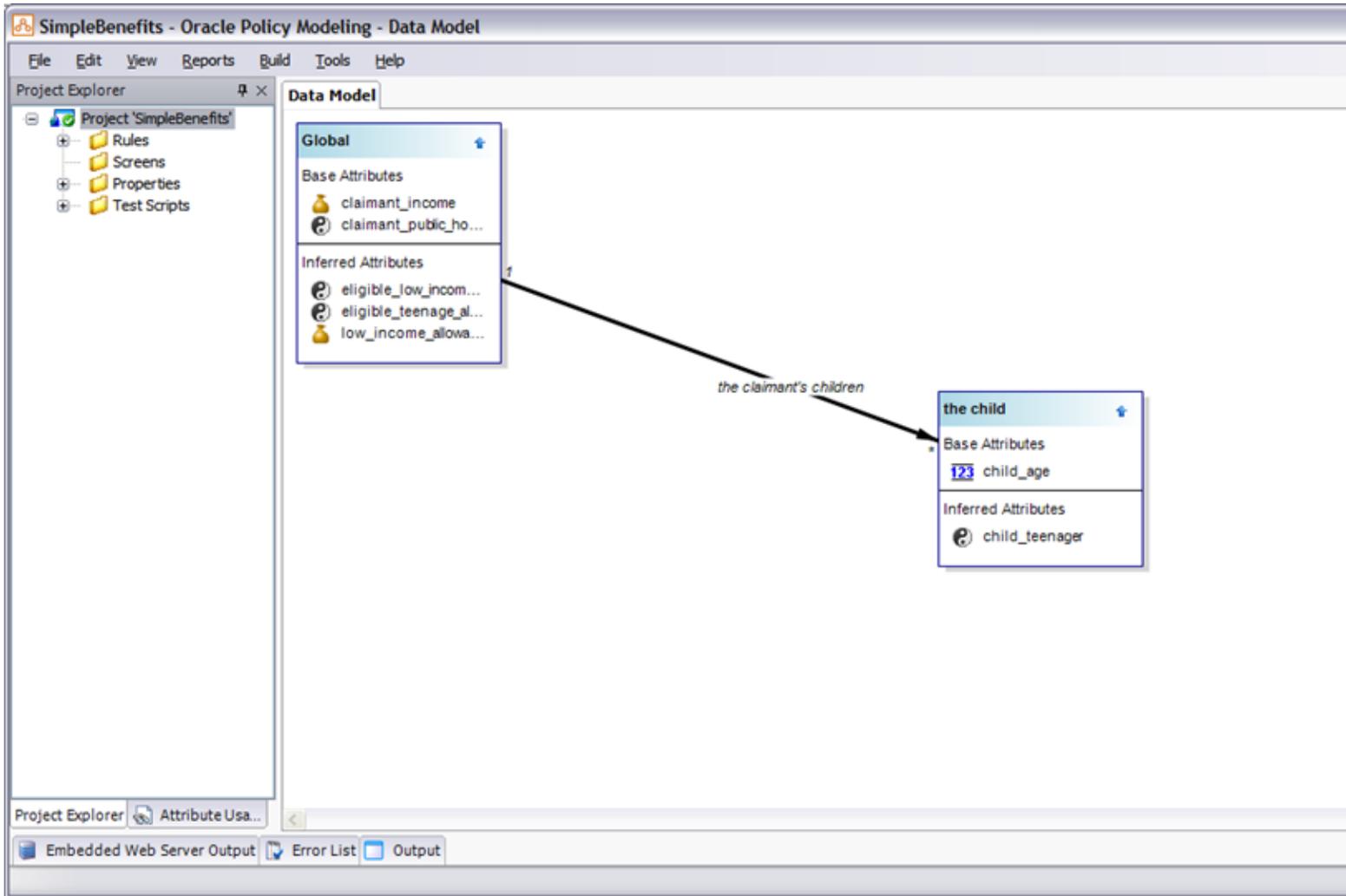
Now that you have created the rulebase service you can test a simple request. It is a good idea to have the rulebase open in Oracle Policy Modeling so you can create attributes, entities and relationships with the correct name. You can see the attributes, entities and relationships that the rulebase uses by viewing the Build Model or the Data Model (**View menu > Build Model/Data Model**).

1. In the left hand pane of soapUI, right-click on Assess operation and choose "New Request". If asked to "Create optional elements in schema?" it is best to choose "No".
2. In the right hand pane, a request will be created with a blank assess request. You can now add entity instances, and their attributes and relationships and execute the request.
3. Once you have the request you can right click on the right hand pane and choose "Validate", which will validate your request against the rulebase WSDL. If everything is correct, soapUI will reply "Validation OK", otherwise it will provide a list of errors.
4. Execute the request by clicking the green arrow. The response should appear in the far right hand pane:
 - If the request was successful, the Determinations Server will fill out values of the specified outcome(s).
 - If there was a problem processing the request, the Determinations Server will return a SOAP fault detailing the error(s) encountered.

Example Generic Assess for SimpleBenefits Rulebase

Looking at the Data Model for the Simple Benefits rulebase ([examples\rulebases\compiled\SimpleBenefits.zip](#) in the Oracle Policy Automation Runtime package), we can see that the Global has some attributes and a relationship to the child entity.

For this example we will investigate the goal "eligible_low_income_allowance" and "eligible_teenage_allowance".



Start the Assess request by adding the two attribute outcomes that we want to investigate in the global entity instance. In this case we will ask for value-only if the attribute is known and a full decision report if the value is unknown. The entire Assess request now looks like:

Request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:assess-request>
      <typ:global-instance>
        <typ:attribute id="eligible_teenage_allowance" known-outcome-style="value-only" unknown-outcome-style="decision-report"/>
        <typ:attribute id="eligible_low_income_allowance" known-outcome-style="value-only" unknown-outcome-style="decision-report"/>
      </typ:global-instance>
    </typ:assess-request>
  </soapenv:Body>
</soapenv:Envelope>
```

```
</typ:global-instance>
</typ:assess-request>
</soapenv:Body>
</soapenv:Envelope>
```

If we execute this request now, each attribute outcome will be unknown, with a decision report telling us which attributes require values in order to reach a determination for the given outcome.

Response:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:assess-response>
      <typ:global-instance>
        <typ:attribute id="eligible_low_income_allowance" type="boolean" inferred="false">
          <typ:unknown-val/>
          <typ:decision-report report-style="decision-report">
            <typ:attribute-node id="dn:0" entity-id="global" instance-id="global" hypothetical-instance="false" attrib-
ute-id="eligible_low_income_allowance" type="boolean" text="Is the claimant eligible for low income allowance?"
inferred="false">
              <typ:unknown-val/>
              <typ:attribute-node id="dn:1" entity-id="global" instance-id="global" hypothetical-instance="false"
attribute-id="claimant_income" type="currency" text="The claimant's annual income is unknown." inferred-
d="false">
                <typ:unknown-val/>
              </typ:attribute-node>
              <typ:attribute-node id="dn:2" entity-id="global" instance-id="global" hypothetical-instance="false"
attribute-id="claimant_public_housing_client" type="boolean" text="Is the claimant a public housing client?" inferred-
d="false">
                <typ:unknown-val/>
              </typ:attribute-node>
              <typ:attribute-node id="dn:3" entity-id="global" instance-id="global" hypothetical-instance="false"
attribute-id="claimant_date_of_birth" type="date" text="The claimant's date of birth is unknown." inferred="false">
                <typ:unknown-val/>
              </typ:attribute-node>
            </typ:attribute-node>
          </typ:decision-report>
        </typ:attribute>
        <typ:attribute id="eligible_teenage_allowance" type="boolean" inferred="false">
```

```

    <typ:unknown-val/>
    <typ:decision-report report-style="decision-report">
      <typ:attribute-node id="dn:0" entity-id="global" instance-id="global" hypothetical-instance="false" attribute-id="eligible_teenage_allowance" type="boolean" text="Is the claimant eligible for the teenage child allowance?" inferred="false">
        <typ:unknown-val/>
        <typ:relationship-node id="dn:1" source-entity-id="global" source-instance-id="global" hypothetical-instance="false" target-entity-id="child" relationship-id="claimantschildren" state="unknown" inferred="false"> </typ:relationship-node>
      </typ:attribute-node>
    </typ:decision-report>
  </typ:attribute>
</typ:global-instance>
</typ:assess-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

From the decision report we can see that for the `eligible_low_income_allowance` attribute to have a value, we need to provide answers for the following attributes:

- `claimant_income` (a currency value)
- `claimant_public_housing_client` (a boolean value)
- the claimant's date of birth (a date value)

We can do that by adding the following to the global entity instance.

```

    <typ:attribute id="claimant_income">
      <typ:number-val>13000</typ:number-val>
    </typ:attribute>
    <typ:attribute id="claimant_public_housing_client">
      <typ:boolean-val>true</typ:boolean-val>
    </typ:attribute>
    <typ:attribute id="claimant_date_of_birth">
      <typ:date-val>1981-03-22</typ:date-val>
    </typ:attribute>

```

According to the decision report for the `eligible_teenage_allowance` goal, the relationship `'claimantschildren'` needs to be known. As this is a containment relationship, this can be done by adding some instances of the `'child'` entity to the global entity instance:

```

<typ:entity id="child">
  <typ:instance id="child_1">
    <typ:attribute id="child_age">
      <typ:number-val>9</typ:number-val>
    </typ:attribute>
  </typ:instance>
  <typ:instance id="child_2">
    <typ:attribute id="child_age">

```

```
<typ:number-val>5</typ:number-val>
</typ:attribute>
</typ:instance>
</typ:entity>
```

Our Assess request should now look like the request below. Now the response (see below) provides the following answers for our goal:

eligible_teenage_allowance is false (the claimant is not eligible because neither of their children is between 13 and 19 years of age)
eligible_low_income_allowance is true (the claimant is eligible because they are a public housing client, they have an income below 20,000, and claimant date of birth is known).

Assess Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:assess-request>
      <typ:global-instance>
        <typ:attribute id="eligible_teenage_allowance" known-outcome-style="value-only" unknown-outcome-style="decision-report"/>
        <typ:attribute id="eligible_low_income_allowance" known-outcome-style="value-only" unknown-outcome-style="decision-report"/>
        <typ:attribute id="claimant_income">
          <typ:number-val>13000</typ:number-val>
        </typ:attribute>
        <typ:attribute id="claimant_public_housing_client">
          <typ:boolean-val>true</typ:boolean-val>
        </typ:attribute>
        <typ:attribute id="claimant_date_of_birth">
          <typ:date-val>1981-03-22</typ:date-val>
        </typ:attribute>
        <typ:entity id="child">
          <typ:instance id="child_1">
            <typ:attribute id="child_age">
              <typ:number-val>9</typ:number-val>
            </typ:attribute>
          </typ:instance>
          <typ:instance id="child_2">
            <typ:attribute id="child_age">
              <typ:number-val>5</typ:number-val>
            </typ:attribute>
          </typ:instance>
        </typ:entity>
      </typ:global-instance>
```

```
</typ:assess-request>
</soapenv:Body>
</soapenv:Envelope>
```

Assess Response

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:assess-response>
      <typ:global-instance>
        <typ:attribute id="eligible_low_income_allowance" type="boolean" inferred="true">
          <typ:boolean-val>true</typ:boolean-val>
        </typ:attribute>
        <typ:attribute id="eligible_teenage_allowance" type="boolean" inferred="true">
          <typ:boolean-val>>false</typ:boolean-val>
        </typ:attribute>
        <typ:attribute id="claimant_income" type="currency">
          <typ:number-val>13000.0</typ:number-val>
        </typ:attribute>
        <typ:attribute id="claimant_date_of_birth" type="date">
          <typ:date-val>1981-03-22</typ:date-val>
        </typ:attribute>
        <typ:attribute id="claimant_public_housing_client" type="boolean">
          <typ:boolean-val>true</typ:boolean-val>
        </typ:attribute>
        <typ:entity id="child" inferred="false">
          <typ:instance id="child_1">
            <typ:attribute id="child_age" type="number">
              <typ:number-val>9.0</typ:number-val>
            </typ:attribute>
          </typ:instance>
          <typ:instance id="child_2">
            <typ:attribute id="child_age" type="number">
              <typ:number-val>5.0</typ:number-val>
            </typ:attribute>
          </typ:instance>
        </typ:entity>
      </typ:global-instance>
```

```
</typ:assess-response>  
</SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

Select the rulebase language a web service uses

Both the Assess and Interview Services allow the language to be used for the assessment/interview to be specified by the requestor. This is done by adding the optional international element to the SOAP Header element of the request. This complies with the Web Services Internationalization standard – WS-I18n (<http://www.w3.org/TR/ws-i18n/>).

Note: Specifying the language to use for a particular interview or assessment only affects the attribute text for attributes, entities and relationships (in decision reports or on screens) and screen items such as captions. It does **not** affect the input or output format for attribute values.

Determining which languages are supported by the rulebase

The list of languages supported by a particular rulebase is listed in the response for the ListRulebases operation in the Server Service, under the 'available-languages' element of the rulebase. For example:

```
<typ:rulebase name="EmployeeLeave">
  <typ:available-languages>
    <typ:language>en_US</typ:language>
    <typ:language>fr_BE</typ:language>
  </typ:available-languages>
  <typ:build-time>2013-07-25T19:03:59Z</typ:build-time>
  <typ:policy-modeling-version>10.4.6.1</typ:policy-modeling-version>
  <typ:available-services>
    <typ:service name="odsInterviewService">http://localhost:5111/ds-102-net/in-
    terview/soap/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsInterviewService104">http://localhost:5111/ds-102-net/in-
    terview/soap/10.4/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsInterviewService102">http://localhost:5111/ds-102-net/in-
    terview/soap/10.2/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsAssessServiceSpecific">http://localhost:5111/ds-102-net/assess/soap/spe-
    cific/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsAssessServiceGeneric">http://localhost:5111/ds-102-net/assess/soap/-
    generic/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsAssessServiceSpecific104">http://localhost:5111/ds-102-net/assess/soap/spe-
    cific/10.4/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsAssessServiceGeneric104">http://localhost:5111/ds-102-net/assess/soap/-
    generic/10.4/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsAssessServiceSpecific103">http://localhost:5111/ds-102-net/assess/soap/spe-
    cific/10.3/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsAssessServiceGeneric103">http://localhost:5111/ds-102-net/assess/soap/-
    generic/10.3/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsAssessServiceSpecific102">http://localhost:5111/ds-102-net/assess/soap/spe-
    cific/10.2/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsAssessServiceGeneric102">http://localhost:5111/ds-102-net/assess/soap/-
    generic/10.2/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsAssessServiceSpecific10">http://localhost:5111/ds-102-net/assess/soap/spe-
    cific/10.0/EmployeeLeave.asmx?wsdl</typ:service>
    <typ:service name="odsAssessServiceGeneric10">http://localhost:5111/ds-102-net/assess/soap/-
    generic/10.0/EmployeeLeave.asmx?wsdl</typ:service>
```

```
</typ:available-services>
</typ:rulebase>
```

This section of the response indicates that the EmployeeLeave rulebase has two available languages, American English (en_US) or Belgian French (fr_BE).

This rulebase can be found in the OPA runtime directory at: `examples\rulebases\compiled\EmployeeLeave.zip`

Specifying the language to use for an Assessment or Interview

As mentioned above, the locale can be specified in any request to the Determinations Server, by adding the international element to the SOAP Header element, and adding the locale sub-element. The locale's value will be in the form of a two letter ISO 693-1 language code followed by the two letter ISO 3166-1 country code separated by an underscore ('_'), that is:

```
<international xmlns="http://www.w3.org/2005/09/ws-i18n">
  <locale>language-locale</locale>
</international>
```

Assess service

Specifying the WS-i18n locale element in the request of an assess operation will cause any attribute, relationship and entity text returned in a decision report to be in the specified language. A locale may also be specified in a ListGoals request which will affect the text of the goals returned. If no locale is specified for each of these requests, the default language for the rulebase will be used.

Interview service

The language used for an interview session can be configured by specifying the locale in the OpenSession operation. If no locale is specified, the session will be created using the default language for the rulebase.

For all other operations, if the locale is specified it must match the locale used to open the session or an error is returned. If no locale is specified, the language used will be that of the session.

Update a Determinations Server rulebase

What do you want to do?

[Deploy an updated rulebase to Determinations Server](#)

[Understand how existing requests are affected when a rulebase is updated](#)

Deploy an updated rulebase to Determinations Server

See the [Deploy Determinations Server](#) topic.

Understand how existing requests are affected when a rulebase is updated

When you update a rulebase in the existing Determinations Server, any changes you have made to the rulebase will come into effect. All Determinations Server operations will run against this new rulebase, so you will have to take into account new or renamed attributes and relationships, deleted attributes and relationships, new or changed rules that may effect the values of goals.

If you are using the Generic Assess service or the Interview service, the interface will not change. However, you may need to change the content of your request(s) based on the data model or screen changes made to the rulebase.

If you are using the Specific Assess service, the WSDL will be changed based on the data model changes made to the rulebase. If you have an automatically compiled .NET or Java interface to the specific web service, you will need to recompile it against the new Specific Assess WSDL.

Interviews

Topics in "Interviews"

- Upgrade an interview experience from a previous version
- Deploy Web Determinations
- Manage rulebases
- Configure the appearance of a Web Determinations interview
- Apply a visual theme to Web Determinations
- Change the appearance or behavior of controls in Web Determinations
- Change the appearance or behavior of screens in Web Determinations
- Configure appearance of Web Determinations by locale
- Secure a Web Determinations deployment
- Save and resume interviews in Web Determinations
- Integrate Web Determinations with another application
- Install and register Web Determinations plug-ins
- Create a custom interview user experience
- Customize the Interview Portlet

Upgrade the interview experience from a previous version

What do you want to do?

[Upgrade from Interactive for Java](#)

[Upgrade from Interactive for .NET](#)

Upgrade from Interactive for Java

The following describes what needs to be done to effect a migration from Haley Interactive (JRBI) to Oracle Web Determinations.

Recompile the rules

One of the first things you will need to do if you are migrating from Haley Interactive to Oracle Web Determinations, is recompile your rules; to do this:

1. Open the rulebase in Oracle Policy Modeling; you will be taken through the upgrade wizard.
2. Build and debug, with screens; your rulebase should function with default styling, allowing you to make determinations.

Screen order and screen flows:

Visio screen flows are no longer supported, and have been replaced by the much simpler integrated screen flows. To update existing rulebases that contain a data review screen, do the following:

1. If the rulebase has a data review screen, open it and check "default screen order" to make screens appear in the same order as the data review screen; this also enables a progress bar.
2. Any Visio screenflows that are more complicated than the default screen ordering, should be rebuilt in the new screen flow editor.

Commentary

Oracle Web Determinations has a different commentary model from Interactive, with commentary deployed and stored with the rulebase itself. Web Determinations also supports screen commentary, which was previously unavailable.

1. Build commentary files: if you wish, you can generate new screen commentary files. If you want to migrate only the attribute commentary that you previously had in Interactive, leave only "base-level attributes on screens" checked. After generating the files click the **Yes** button to open the newly created directory with attribute commentary files.
2. Synchronize your existing attribute commentary directory into the new directory by renaming the existing attribute commentary files from *.htm* to *.html* if necessary, so that the file names match exactly that which was built by Oracle Policy Modeling. Existing 'no help' or commentary 404 error pages can be ignored; Web Determinations will not show a commentary link where there is no corresponding commentary file to link to.
3. Any links in the commentary itself, including CSS file links, are now relative to the commentary URL, not the location of the commentary html file. For example, if you previously had a commentary html file that included:

```
<link rel=stylesheet href="commentary.css"> 
```

you will need to move the relevant files to:

```
<project>Release\web-determinations\WEB-INF\classes\resources\commentary.css  
<project>Release\web-determinations\WEB-INF\classes\images\foo.jpg
```

and you'll need to change the references in the commentary HTML file to:

```
<link rel=stylesheet href="../../resources/commentary.css">

```

IsHTML images

modify all URLs in isHTML.:

- For images on a summary screen, change
`src="images/foo/bar.jpg"`
to
`"../../images/foo/bar.jpg"`
(`"images"` is now the mandatory image file location, where before it was a reasonable default location)
- For question screens, add an extra `../` to the beginning; for example:
` \Release\web-determinations\WEB-INF\classes\images`

IsHTML commentary

- For per-word commentary on summary screen, change `href=` to `"../../commentary/<compiled rulebase name>/<-locale>?target=<subdirectory and filename WITHOUT extension>"`
- For question screens, add an extra `../` to the beginning.
- Copy any remaining (non-attribute) commentary HTML files over to the new commentary directory in:
`<project> \Development\include\commentary\<locale>\<whatever>.html`

For example, if your rulebase is named *myRulebase* and has the default US-English locale, you might have a control text of:
`How many exemptions can you claim?`

You should then move your commentary file to:

```
<project> \Development\include\commentary\en-US\perwordstuff\exemptions-help.html
```

Visual styles

This is a complex manual process because Web Determinations uses a completely different rendering process than was used by Interactive.

You are likely to find that there are small differences in the shared portions of code (headers, footers, and so on) between the various customized pages in your Interactive project. These will need to be reconciled, exceptions made according to screen name or screen type where necessary, JSP code translated to velocity template code, and placed into the various templates (header.vm, footer.vm, question_screen.vm, and so on) in the Web Determinations structure.

Another difference that you will need to account for is that Web Determinations has shifted to a completely CSS-based layout in place of the table-based layout used by Interactive.

Change the progress bar from textual to graphical:

1. Verify you have the textual progress bar displaying correctly.
2. If the stages displayed are not what you like, rearrange your question screens into folders with the names (and in the order) that you like.
3. Rename (or create) graphics for the progress stages, where the names of the graphics files are based on the exact text of the progress stages. For example, if you have a progress stage with text, "Household" then create two graphics, "Household_active.jpg" and "Household_inactive.jpg"
4. Modify the includes/stages.vm velocity template to create `` tags based on the stage names, instead of just printing the stage names directly.

See also:

[isHTML and Web Determinations customization](#)

Upgrade from Interactive for .NET

A major feature of Oracle Policy Automation 10.0 is that it provides support for Web Determinations on the .NET platform for the first time since 8.5. In that time significant changes have been made to the product suite which means that the upgrade path from RBI .NET 8.5 to Web Determinations is dependent on the specific needs of the individual project.

The general things to consider when upgrading are:

- The rulebase project itself will need to be upgraded and re-compiled in Oracle Policy Modeling.
- Any look and feel changes that have been made to the RBI via XML, HTML and/or CSS must be ported to the stylesheets and velocity templates provided by Web Determinations. For more information see [Change the look and feel](#), in the [Configure the appearance of a Web Determinations interview](#) topic.
- Any code customizations must be re-implemented using the extension framework provided by Web Determinations (see the topic, [Introduction to Web Determinations extensions](#)).

Deploy Web Determinations

For information regarding the deployment of Oracle Web Determinations for Java and .Net, refer to the *Oracle Policy Automation Installation Guide*.

For information regarding the installation and registration of Web Determinations extensions, refer to the topic, [Install and register Web Determinations extensions](#).

Manage rulebases

What do you want to do?

- [Manage multiple rulebases](#)
- [Hot-swap rulebases](#)

Manage multiple rulebases

Oracle Web Determinations is capable of serving multiple rulebases through the same instance of Web Determinations, meaning that it is necessary to configure the look and feel and the extension just once, after which they will automatically be applied to each rulebase.

To do this, simply deploy each rulebase zip file to the location specified in the configuration file and start the application. When you open a web browser, the default page of your Web Determinations deployment will display the list of rulebases that have just been deployed.

Hot-swap rulebases

Instead of taking the application offline to deploy rulebase changes, you can use hot swapping to deploy updated rulebases. This requires the following parameters in the configuration file (application.properties) to be set as follows:

```
load.rulebase.as.resource = false
rulebase.path = /WEB-INF/classes/rulebases
cache.loaded.rulebases = false
```

Note: The rulebase path may not work correctly if the path is relative to the file system rather than relative to the application root (such as on WebLogic on Linux). In this case, the leading "/" should be removed.

To test this, do the following:

1. Start up the application and pick one of the deployed rulebases; perform a simple investigation.
2. Open the same project in Oracle Policy Modeling and make a few changes to the layout of the summary screen.
3. Recompile.
4. Copy the recompiled rulebase zip file to the rulebase directory of the still running Web Determinations deployment, overwriting the existing zip of that rulebase.
5. Start a new investigation on that same rulebase.
6. Pull up the summary screen; you will now see the changes that were just made.

Rulebase hot-swapping on .NET

By default, rulebases are set up to be deployed to the 'bin/rulebases' directory of the web application. However, adding, deleting or modifying the files contained in this directory can lead to issues due to the way that ASP .NET recycles application domains. In short, changing the contents of a virtual directory at runtime may cause ASP .NET reload the whole application which will have the effect of destroying any active interview sessions. Therefore, if you are intending to use the rulebase hot-swapping feature, it is strongly advised that you change the location of your rulebase directory to one that is located outside the application's virtual directory. This can be done by:

1. Creating a directory located outside the application's virtual directory.
2. Setting the permissions on this directory to enable it to be read by the application. Generally this involves granting read permissions to the ASPNET user.
3. Changing the 'rulebase.path' property in the 'application.properties' file to the absolute path of the directory created in step 1

Configure the appearance of a Web Determinations interview

The appearance of the Web Determinations interview is controlled by what is termed *skinning and theming*. Essentially, this is the ability to configure the look and feel of the user interface to suit a particular deployment's needs, which could be anything from simply changing a few colors and logos to a complete re-design of the layout and rendering of every screen in the application.

Customization can be managed at a number of levels ranging in complexity as follows:

1. **Properties** (*appearance.country-locale.properties*)
The default properties containing display text and logos can be changed to alter the basic look and feel of your interview.
2. **CSS** (*main.vm.css*)
The core page structure is a pure CSS layout. Simple look and feel/theming changes can be achieved by using CSS alone, but when used in conjunction with property changes, you can also effect changes to positioning, text styles, colors and layout.
3. **Page Structure and CSS** (**.vm*)
The core page structure is put in place using velocity templates to directly edit both HTML and CSS content directly, allowing deeper and more complex customization and extension of the web experience. Velocity templates take the form of *page.vm* or *page_css.vm* and are used to change text, logo's, colors, text styles, layout, core html structure, provide javascript and other rich media extension, customize form behavior as well as any html generated content. Documentation is available at <http://velocity.apache.org/engine/devel/user-guide.html>.

What do you want to do?

Change the look and feel

Understand IsHTML and Web Determinations customization

Change the appearance of a web interview control

Change the appearance of only one screen type

Localize the interview

Change the look and feel

To change the layout and styling of your Web Determinations interview, do the following:

1. Locate the web determinations main folder as follows:

Tomcat (Java) – default is the Web Determinations web application in the Tomcat webapps folder; for example:
`C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\classes\`

IIS (.Net) – default is in:
`C:\inetpub\web-determinations\bin\`

2. Modify the following to change the styling, layout, and web widgets (images, and so on) to match corporate branding:
 - a. Modify the Velocity templates in the `templates\includes` folder to change the layout of the interview screens. For more information about working with Velocity templates, see the [Velocity Templates Developer Guide](#).
 - b. Modify the `templates\main.vm.css` to change CSS-controlled HTML styles, and also `configuration/appearance.[locale].properties` for other styles and widget controls.

- c. Add web widgets such as images or flash objects that are used by the templates and styles in **images** and **resources** folders.
3. Make modifications to the text contents of the pages/templates in **configuration/messages.[locale].properties**.
4. Save the changes; note that by default, changes to templates and properties files do not require a server restart, but the Web Determinations application server can be configured to load and cache them (**configuration/application.properties**) in which case a restart is required.

Change the appearance of a web interview control

The following steps describe how to change the appearance of all web interview controls of a particular type; for example, a list box:

1. Locate the web determinations main folder as follows:

Tomcat (Java) – default is the Web Determinations web application in the Tomcat webapps folder; for example:
C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\classes

IIS (.Net) – default is in:
C:\inetpub\web-determinations\bin

2. Go to the **templates** folder and locate the template for the control to be modified; see [Velocity Templates Developer Guide](#) for a description of each of the available predefined templates.
3. Make modifications to the text associated with the control in **configuration/messages.[locale].properties**.
4. Save the changes; note that by default, changes to templates and properties files do not require a server restart, but the Web Determinations application server can be configured to load and cache them (**configuration/application.properties**) in which case a restart is required.

Change the appearance of only one screen type

The following steps describe how to change the appearance of screens of a particular type a particular type; for example, an investigation screen

1. Locate the web determinations main folder as follows:

Tomcat (Java) – default is the Web Determinations web application in the Tomcat webapps folder; for example:
C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\classes

IIS (.Net) – default is in:
C:\inetpub\web-determinations\bin

2. Go to the **templates** folder and locate the template for the screen type to be modified; see [Velocity Templates Developer Guide](#) for a description of each of the available predefined templates.
3. Make the desired modifications to the screen template.
4. Save the changes; note that by default, changes to templates and properties files do not require a server restart, but the Web Determinations application server can be configured to load and cache them (**configuration/application.properties**) in which case a restart is required.

Localize the interview

The following steps describe how to localize the interview:

1. Locate the web determinations main folder as follows:

Tomcat (Java) – default is the Web Determinations web application in the Tomcat webapps folder; for example:

`C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\classes\`

IIS (.Net) – default is in:

`C:\inetpub\web-determinations\bin\`

2. Create a new `messages.[locale].properties` file in the `configuration` folder, where the `[locale]` is in the standardized format `lang-local`, for example, `en-GB`.
3. Repeat the above for `appearance.[locale].properties` in the `properties` folder.
4. Save the changes; note that by default, changes to templates and properties files do not require a server restart, but the Web Determinations application server can be configured to load and cache them (`properties/application.properties`) in which case a restart is required.

See also:

[Configure appearance of Web Determinations by locale](#)

Apply a visual theme to Web Determinations

What do you want to do?

Use a corporate logo or background image

Apply a corporate color scheme

Change the header or footer of interview screens

Use a corporate logo or background image

To display your corporate logo in a Web Determinations interview, do the following:

1. Locate the web determinations main folder as follows:

Tomcat (Java) – default is the Web Determinations web application in the Tomcat webapps folder; for example:

`C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\classes\`

IIS (.Net) – default is in:

`C:\inetpub\web-determinations\bin\`

2. Add the new logo in the `images` folder.
3. Change the default logo in `properties\appearance.locale.properties` to refer to the new image.
4. Save the changes; note that by default, changes to templates and properties files do not require a server restart, but the Web Determinations application server can be configured to load and cache them (`properties\application.properties`) in which case a restart is required.

Apply a corporate color scheme

To apply your corporate color scheme to a Web Determinations interview, do the following:

1. Locate the Web Determinations main folder as follows:

Tomcat (Java) – default is the Web Determinations web application in the Tomcat webapps folder; for example:

`C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\classes\`

IIS (.Net) – default is in:

`C:\inetpub\web-determinations\bin\`

2. Change the default colors in `properties\appearance.[locale].properties` to match your corporate colors.
3. Modify the following to change the styling, layout, and web widgets (images, and so on) to match corporate branding:
 - a. Modify the Velocity templates in the `templates\includes` folder to change the layout of the interview screens. For more information about working with Velocity templates, see the [Velocity Templates Developer Guide](#).
 - b. Modify the `templates\main.vm.css` to change CSS-controlled HTML styles, and also `properties\appearance.[locale].properties` for other styles and widget controls.
 - c. Add web widgets such as images or flash objects that are used by the templates and styles in the `images` and `resources` folders.

4. Save the changes; note that by default, changes to templates and properties files do not require a server restart, but the Web Determinations application server can be configured to load and cache them ([configuration/application.properties](#)) in which case a restart is required.

Change the header or footer of interview screens

To change the header and footer on your interview screens, do the following:

1. Locate the Web Determinations main folder as follows:

Tomcat (Java) – default is the Web Determinations web application in the Tomcat webapps folder; for example:

`C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\classes\`

IIS (.Net) – default is in:

`C:\inetpub\web-determinations\bin\`

2. Modify the header and footer Velocity templates as required, in the [templates/includes](#) folder. For more information about working with Velocity templates, see the [Velocity Templates Developer Guide](#).
3. Save the changes; note that by default, changes to templates and properties files do not require a server restart, but the Web Determinations application server can be configured to load and cache them ([configuration/application.properties](#)) in which case a restart is required.

Change the appearance or behavior of controls in Web Determinations

What do you want to do?

[Change the appearance of all controls of a particular type](#)

[Change the appearance of a specific screen control](#)

[Implement a custom control](#)

Change the appearance of all controls of a particular type

For information about changing the appearance of all controls of a particular type, refer to [Change the appearance of a web interview control](#) in the topic [Configure the appearance of a Web Determinations interview](#).

Change the appearance of a specific screen control

There are two ways in which the appearance of a specific screen control can be change; they are as follows:

1. Through Oracle Policy Modeling, using the Screen Authoring option, in which the author can customize how the input control of a selected attribute is displayed by using either or all of the following: IsHTML, CSS Style, CSS Class, Custom Properties.
2. Find the template of the control ([templates](#)), and add logic via Velocity and session information so that if the current input control is for a certain attribute name (for example, `child_name`) then display a different HTML codeblock

See also:

[IsHTML and Web Determinations customization](#)

Implement a custom control

Custom controls are defined by implementing the **CustomControl** interface. This is an empty marker interface that extends from the base **Control** interface and which defines a variety of methods that controls have to implement.

Additionally, there is a **CustomInputControl** interface to be implemented if your custom control is intended to collect user input data that maps to at least one rulebase attribute. Ensure that your implementation matches the following requirements:

- The class name you have chosen for your custom control implementation does not conflict with any of the native controls.
- You have provided a template to render your custom control named *[custom control class name].vm*.
- You have implemented the **getControlType() : String** method to return the class name of your control.

See also:

[Custom Screen and Custom Control Provider plugins](#)

[Create a Custom Control](#)

[Custom Control - BenefitCode walkthrough example](#)

Change the appearance or behavior of screens in Web Determinations

What do you want to do?

[Change the appearance of a specific screen](#)

[Implement a custom screen](#)

Change the appearance of a specific screen

Find the template of the screen (see [Template files](#)), and add logic via Velocity and session information so that if the current screen has a certain name (for example, `child_details`) then display a different HTML codeblock.

Implement a custom screen

For information on implementing custom screens, refer to the following topics:

- [Custom Screen and Custom Control Provider plugins](#)
- [Create a Custom Screen](#)
- [Create a Custom Screen example](#)

Configure appearance of Web Determinations by locale

What do you want to do?

[Understand how Web Determinations works with locales](#)

[Apply a language specific theme to Web Determinations](#)

[Change the text displayed for a specific language](#)

Understand how Web Determinations works with locales

Interview Engine versus Web Determinations

When discussing localization in Oracle Web Determinations, it is important to distinguish between what is the concern of the Interview Engine and that of any clients to that engine such as the Web Determinations platform. In simple terms the Web Determinations client is responsible for localizing everything that is not authored as a part of the rule-base. Further, all Web Determinations and rule engine sessions are created in a specific locale and it is assumed that anything coming out of the rule engine session will be localized according to that locale.

Examples of what the Interview Engine is responsible for include:

- Attribute text, sentence forms, decision reports and so on.
- Controls and screens authored within Oracle Policy Automation.
- The display format of attribute values.
- Warning and Error daemon text.

Examples of what Web Determinations is responsible for:

- Accepted attribute input format(s).
- Any text on screens or screens themselves that are not authored inside Oracle Policy Automation; for example, copyright statements, button text, menu text.
- Error and warning messages, excluding the text of error and warning messages that are authored in Oracle Policy Automation (for example, daemons).
- Templates and style sheets used for rendering screens.

Select the session locale

Select the locale for an interview session

Once an interview session is started in Web Determinations for a given rulebase, the locale that was used to create that session controls the culture of the screens that are displayed for the duration of that interview. There are two ways of selecting the locale for a session:

1. Select the locale on the locale selection screen that is presented after the user selects the rulebase they wish to use. In the case where the selected rulebase is only available in one locale, Web Determinations will bypass the locale selection screen and automatically start an interview session in that locale.

2. By specifying the rulebase and locale to use via the URL This method allows investigations to automatically be started for a given rulebase in a given locale, thereby bypassing both the rulebase and locale selection screen. The format of the URL is:

```
http://<server>:<port>/<webapp>/startsession/<rulebase>/<locale>
```

For example, to start a session in the 'en-US' locale for the rulebase 'MyRulebase' the URL would look something like:

```
http://server:8080/web-determinations/startsession/MyRulebase/en-US
```

Specifying the default locale

Whilst the vast majority of screens in Web Determinations come from the rulebase, there are a few that are not bound to an interview, most notably the rulebase and locale selection screens. In this case, the locale that is used is defined by the 'default.locale' property in the [application.properties](#) configuration file.

Localizing error and warning messages

For information on the localization of error and warning messages, go to the topic below, [Change the text displayed for a specific language](#).

See also:

[messages.<locale>.properties file](#)

Localizing Input and Output formats

Web Determinations allows the accepted input and output format(s) of Date, DateTime, Currency and Number attributes to be specified on a per locale basis using its default formatter. The relevant configuration properties are located in the [message.properties](#) file.

File Encoding

It is imperative that all localized files are saved with **UTF-8** encoding. Failure to do this will cause errors when the properties located in non-UTF-8 encoded files are loaded and retrieved.

Locale codes

The two letter ISO 639-1 code is used to identify any supported language. Where there is the need to support multiple variations of any given language, this may be followed by the two letter ISO 3166-1 country code separated by a hyphen.

Out of the box we provide localizations for the following languages:

Language	Locale Code
Arabic (Saudi Arabia)	ar
Chinese (Simplified)	zh-CN
Chinese (Traditional)	zh-HK
Czech	cs
Danish	da
Dutch	nl

Language	Locale Code
English (American)	en
English (Great Britain)	en-GB
Finnish	fi
French	fr
German	de
Hebrew	he
Italian	it
Japanese	ja
Korean	ko
Polish	pl
Portuguese (Brazilian)	pt-BR
Portuguese (European)	pt-PT
Russian	ru
Spanish (Modern)	es
Swedish	sv
Thai	th
Turkish	tr

Apply a language specific theme to Web Determinations

Web Determinations provides the ability to change the look and feel of the application on a per locale basis. The appearance of the application is controlled by the [appearance properties](#). By default, Web Determinations only provides one set of appearance properties, however, these can be overridden on a per locale basis by copying the *appearance.properties* file, changing the name to *appearance.<locale>.properties* and then editing the property values as required.

Change the text displayed for a specific language

Localization of error and warning messages is, for the most part, the responsibility of the Oracle web Determinations platform. Only error and warning messages that implement either *com.oracle.web-determinations.engine.data.error.Error* or *com.oracle.web-determinations.engine.data.error.Warning* are localized in Web Determinations; uncaught exceptions and other non-recoverable errors are handled differently.

Within the Oracle Web Determinations platform, localization is performed through the message service which is responsible for loading the correct message bundle for the specified locale and returning the message for the specific error or warning to be localized. The message bundle itself is a properties file called *messages.<locale>.properties*. The location of this file is specified by the 'messages.path' property in the configuration file.

The message service also utilizes the Velocity Templating Engine to allow substitution of current property values in the error message; for example, if you wanted to display the attribute ID and the actual value that was entered in an invalid value error, you could do so by setting your error property as:

```
AttributeValueError = The value ${message.value} is not valid for attribute ${message.attributeId}
```

The localization settings for displayed text in Web Determinations are managed in the [messages.<locale>.properties file](#).

Secure a Web Determinations deployment

Oracle Web Determinations does not have any in-built support for authentication or secure access. To configure secure access, Web Determinations relies upon the security framework of the host Applications Server. Some pointers and procedures for configuring secure access for some application servers can be found below:

SSL on IIS 6 and IIS 7

The Microsoft IIS Community documentation for enabling SSL on IIS 7 can be found at <http://learn.iis.net/page.aspx/144/how-to-setup-ssl-on-iis-70/>

Configuration for IIS 6 is very similar.

While those instructions list several methods for configuring SSL access, this is a summary of the "IIS Manager" section intended to assist you to quickly configure a test server using a self-signed certificate. For a production server, you will want to thoroughly familiarize yourself with your server documentation.

Before starting these instructions, first verify that you have a working Web Determinations installation without SSL; point your web browser at the URL at which you have configured Web Determinations. This may be

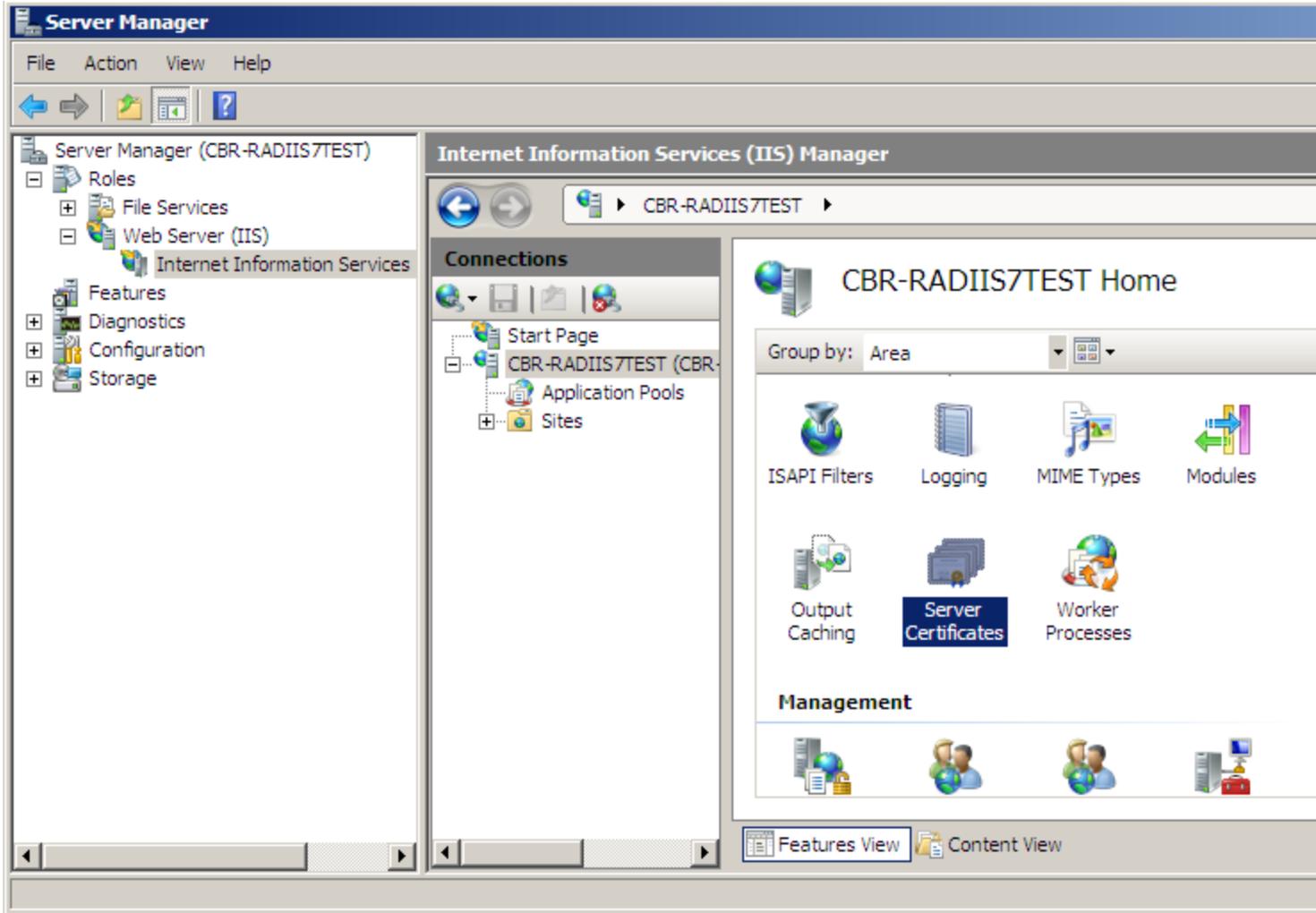
<http://localhost/web-determinations>

If you are not yet able to connect normally to Web Determinations, please refer to the topic *Deploying Oracle Web Determinations for .NET on IIS* in the *Oracle Policy Automation Installation Guide* before proceeding to enable SSL.

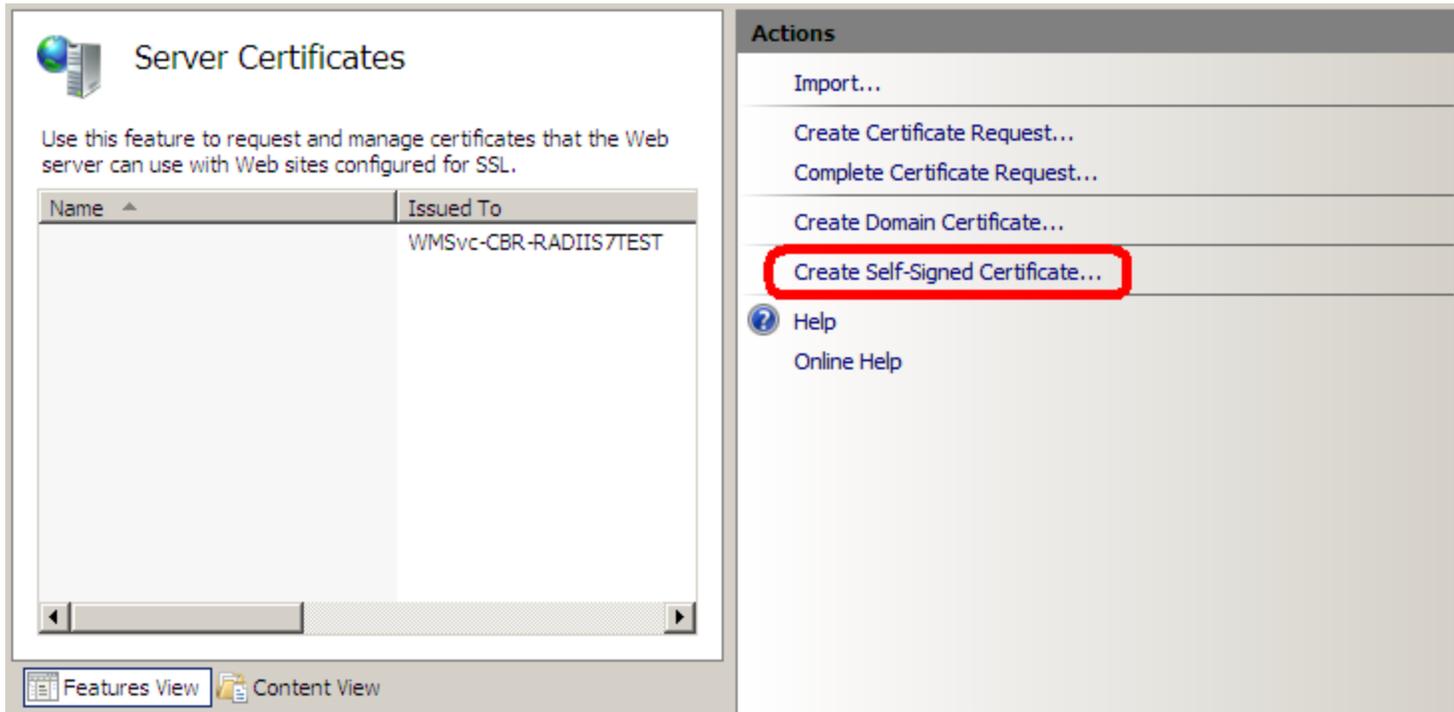
To enable SSL on a default IIS 7 setup there are two key steps:

Step 1 - Create a certificate (if you already have a certificate configured in IIS, you may skip this step)

Create a certificate using the IIS Manager's Server Certificates panel, which can be reached by navigating the trees of the Server Manager as shown here:



Open the *Server Certificates* feature and select the *Create Self-Signed Certificate...* task shown outlined below; for production servers, you will most likely want to purchase or import a properly signed certificate using the *Import...* task instead, but using a self-signed certificate will allow you to proceed immediately with configuration and testing.

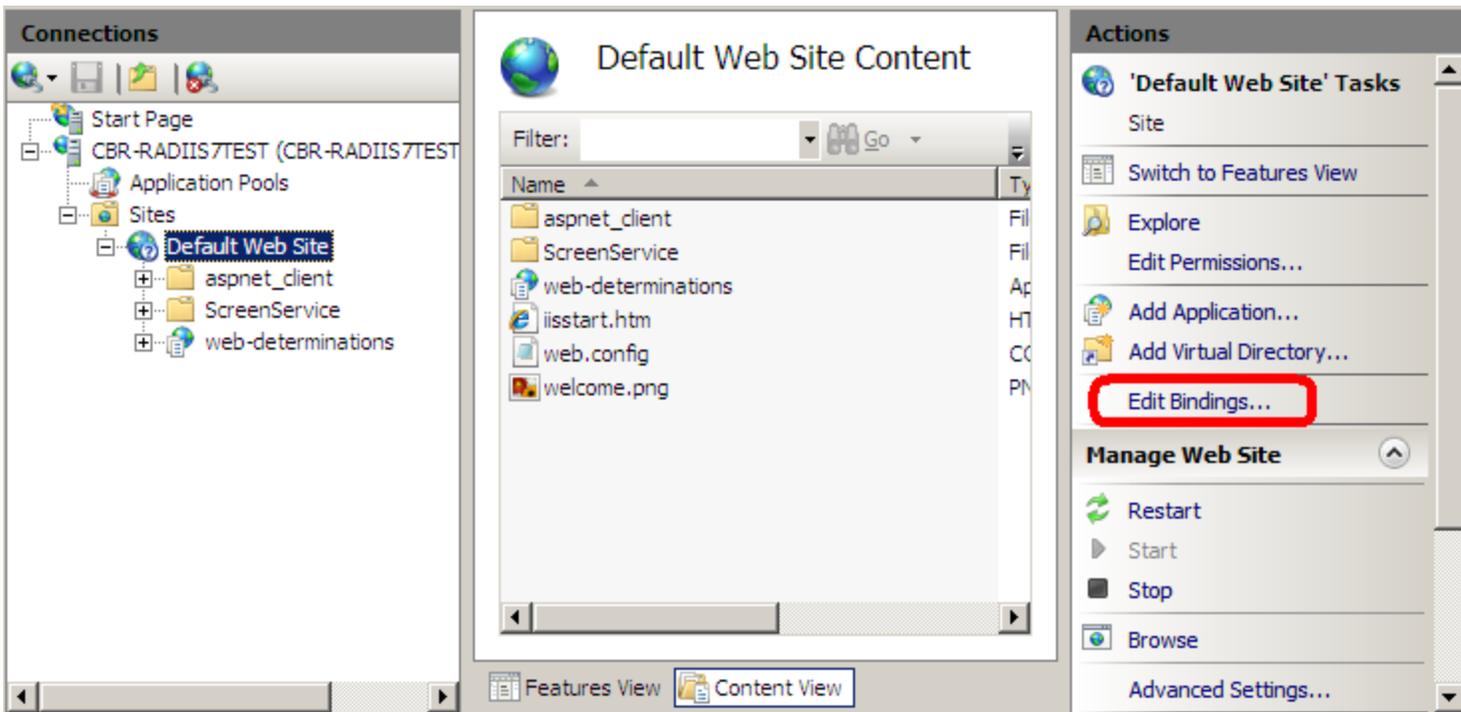


You will be asked for a friendly filename; for the purposes of this example we will use *web-determinations*, and click **OK** to complete the self-signed certificate.

Step 2 - Create an SSL binding

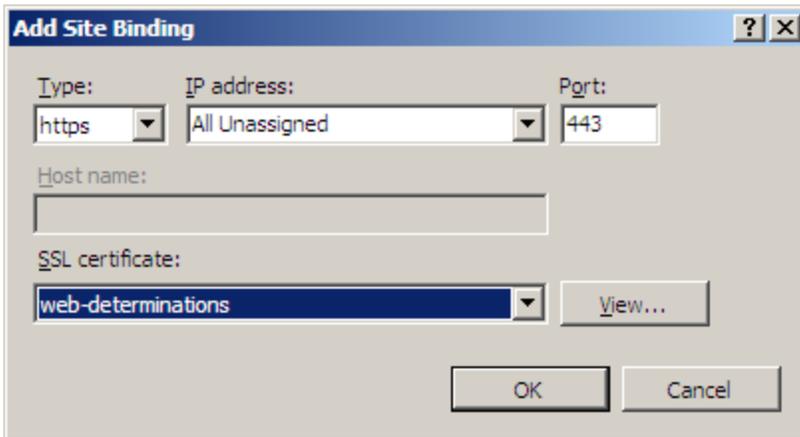
Now you need to configure IIS to use your certificate in SSL connections to web determinations.

Select the IIS Web Site which is serving Web Determinations, and click on the *Edit Bindings...* task in the *Actions* pane:

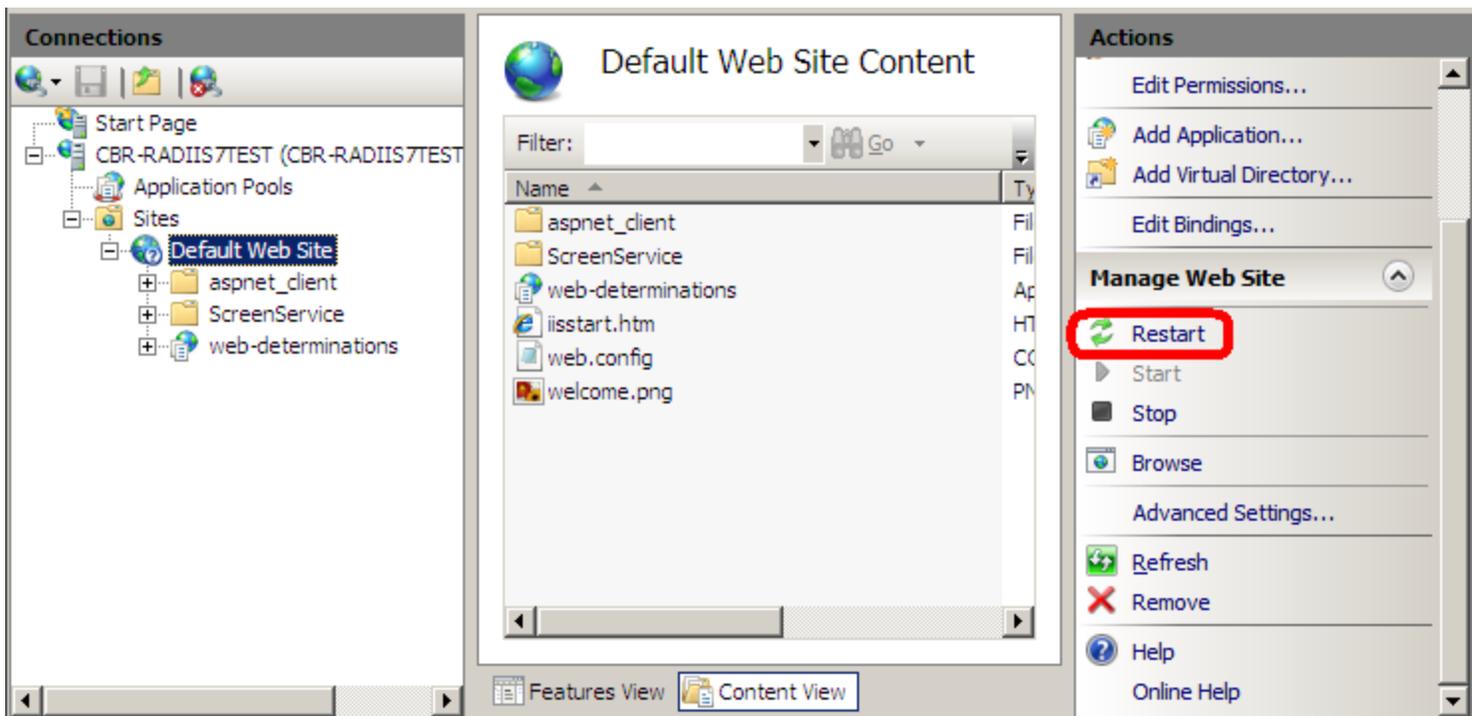


In the *Site Bindings* dialog, click the **Add...** button; the *Add Site Binding* dialog is presented.

Set the *Type* to "https" and the *SSL certificate* to the newly created self-signed certificate, or other existing certificate you wish to use:



Click **OK** to the *Add Site Binding* and *Site Bindings* dialogs, and restart your website using the *Restart* task from IIS Manager's *Actions* pane:



You should now be able to access Web Determinations via SSL.

In your browser, take whatever URL you normally use to access Web Determinations, and change "http:" to "https:"; for example, <https://localhost/web-determinations>. If you changed the port in the *Add Site Binding* dialog away from the default 443, you will need to include the non-default port in your URL, for example, <https://localhost:8443/web-determinations>.

Because you are using a self-signed certificate, users of all browsers will initially be warned that your website cannot be trusted.

Once you have the basic SSL access going, you should review the IIS documentation for more advanced options for production use, including importing a properly signed certificate from a known CA so that users will not be prompted to make a security exception, limiting access to authenticated users only, and disabling the default non-secure access.

Note:

If you are using Windows Server 2008 or later running IIS 7, attempts to view Oracle Web Determinations will bring up an error screen complaining about httpModules in a web.config file. The error screen suggests a number of resolutions, and the first and most preferable (according to the list) is to run a command line, approximately, "appcmd migrate config "Default Website/". Executing this command is safe and will resolve the problem.

SSL on Tomcat 5.5 and Tomcat 6

The official Apache documentation for enabling SSL on a standalone Tomcat 6 webserver can be found at: <http://tomcat.apache.org/tomcat-6.0-doc/ssl-howto.html>

This is a summary of those instructions, for the case that your Tomcat installation is running only Web Determinations, and you have not previously configured SSL for any other web applications on this Tomcat instance. If your installation is more complicated than that, you will need to refer to the Tomcat documentation linked above.

Before starting these instructions, first verify that you have a working Web Determinations installation without SSL; point your web browser at the URL at which you have configured Web Determinations. This may be:

`http://localhost/web-determinations`

or it may be:

`http://localhost:8080/web-determinations`

If you are not yet able to connect normally to web determinations, please refer to the topic, *Deploying Oracle Web Determinations for Java on Tomcat* in the *Oracle Policy Automation Installation Guide* before proceeding to enable SSL.

Assuming a default installation of Tomcat, a brief summary of the Apache instructions referenced above is as follows:

- **Create a certificate:**

This is most easily done using the Java "keytool" utility. Verify that your JDK bin directory is in your path, or CD to the JDK bin directory; for example, on windows the keytool executable might be found in `c:\sun\sdk\jdk\bin`, then enter the command:

```
keytool -genkey -alias tomcat -keyalg RSA
```

The tool will ask for a keystore password; by default, Tomcat expects the password "changeit" so you must use that password on your newly created keystore, or configure Tomcat to use a different one of your preference. The tool will then ask for the details of your organization for inclusion in the self-signed certificate, and finally a certificate password which tomcat requires to be the same as the keystore password you used before. This creates a `.keystore` file in your home directory (`<drive letter>:\Documents and Settings\<user>\.keystore` on windows) which is where Tomcat will look for it by default. See the Tomcat documentation if you wish to specify an alternate location for the keystore file in the case where Tomcat is running as a different user.

Note:

If you have already got a `.keystore` file in your home directory (with a certificate that you don't want to delete) see the Tomcat documentation referenced above for how to configure Tomcat to use it.

- **Enable SSL using the certification**

For a default Tomcat install, this can be done by uncommenting the *SSL HTTP/1.1 Connector* entry at `$CATALINA_BASE/conf/server.xml`; for example, on Windows the config file might be found at `C:\Program Files\Apache Software Foundation\Tomcat 6.0\conf\server.xml`

Find the section...

```
<!--<Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
    maxThreads="150" scheme="https" secure="true"
    clientAuth="false" sslProtocol="TLS" />-->
```

...and remove the `<!-- -->` from the ends.

Save, and restart Tomcat.

Note that your "port" value may be different from the one listed here.

You should now be able to access Web Determinations via SSL. In your browser, take whatever URL you normally use to access Web Determinations, change "http:" to "https:". If the port value is specified in your normal URL (for example, ":8080") you will need to change it to match the port value specified in the newly uncommented section of your config file (for example, <http://localhost:8080/web-determinations/> becomes <https://localhost:8443/web-determinations/>)

Once you have the basic SSL access operational, you should review the Tomcat documentation for the more advanced options for production use, including importing a properly signed certificate from a known CA so that users will not be prompted to make a security exception, limiting access to authenticated users only, and disabling the default non-secure access.

Save and resume interviews in Web Determinations

What do you want to do?

[Use the default file based persistence behavior](#)

[Add custom persistence behavior via a data adapter](#)

[Automatically save and resume interviews](#)

[Saving data from a Web Determinations interview to an external data source](#)

Use the default file based persistence behavior

The default is XDS Data Adaptor

- During a Web Determinations interview, user can click on **Save** to persist current session data, saved in `[web determinations webapp]\data\[rulebase]\[caseID].xds`.
- User can also load, the list picks up all xds files in `[web-determinations webapp]\data\[rulebase]` using the Case ID.

Add custom persistence behavior via a data adapter

Basically, a Data Adaptor allows the developer to create a custom way for the interview session data to be saved to a data source that they want.

The Data Adaptor controls the way the **Save/Load** buttons in the interview and how/where they save.

Usually it also allows the user to specify the name to save the session as.

Automatically save interviews

Saving interviews at pre-defined points during an investigation can be accomplished by creating an event handler which fires at the point in which the data is to be saved and invokes the save data method on the interview session; the following Events may be useful in this regard:

Event	To...
OnCommitEvent	Automatically save any data that has been committed to the session.
OnRenderScreenEvent	Automatically save data when a particular screen, or any screen is displayed.
OnInvestigationEndedEvent	Automatically save data once an investigation has been completed.

See also:

[Events and Event Handlers](#)

Automatically load interviews

Previously persisted interview data can be automatically loaded into a Web Determinations session in either one of two ways:

- By passing the case ID of the data to be loaded via the URL when starting a session. The format of the URL is:

`http://[server>]:[port]/[web-app]/startsession/[rulebase]/[locale]? caseID=[caseid]`

- By creating an **OnSessionCreatedEvent** event handler and invoking the load data method on the interview session.

See also:

[Events and Event Handlers](#)

Save data from a Web Determinations interview to an external data source

For information on saving data from a Web Determinations interview to an external data source, go to: [Data Adaptor plugin overview](#).

Integrate Web Determinations with another application

What do you want to do?

Integrate an interview experience within an existing application or process

Initialize an interview with data from an outside source

Pass interview data, conclusions and decision reports to an external application

Use an external data store for Web Determinations resources

Integration with another application will most likely involve use of the Web Determinations' Data Adaptor plugin which allows the current interactive session to use an arbitrary data source for loading and saving session data.

- Web Determinations session data can be saved to any data sources to which a Java/.NET app can connect. This ranges from simple file systems to databases, and also large applications that expose their data source via API's.
- A new Web Determinations session can be created and pre-seeded with rule data from data sources.
- A user's current session can be persisted to data sources. The user is able to specify their own Save ID, or can save to the current Save ID .

Integrate an interview experience within an existing application or process

Go to:

Embed Oracle Web Determinations inside a frame

Configure Oracle Web Determinations' look and feel to appear embedded

Integrate Oracle Web Determinations into a parent application's workflow

Embed Oracle Web Determinations inside a frame

To get Oracle Web Determinations to embed inside a frame, you need to turn off the commentary frameset and tell Web Determinations to target the current frame rather than the top frame. This can be done by going to [appearance.properties](#) and making the following settings:

```
use-session-frameset=false frameset-top-target=_self commentary-target=commentary_frame
```

The application can then be started by either going to the main page using:

Java:

```
http://<server>:<port>/<webapp>
```

.NET:

```
http://<server>:<port>/<webapp>.aspx
```

Or you can start a session for a rulebase directly by going to:

Java:

`http://<server>:<port>/<webapp>/startsession/<rulebase>/<locale>`

.NET:

`http://<server>:<port>/<webapp>/startsession/<rulebase>/<locale>.aspx`

Configure Oracle Web Determinations' look and feel to appear embedded

This following discusses how to configure the Oracle Web Determinations' theme/skin/UI so that it appears to be part of the parent application.

Page styling - colors, fonts, and images

The first thing to change in order to embed Oracle Web Determinations into an application are: the colors used, the text font/size, image and branding. Changing these will achieve the most in terms of making Oracle Web Determinations feel part of the parent application.

A lot of configuration is available in the *appearance.properties*, such as the body text font and color as well as styling of various Web Determinations page components (progress bar, menu bar, header, and so on).

While there is a lot of configuration available, only some parts of Web Determinations' look and feel can be controlled via *appearance.properties*. The *appearance.properties* file allows easy modification of commonly-customized Web Determinations areas.

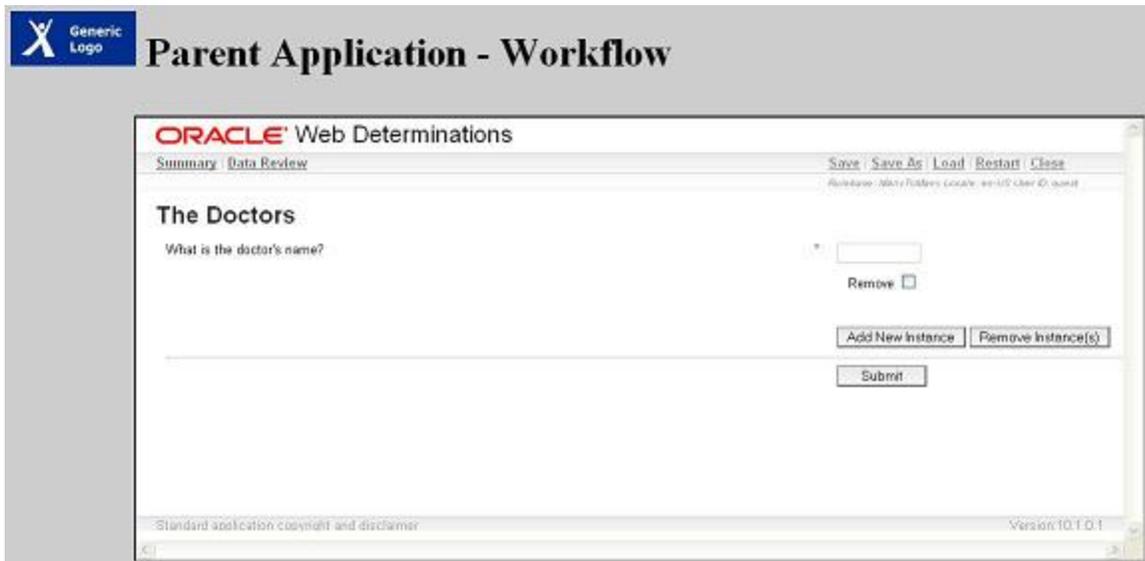
Specific styling changes not possible in *appearance.properties* can be made by modifying the **main.vm.css** - located in `<OWD webapp>/WEB-INF/classes/templates/main.vm.css`. Note that this requires knowledge of HTML and CSS.

Here are a list of things to consider changing in Web Determinations that are configurable in *appearance.properties*:

- body text color and font.
- body background color.
- link text color.
- header image/title/font/color.
- progress and menu bar.
- footer text/background.
- messages text/background.

The following two screen shots demonstrate the difference that can be made with a few tweaks in the *appearance.properties*.

Example - appearance.properties untouched:



Example - appearance.properties optimized

- header image disabled.
- body font changed to match parent application.
- body background color changed to match parent application.



Disabling Oracle Web Determinations page components

Oracle Web Determinations has some page components that can be removed so it does not look like a stand-alone application. Some page components can be disabled via `appearance.properties`, while others need to be disabled manually by manipulating the Web Determinations templates. Note that manipulating templates requires knowledge of Velocity.

Header and Footer

The Web Determinations Header and Footer would be the primary components to be removed for embedding Web Determinations. As the Header and Footer cannot be disabled via `appearance.properties`, this needs to be done manually by modifying the Web Determinations templates. A main template is a template that contains the `<html>` tag; for example, `start_screen`, `summary_screen`, `question_screen`.

Bars

The next set of components that can be removed from Web Determinations are the bars - the menu bar, progress bar, and status bar. They can be completely disabled via `appearance.properties`.

Example - Bars removed

Below is an example of what it would look like if the menu, status, and progress bars are disabled.



Modifying page layout and other specific modifications

If the parent application has a layout or characteristics that are unique and do not follow standard web layout, the Oracle Web Determinations page layout may need to be tweaked to match.

This is not a simple change, and is considered advanced customization. Customizing the Web Determinations Velocity templates allows for nearly limitless changes, but can also reduce stability and maintainability.

To prevent reduction in stability and maintainability,

- keep the modifications to a minimum.
- avoid complex modifications.
- avoid modification of Web Determinations' main templates (templates that have the <html> tags); that is, if the changes are to a specific form or control.
- styling changes should be done via CSS, not via templates.

Some examples of page layout and specific modifications are:

- changing the summary screen layout.
- changing the question screen layout.
- customizing controls appearance/placement such as labels, text fields

Integrate Oracle Web Determinations into a parent application's workflow

This following describes an approach to integrating Oracle Web Determinations into a parent (web) application's workflow/process. This means that part-way through the workflow, the parent application will hand over data to Web Determinations, which will then conduct an interview.

Data (both provided and determined) during the Web Determinations interview is then passed back to the parent application. The parent application can then use the data passed back for further analysis or to drive the workflow.

How to integrate Oracle Web Determinations within a workflow

Integrating an Oracle Web Determinations interview inside an application's workflow means that the application can leverage Web Determinations' capability for handling complex determination scenarios.

To integrate the Web Determinations interview, the parent application needs to:

- pre-seed the Web Determinations interview with data collected from previous workflow forms or from the parent application's datasource.
- start the Web Determinations interview.
- retrieve the inferred or provided data from the Web Determinations interview.
- resume original workflow.

Pre-seeding the interview

Pre-seeding a Web Determinations interview makes the integration with the workflow seamless - the user does not have to re-enter data that was provided already to the parent application (either via a previous form on the same workflow or another workflow/app altogether).

The parent application can feed data to the Web Determinations interview by using the **Load** action. The data can be a mixture of:

- data that is gathered from previous forms in the workflow.
- data that the parent application retrieves from a datasource.

The Web Determinations **Load** action uses the Data Adaptor to fetch and load data to a fresh interview, 'pre-seeding' the interview before it begins. More information about pre-seeding and Data Adaptor can be found in [Data Adaptor - common scenarios](#)

Starting the interview

To start a Web Determinations interview, a specific URL needs to be constructed. The URL construction works for both starting a fresh interview and pre-seeded interview, although the latter is probably the expected usage when integrating.

Web Determinations has a REST-based URL, allowing for a specific rulebase, locale, goal, and session ID to be provided. This avoids the user having to manually make the selection for those items, and helps with a seamless integration.

The REST URL to construct for starting the pre-seeded Web Determinations interview is as follows:

http://<OWD_webapp>/startsession/<rulebase_name>/<locale>/<goalID>?caseID=<caseID#>&user=guest

Description:

- **OWD_webapp** - the domain name and path the Web Determinations web app is accessible from http; for example, for locally installed tomcat - <http://localhost/web-determinations>.
- **rulebase_name** - the name of the rulebase, spaces are concatenated.
- **locale** - the locale the rulebase interview should be run, each rulebase has at least one locale.
- **goalID** - the goal ID to start the interview investigation/determination on. This is optional, but providing this means that the user does not have to manually select a goal in the Summary screen (even if there is only 1 goal in the Summary screen).
- **caseID** - the case ID is provided to the Data Adaptor by the 'Load' action. The Data Adaptor will use the ID to search and locate for a saved 'interview session data' for the current rulebase, and load that session data to pre-seed the interview.

High level process for pre-seeding and starting an Oracle Web Determinations interview

1. The parent application workflow reaches the stage where it requires a determination.
2. The parent application's server-side has saved workflow data in a datasource that the Web Determinations Interview will require for pre-seeding.
3. The parent application calls the Web Determinations interview to start, using a special (REST) URL:
 1. the REST URL instructs Web Determinations to 'load' a specific ID (for example, the applicant's ID).
 2. the REST URL can also instruct Web Determinations to start on a specific Goal, therefore skipping the Summary Screen (which is crucial if Web Determinations is to be embedded).
4. Web Determinations' **Load** action uses the Data Adaptor to load the ID.
5. The Web Determinations Data Adaptor, which has been created specifically to access the parent application's datasource, accesses the datasource and retrieves the necessary data for pre-seeding the interview.
6. The Data Adaptor loads the data into the Interview, and the Interview Engine does a 'think' with the present data to check if any goals have been determined (for integration with a workflow, it should not reach a goal).
7. Web Determinations starts the interview and presents a Question screen to the user.

Retrieving the data from the Oracle Web Determinations interview

The parent application has a few options to retrieve the data from a Web Determinations interview session.

1. Let the Data Adaptor save the data into a datasource; for example, database in the parent application:
 1. Best approach.
 2. Because it is attached to the default Data Adaptor 'Save' action, can be called via different means; for example, Extension event, REST URL.
2. Use an Extension event to access the interview session data, and save it into a datasource:
 1. If the Data Adaptor 'save' action needs to be mapped to another save process, this is another option.
 2. Cannot be called manually, so the event has to be triggered for the save to happen.
3. Create a Custom Screen that redirects to a URL link, supplying the data through a HTTP call:
 1. Not the cleanest solution, low security, not suitable for anything but simple/primitive data.
 2. Very flexible.
 3. Can also serve dual purpose - to retrieve data from the Web Determinations interview and also resume flow in the parent application workflow.

Resuming the original workflow

To resume the original workflow, the Web Determinations interview needs to direct the user to the next form in the workflow after the interview is completed.

There are many different approaches for this since web applications vary in implementation. Some are complex and dynamic, employing heavy Javascript/AJAX while others are simple and rely on basic URL request/response to drive the flow.

For basic (non-dynamic) webapp setups, the Web Determinations interview can change the URL of the frame (or the parent page) to the next form in the workflow.

To achieve this:

- Web Determinations must know how to construct the URL to redirect to, or must be given the URL (by the parent application) when the interview is started.
- The workflow must be built in such a way that a session ID and stage can be provided, so that it knows what session to continue, and at what page.
- Web Determinations needs to have a Custom Screen setup that will redirect to the URL.
- To change the URL of the parent page - Javascript can be used to access and trigger the parent page to refresh to the new URL. Alternatively a link to the URL with target="_parent" can also be used.

Some web applications employ DHTML/Javascript, and can control the page or frame's URL. In this case, Web Determinations may only need to be able to signal via Javascript that it has completed, and the parent application's DHTML/Javascript architecture can resume control of the workflow.

Initialize an interview with data from an outside source

You can initialize an interview by *pre-seeding* it with data from an outside source in instances where:

- A client data source already knows information about the current user.
- The client system wants to redirect the user through Web Determinations and start an interview pre-seeded with known information about the user (from the datasource).

To enable pre-seeding, do the following:

1. Create the Data Adaptor for the Web Determinations web application that is to run the pre-seeded interview.
2. Proceed to the Web Determinations interview via a constructed link that defines the Case ID for the user; this case ID is then used by the Data Adapter to retrieve the necessary data from the external data source.
3. Commence the Web Determinations interview with the pre-seeded data.

The URL that will direct the user to the pre-seeded interview allows the client system to specify a Case ID that the Data Adaptor will use to retrieve data from the external datasource. The URL format is:

`http://<webserv address>/<WD web app name>/startsession/<rulebase name>/<locale ID>?case-ID=<caseID#>&user=guest`

The client system needs the following information in order to construct the URL:

- **webserv address** - the address of the web application server on which Web Determinations is installed.
- **WD web app name** - the name of the Web Determinations webapp (default is web-determinations).
- **rulebase name** - the name of the target rulebase installed in Web Determinations.
- **locale** - the locale for the interview session; for example, en-US, es-CR for Spanish - Costa Rica.
- **case ID** - the Case ID to be provided by the client system. The client system may already have the data needed to retrieve/-construct the Case ID, or the client system may authenticate the user retrieve or generate the Case ID.

To build the Data Adaptor that will load the data from the datasource to the target rulebase you first need to determine what data is to be extracted from the datasource, and how it will map to the rulebase as instance data.

Pass interview data, conclusions and decision reports to an external application

In the event of the data being inferred from a Web Determinations interview session needing to be saved to an external data source, you must first take the following action in order to determine what action will be taken to save the data:

- Automatic saving of data requires Event handlers that call the Data Adaptor to save the current data.
- You are also able to save by clicking on the default Save/Save As command available in a Web Determinations interview session; this also calls the Data Adaptor to save the current data.

A Data Adaptor needs to be developed for the client system so that when the 'save' action is issued, the custom-developed Data Adaptor will persist the specific Web Determinations interview session data into the external datasource.

This scenario also has variations/extensions, which are:

- Auto-saving of Web Determinations interview data whenever the user completes a screen.
- Loading base instance data and saving inferred instance data

Note that when calling the Data Adaptor save, only the Case ID and interview session data is passed in.

Auto-saving data during a Web Determinations interview

Auto-saving data during a Web Determinations interview can be driven by Event Handler plugins.

Auto-saving will use the 'save' capability of the loaded Data Adaptor. To persist Web Determinations interview data into an external datasource see "Saving data from a WD Interview to an external data source".

Below are common triggers for auto-save, and the Event to use:

- when the user completes a screen (**OnCommitEvent**).
- when a goal is reached (**OnCommitEvent** + checking the Session data that the target goal attribute value has been provided).
- when certain non-base attributes are inferenced (**OnCommitEvent** + checking the Session data that the target attribute value has been provided).

More complex scenarios can combine the above so that, for example, auto-save per screen is performed and also when the goal is reached.

Because Case ID and interview session data is the only data passed through to the Data Adaptor, design of the Event Handler and Data Adaptor must take this into account. This means that the Data Adaptor must be able to operate based on information available in the Interview Session data passed; that is, the Event Handler cannot tell the Data Adaptor if the save is for a completed screen, or a completed goal.

With this design restriction, the Data Adaptor needs to be able to persist the data based on the actual instance value together with the rulebase model data.

Use an external data store for Web Determinations resources

There are situations where rulebases of an Oracle Web Determinations implementation need to be stored and accessed from a custom datasource. The **RulebaseResolverPlugin** is an Interview Engine plugin that allows usage of custom datasources to store and retrieve rulebases from. Also it lets the implementer customize the rulebase service functionality. For information on how to create a Rulebase Resolver plugin, go to the topic [Create a Rulebase Resolver plugin](#).

Install and register plugins

The following is intended as a guide for implementing plugins that are loaded and invoked by the Oracle Web Determinations and Determinations Server extension framework.

Plugin interface

All extension implementations must implement an interface that is descended from the super plugin interface **Plugin**. Typically, you are required to implement the most specific interface from this hierarchy, according to the type of plugin that you are writing. For example, if you wanted to implement a plugin that was an event handler for the **OnSessionCreatedEvent** engine event, you would implement the **OnSessionCreatedEventHandler**, **NOT** the generic **EngineEventHandler** interface.

Regardless of the specific interface implemented, all plugin implementations must conform to the following to be loaded and invoked properly by the extension framework:

- must implement the **getInstance(args : RegisterArgs) : Plugin** method. This method is to return an instance of the plugin object if the plugin has determined (using the contents of the specific **RegisterArgs** implementation instance passed) that it is usable.
- must have a constructor containing no parameters.
- must correctly implement all methods defined by the specific interface that it implements. This includes methods defined in super-interfaces of the specific interface implemented by the plugin.

Extension types

The point at which an extension is registered and the registration arguments that are passed to it, is determined by its type. There are several types of plugins:

- **Engine plugin:** Interview Engine plugins are available to customize the behavior of the Interview Engine. Interview Engine plugins are registered when the instance of an engine is create.
- **Platform plugin:** plugins that are available in the Web Determinations web application, as opposed to the Interview Engine or session. These are registered when an application session is created, as distinct from an Interview Session.
- **Service plugin:** Determinations Server plugin that allows the leveraging of the Web Determinations technology to create a custom SOAP/HTTP based web service.
- **Session plugin:** plugins that attach to a particular Interview Session. Interview Session plugins are registered when the instance of a session is created.

What do you want to do?

[Install extensions to work on Oracle Policy Modeling Build and Run instances](#)

[Register an extension](#)

Understand what installation methods are available

There are two methods of installing extensions:

1. Auto-discovery
2. Manual configuration

Auto-discovery

It is possible to automatically discover and register extensions when the applications start. Using this method, plugin installation is simply a matter of compiling the plugin libraries (.jar in Java or .dll in .NET) and deploying them to the plugins folder and restarting the application. This folder is located in `<web root>/WEB-INF/classes/plugins` in Java or `<web root>/bin/pluginson` .NET. Multiple plugins can be compiled into a single library or multiple plugin libraries can be used. Any third party dependencies must be copied into a location that will cause them to be automatically loaded when the applications starts such as `<web root>/WEB-INF/lib` in Java or `<web root>/bin/` on .NET.

Note for Java deployments:

Because the Auto-Discovery mechanism requires the ability to introspect the class path, this method may not be available in certain application servers such as Oracle WebLogic Application server, when running the war file unexpanded. In such cases, plugins must be loaded using the manual configuration option

Manual configuration

Manually configuring which plugins to load involves deploying the extensions, and any third party dependencies to a location that will automatically be loaded when the application starts such as `<web root>/WEB-INF/lib` in Java or `<web root>/bin/` on .NET. The list of plugin classes to be loaded must then be manually specified in the `plugin.libraries` section in the `application.properties` file.

Install extensions to work on Oracle Policy Modeling Build and Run instances

Installing extensions to enable Oracle Policy Modeling Build and Run instances to use those extensions is the same process as installing either onto the Determinations Server or onto a Web Determinations server.

The Oracle Policy Modeling Build and Run command invokes a Java web server as well, and its plugin folder can be found in the 'Release' folder directly in the project folder; for example, say we have a rulebase named 'MyRulebase', located in `C:\-projects\MyRulebase`, the plugins folder will be located in the path:

`C:\projects\MyRulebase\Release\web-determinations\WEB-INF\classes\plugins`

Register an extension

Regardless of the specific interface implemented, all extensions must conform to the following to be loaded and invoked properly by the extension framework:

- Must implement the **getInstance(args : RegisterArgs) : Plugin** method. This method allows an Extension class to read the **RegisterArgs** and its contents to determine whether it should be registered to the current Web Determinations Interview (by returning an instance of itself). In addition to that, because the plugin has control on what instance of itself it returns, it is able to instantiate an instance of itself with arguments from the **RegisterArgs** args.
 - This allows the plugin to only become loaded on specific rulebases, or locale, and so on if rulebase or locale data is in the passed **RegisterArgs**.
 - This also allows the plugin to choose which constructor to use for instantiation; for example, new **CustomPlugin (SessionContext ctx)**.
 - This is critical for ensuring that only one plugin is registered per Web Determinations interview.
 - Note that the above **RegisterArgs** is more specific depending on the plugin; for example, **DataAdaptorPlugins** receive **InterviewSessionRegisterArgs**, **CustomScreenProviders** receive **PlatformSessionRegisterArgs**, and so on.
 - It is permissible to implement singleton plugins by returning the same instance of the plugin for multiple invocations of **getInstance()**, however, it is the implementer's responsibility to ensure thread safety in such situations.
- Must have a public parameter free constructor.
- Must correctly implement all methods defined by the specific interface that it implements. This includes methods defined in super-interfaces of the specific interface implemented by the plugin.

Code sample - sample plugin - RulebaseSpecificDataAdaptor

```
public class RulebaseSpecificDataAdaptor implements DataAdaptorPlugin { //TIP: Each Plugin implements a
Plugin Interface

    /**
     * Only provide this plugin if the Web Determinations session's rulebase is the 'SpecialRulebase' rulebase.
     * The specific RegisterArgs object we get here is an instance of InterviewSessionRegisterArgs.
     */
    public Plugin getInstance(InterviewSessionRegisterArgs args) { //TIP: Each Plugin interface implemented
have different getInstance() argument
        if (args.getSession().getRulebase().getIdentifier().equals("SpecialRulebase")){
            return new RulebaseSpecificDataAdaptor ();
        } else {
            return null;
        }
    }

    /**
     * Parameterless constructor to conform to the plugin requirements
     */
}
```

```
public RulebaseSpecificDataAdaptor (){  
  
}  
  
...
```

Notes about Hot-swapping mode

If Hot-swapping mode is turned on, then:

- When a new rulebase is added, the Custom Service plugin will be given an opportunity to register a service against that rulebase.
- When a rulebase is deleted, any service registered against it will be automatically unloaded.
- Updating an existing rulebase is equivalent to a delete and load.

Do's and don'ts

- As part of the initialization logic implemented in your implementation of the **getInstance** method or the parameter-less constructor, **DO NOT** set a static (class-scope) reference to any of the objects encapsulated by the given **RegisterArgs** parameter.
- It is not permissible for plugins to directly alter the state of any of **RegisterArgs** members passed in the get instance method.
- Do not make assumptions about the number of times that the **getInstance** method is called. It is up to the logic implemented in this method to determine whether the plugin is to be available in a particular context. Every call to this method is another chance to the plugin to inform the application that it is ready to be used or to invalidate itself for use.
- Once the first call to the **getInstance** method returns, the plugin's code may be called at any point. That is, *you may not make any assumptions about the order and number of calls into your code.*

The Interview Portlet

A portlet is a pluggable website component that is managed and displayed in a web portal; the Interview Portlet allows interviews to be conducted in such a web portal.

When customizing the Interview Portlet, there are many similarities to customizing Web Determinations; this page will only have portlet-specific customizations, but if you wish to obtain more information, see the topic [Customize the existing Web Determinations user experience](#) in the [Custom development](#) section.

Topics in "The Interview Portlet":

- [Look and feel customization](#)
- [Interview Portlet events](#)
- [Plugins](#)
- [Pass in parameters](#)
- [Display commentary](#)
- [Set the portlet size in Webcenter](#)

Look and feel customization

In order to help you with your customization, an example of a fully customized Interview Portlet using the *Social Services Screening* rulebase can be found in the Java runtime zip file at `examples\interview-portlet\social-services-screening`. Additionally, the rulebase source can be found at `examples\rulebases\source\SocialServicesScreeningPortlet.zip`.

Add static resources

Like in Web Determinations, developers can customize the Interview Portlet by adding custom images and other static resources that can be either local (within the `opa-interview-portlet` web application), or external (accessed via URL).

Load local resources

To add an image to a portlet page, do the following:

1. Place the image file in the "images" directory of the deployed portlet (`opa-interview-portlet\WEB-INF\classes\images`).
2. Refer to the image in the appropriate Velocity template by using the following code, where `imgFileName` is set to the name of the image file:

```

```

For instance, in the Social Services Screening example, some goal controls on the summary screen have a custom attribute **PortletImgPath**. This property is set to a value like `sss/AmIEligible_v03.gif`, where `AmIEligible_v03.gif` is an image file found in the directory `WEB-INF\classes\images\sss` of the portlet war file. The customized `GoalControl.vm` template then renders the image as follows:

```
#set ($imgPath = ${control.getProperty("PortletImgPath", "")})
#if ( $imgPath == "" )
    ${control.getText()}
#else
    
#end
```

To reference a static HTML page in a portlet page, do the following:

1. Place the HTML file in the `resources` directory of the deployed portlet (`opa-interview-portlet\WEB-INF\classes\resources`).
2. Refer to the resource in the appropriate Velocity template by using the following code, where `htmlFileName` is set to the name of the HTML file:

```
<iframe src="{urlRewriter.getResourceURL("{resource-request}/htmlFileName")}" />
    <p>Alternative text</p>
</iframe>
```

For instance, in the Social Services Screening example, some label controls on the summary screen have a custom attribute **PortletResourcePath**. This property is set to a value like `help/more_info_food_stamps.html`, where `more_info_food_stamps.html` is an HTML file found in the directory `WEB-INF\classes\resources\help` of the portlet war file. The customized `LabelControl.vm` then renders the link as follows:

```
#set ($resPath = ${control.getProperty("PortletResourcePath", "")})
...
```

```
#if ($resPath != "")
  <a href="{urlRewriter.getResourceURL("{resource-request}/{resPath}")}">
#end
#parse("investigation/control-text.vm")
#if ($resPath != "")
  </a>
#end
```

Note:

Because the portlet only generates an HTML fragment, it does not have access to the HEAD element and therefore it is not possible to include CSS style-sheets in the generated HTML. Any styling must be done at the portal page level.

Configure the location of images and static resources

By default, images are loaded from the directory `opa-interview-portlet\WEB-INF\classes\images` and resources are loaded from the directory `opa-interview-portlet\WEB-INF\classes\resources`.

These directories can be changed to another location by setting the **images.path** and **resources.path** properties in `opa-interview-portlet\WEB-INF\classes\configuration\application.properties`. (For more details, see [Resource Loading properties](#))

Load external resources

To add an external image to a portlet page, refer to the image in the appropriate Velocity template by using the following code, where `imgURL` is set to the URL of the external resource: ``

Customize appearance

As the portlet framework does not allow for a standards-compliant way of including CSS files, the templates have been modified to *hard include* style information into HTML elements.

These styles are controllable through new settings in the `appearance.properties` file and have been given names that reflect the original style (basically by appending '-style' to the original style name; for example, the style settings for the "warning" style is contained in the "warning-style" setting).

Someone customizing the appearance of the portlet has the choice of either modifying the new style settings in the `appearance.properties` file or modifying the template files directly - which basically corresponds to the web determinations process of either modifying the `.css` file or the templates.

Custom screen and custom control plugins

Custom plugins can be deployed just as for Web Determinations, by placing the .jar file into the `WEB-INF\classes\plugins` directory of the `opa-interview-portlet` web application. More detailed information about plugins can be found in the topic [Customize the existing Web Determinations user experience](#).

The following are portlet-specific plugin considerations:

Custom screens

Custom screens for the portlet are developed using the same methodology as [custom screens for Web Determinations](#), with the following restrictions:

- If the screen needs to generate a URL to another screen or resource, you need to use the URL Rewriter to obtain a URL encoded for the portal server. Specifically, you need to use `com.oracle.determinations.web.platform.controller.URLRewriter.getURL()`. You can obtain an instance of the URL Rewriter using `com.oracle.determinations.web.platform.controller.SessionContext.getCustomURL()`. More information about the URL Rewriter can be found in the Javadoc (see the `help\api` directory of OPA Help).
- If your custom screen needs to return an `ExternalRedirectResponse`, you cannot do so in the `processAndRespond` method, since this method is invoked during the portlet's render phase, when the portal framework does not allow external redirects. Instead, you can develop an event handler for the `OnAfterProcessActionEvent`, which updates the portlet response about to be rendered.

The portlet custom screen example ([Example: Create a Custom Screen for the Interview Portlet](#)) provides an example on how to implement redirects, including external redirects, in a custom screen for the portlet.

Custom controls

If your custom control needs to place a hyperlink on the page (for example, to link to another interview screen), the control Java code needs to use URL Rewriter to obtain a URL encoded for the portal server. Specifically, you need to use:

```
com.oracle.determinations.web.platform.controller.URLRewriter.getURL().
```

You can obtain an instance of the URL Rewriter using `com.oracle.determinations.web.platform.controller.SessionContext.getCustomURL()`. More information about the URL Rewriter can be found in the Javadoc (see the `help\api` directory of OPA Help).

As an example, the following Java code allows a custom control to build a URL for the summary screen (this is an additional method for the `BenefitCodeControl` class found in the topic [Custom Control - BenefitCode Walkthrough Example](#)):

```
/**
 * An example of how to get a portlet URL for the summary screen, using the URL rewriter
 */
public String getCustomURL() {
    SessionContext ctx = screen.getSessionContext();
    URI summaryURI = new URI(
        "screen",
        ctx.getInterviewSession().getRulebase().getIdentifier(),
        ctx.getInterviewSession().getLocale(),
        ctx.getSecurityToken().getIdentifier(),
        ctx.getCaseID(), new String[]{EngineConstants.DEFAULT_SUMMARY_SCREEN_NAME}
    );
}
```

```
);  
    return ctx.getURLRewriter().getURL(summaryURI, false);  
}
```

The following needs to be inserted in the respective Velocity template for this custom control:

```
<a href="{control.getCustomURL()}">Click here to go to the summary screen</a>
```

Pass in Parameters

Pass in parameters for an Interview Portlet across WSRP

We pass in parameters for an Interview Portlet across WSRP by this using 'public render parameters' as defined in JSR268 and WSRP2; by default, the portlet is configured to receive the following parameters:

- **start_investigation_proxy**
- **rulebase_proxy**
- **locale_proxy**
- **case_id_proxy**
- **user_id_proxy**
- **goal_proxy**
- **queryParam_preseedID**

If the value of **start_investigation_proxy** is **true**, then the values for the other parameters are inspected and used to start an investigation. Note that this will end any investigation currently in progress.

These public render parameters can be set by another portlet located on the same portal page as the Interview Portlet.

For an example of passing parameters, see the topic [Pass in parameters for an Interview Portlet across WSRP](#).

Notes:

- The **goal_proxy** parameter is only taken into account if the **locale_proxy** parameter is set in addition to the **start_investigation_proxy** parameter.
- **queryParam_preseedID** is the preferred way of starting a new Web Determinations session with pre-seeded data; the **case_id_proxy** parameter is still supported, but ideally, should only be used for loading existing sessions.

Setting portlet parameters on WebCenter

On WebCenter, portlet parameters can be bound to page parameters, which in turn can be bound to portal page URL parameters and populated at runtime. This way, it is possible to launch a portal page with specific parameters for the interview portlet.

- To bind a portal parameter to a page parameter, set its value to an expression like **`#{bindings.queryParam_preseedID}`**.
- To bind a page parameter to a URL parameter, set its value to an expression like **`#{param.queryParam_preseedID}`**.
- Finally, to set the URL parameters, access the portal page using a URL like:
`<path>?start_investigation_proxy=true&rulebase_proxy=AdminSmokeTest&queryParam_preseedID=0-1.`

Additionally, portlet parameters can also be set at design time by the administrator of a page by accessing the **Edit -> Parameters** option for the Interview Portlet.

Note: Page query parameters will persist across restarting and closing the interview. This may cause undesirable behaviour; for example, if a session is started with a particular **queryParam_preseedID** specified as a page parameter, but then restarted, the subsequent session will also have the same **queryParam_preseedID**.

Receive additional parameters

Any custom parameters can be received by the portlet, and processed; for example, in the custom event handlers. To receive additional parameters, the *portlet.xml* file needs to be augmented with these additional parameters, and the parameter setting portlet needs to set them.

Pass in parameters for a local interview portlet

In some cases, users might want to launch straight into a particular rulebase, locale, goal, as well as supply user-id and locale-id parameters.

In Web Determinations, these parameters are passed as URL query string parameters; however, this does not work for the portlet out of the box, because the portlet does not have access to the URL parameters of the portal page, nor the HTTP request. Instead, we have created a servlet which acts as a proxy, receiving the parameters, storing them in the session and performing a redirect to the portal page after which the portlet extracts the parameters from the session.

Note: If a portal page has more than one instance of the interview portlet, invoking the proxy will cause each portlet's investigation to be started with the supplied parameters.

Configure the proxy

The web.xml file in the opa-interview-portlet directory configures the proxy servlet. Specifically, the following elements can be configured:

- the **url-pattern** element in **servlet-mapping** defines the URL where the proxy servlet can be accessed from. This may be left unchanged from the default value **/StartInvestigation**.
- the **init-param/param-value** element in **servlet** defines the URL of the portal page containing the portlet, which is to receive the parameters. This **must** be set to the URL of the actual portal page.

Note: The portal page must reside on the same application server as the proxy servlet.

Sample web.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:javaee="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" id="interactive-platform">
  <display-name>OPA Servlet-Portlet</display-name>
  <description>OPA Servlet-Portlet</description>

  <!-- The servlet is a proxy for starting an investigation for cases where where want to pass additional parameters e.g. rulebase. Note that we cannot pass URL params directly to the portlet because the portlet doesn't have access to the URL. -->
  <servlet> <servlet-name>StartInvestigationProxyServlet</servlet-name>
  <servlet-class>com.oracle.determinations.portlet.StartInvestigationProxyServlet</servlet-class>
  <init-param>
    <!-- This parameter specifies the URL of the portal page containing the portlet, to which we should redirect the request once we have put parameters in the portlet session -->
    <param-name>redirect</param-name>
```

```

        <param-value>MyPortal/MyPortalPage</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet> <servlet-mapping>

    <servlet-name>StartInvestigationProxyServlet</servlet-name>
    <url-pattern>/StartInvestigation</url-pattern>
</servlet-mapping>

</web-app>

```

Session management

The application server needs to be configured so that the proxy servlet shares the session state with the portlet. On Tomcat-based application servers, this can be done by adding the `emptySessionPath="true"` attribute to the `Connector` element in `conf\server.xml`.

Supported parameters

The proxy supports the following parameters:

- **case_id_proxy**
- **user_id_proxy**
- **rulebase_proxy**
- **locale_proxy**
- **goal_proxy**
- **queryParam_preseedID**

The query string format is `?param1=val1¶m2=val2...`

Examples

Rulebase and Case ID:

```
http://localhost:8080/opa-interview-portlet/StartInvestigation?rulebase_proxy=AdminSmokeTest&case_id_proxy=1-1RN26,guest,0-1,AdminSmokeTest
```

Rulebase and locale:

```
http://localhost:8080/opa-interview-portlet/StartInvestigation?rulebase_proxy=SuperSimple&locale_proxy=fr-FR
```

Rulebase, User ID, Goal ID and locale:

```
http://localhost:8080/opa-interview-portlet/StartInvestigation?rulebase_proxy=y=SocialServicesScreeningPortlet&user_id_proxy=Bob&goal_proxy=Attribute~interview_complete~global~global&locale_proxy=en-US
```

Display Commentary in the Interview Portlet

The Interview Portlet does not support commentary display in a popup window. Commentary only exists for question screens and can only be displayed in the screen's associated iframe.

There are three aspects to handling commentary:

- Screen commentary - commentary associated with the question screen.
- Control commentary - commentary associated with the control text.
- Commentary plugins.

Commentary URLs need to be modified so that they take into account that the interview portlet is displayed within a portal. The commentary is displayed as a static page and so it is necessary to encode the URLs as a reference to a static resource.

To display commentary as a static page use the **CommentaryContentAction** action in web-determinations.

When using the **getStaticCommentaryURL** method in **LocalAbstractNativeScreen** and **LocalAbstractControl**, the commentary target is returned as a **ResourceURL** with the action set to **CommentaryContentAction**.

The **ResourceURL** is created for the commentary target using the **PortletRewriter** and when clicked, it triggers the **serveResource** method in interview portlet.

It is necessary to modify the associated templates so that they call this method for loading HTML pages as a resource. This is managed by replacing the **screen.getCommentaryURL()** method with the **getStaticCommentaryURL()** method.

- Screen commentary - the screen title is displayed using the screen-title.vm in templates/investigation.
- Control commentary - the control text is displayed as link using the control-text.vm in templates/investigation.

Special case for screen commentary:

For screen commentary it is also necessary to handle the displaying of associated commentary on the same page. In web-determinations, this is done using frames. If a screen has commentary then the *frameset.vm* template is used in place of the *question_screen.vm* template, to display the investigation screen and commentary side by side. As framesets do not work in the interview portlet framework, it is necessary to use iframes to display the commentary on the same screen. To manage this, the *question_screen.vm* template has been modified to display the associated commentary in an iframe which is now the commentary-target. This is done by setting the value of commentary-target in *appearance.properties* to the name of the commentary iframe in *question_screen.vm*.

The **isPortletURI()** query is used to indicate that the URI was created in context of a portlet (if a value of "true" is used); otherwise the default value is false. The interview portlet uses **setIsPortletURI(value)** call to set the value to true. When a question screen is being rendered this value dictates whether framesets should be used or just the question screen which includes the iframe.

Special case for control commentary:

For control commentary, the control raw text itself may have embedded commentary links. For example, in the statement "Is the sun shining?", there may be commentary associated with the words "sun" and "shining"; note that embedded commentary links are not supported at this stage .

Commentary plugins work by re-mapping the commentary target to a new URL; when **getStaticCommentaryURL()** is invoked, it returns one of the following:

- a. If a plugin for redirecting commentary is provided, then return the target URL provided in the plugin
- b. In absence of a plugin, return the original commentary URL

Special case for commentary in Webcenter:

Portals do not generally have any problems displaying commentary in an iframe using the standard redirecting of url in an A tag by using the target attribute. In Webcenter however, the URL for an A tag is converted to a javascript call which only works for same-screen display. Redirecting to an iframe or setting the target to open the URL in either a new tab or a new window, does not work. Clicking on the link in the case of iframe and a new tab, results in no action. Using Firebug console, it is possible to see the error message being thrown. In the case of a new window, the result is a blank page.

Same screen display is not a good solution since the user will lose the interview page. Moreover on using the back button, the user will be redirected to the start screen in Webcenter portal.

In order to be able to display commentary in an iframe in Webcenter, a span tag must be used together with an onclick event to mimic a link. Clicking on the span tag will trigger an event that changes the source URL of the commentary iframe. This tag does not get changed by the ADF library and as such behaves as expected.

To deploy this in Webcenter, set the value of use-javascript-for-commentary-display in appearance.properties to true; for accessibility reasons, by default it is assumed to be false unless explicitly set to true since not all browsers may support Javascript.

Both screen commentary and control commentary have corresponding javascript template versions that are used instead of the normal ones when use of Javascript is enabled through appearance properties.

The display styles for screen commentary and question control commentary are set using javascript-link-question-style.

The Javascript versions of templates can be found in the [templates/javascript_links/](#) directory.

Set the portlet size in Webcenter

It should be noted that the Interview Portlet does not have a fixed size. A portlet's contents are within a DIV that is rendered by the portal; the height of which is determined by the content inside it while the width is determined by the enclosing container. In the case of the Interview Portlet, these are the enclosing DIVs that have been provided by the portal. The Interview Portlet window does not have a fixed size in (meaning that the size remains consistent as you step through the various screens) in Webcenter.

In Webcenter, when the Interview Portlet is added to a page that uses the *Applications Page Template*, the portlet window size is not consistent, but changes when stepping through an interview. Using the tool highlighted in the shot below, the container size can be set (size of the *Add Content* box); this will fix the fluctuation in size of the portlet window. It is also possible to set the size of the portlet itself using a similar tool which is just below the highlighted tool.

Note: The screen shown below is an example only and may not be the same screen that you will be presented with - however, the tools and method used to set the size, should be the same.



A screenshot of the Webcenter user interface. At the top, there is a navigation bar with tabs for 'Personal', 'Small Business', 'Commercial', and 'Customer Support'. Below this is a header area with 'Editing Page: New Page' and options for 'View', 'Page Properties', and 'Reset Page'. The main content area is enclosed in a dashed box and contains a '+ Add Content' button. Below that is a dropdown menu for 'OPA Interview Portlet'. Underneath the dropdown, there is a section titled 'Select rulebase in which to conduct an investigation:' with a list of four options: 'Parents and Children', 'HealthyEating', 'SimpleBenefits', 'everything', and 'commentaryExample'.

In Webcenter, the Interview Portlet's window size is affected by the page template being used. The window size is constant if we use either the *Swooshy* or the *Globe* page template.

Custom development

This section provides you with information regarding the customization of your Oracle Policy Automation installation. It should be noted that the API documentation provided in the Help directory of the Oracle Policy Automation installation is intended as your primary source of information. The information provided in the following topics, should be regarded as supplemental to the information found in those documents.

Topics in "Custom development"

- [Understand which API to use](#)
- [Embed determinations inside another application](#)
- [Implement a custom interview client](#)
- [Loading rulebases and rule modules in the Determinations Engine](#)
- [Introduction to plugins](#)
- [Write a Custom Formatter](#)
- [Write a Custom Function extension](#)
- [Maintain a Custom Function handler](#)
- [Migrate a Custom Function handler to a Custom Function extension](#)
- [Customize the existing Web Determinations user experience](#)
- [Customize the inferencing cycle with custom functions and inferencing listeners](#)
- [Initialize a new session with a rulebase listener](#)
- [Create a custom interview user experience](#)

Understand which API to use

What do you want to do?

Embed determinations inside another application

Create a custom interview client

Take action while rules are being processed

Change the format of returned data values

Embed determinations inside another application

If you want to make use of determinations within your application the best choices are the API to the Determinations Engine, or the Determinations Server.

The Determinations Engine API provides a direct programming solution for Java or C#, allowing you to quickly integrate determinations without the need to conform to a service communication medium (like XML in the case of the Determinations Server). Accessing the Determinations Engine in this way is also very fast, because values can be got and set directly (again, without the need to communicate via XML).

The disadvantage of using the API for integration is that the engine can only provide determinations services within the virtual machine (Java or .NET for C#) of the application; it is not able to act as a service. Additionally, even though the engine is fast and efficient, it will be difficult to scale as the demand for determinations increases, as it must continue to exist within the virtual machine of your application.

Currently the Determinations Engine only supports Java and C# (for .NET).

Oracle Determinations Server provides a way to make use of determinations within your application by accessing them as a service. Communication with the Determinations Server is done through SOAP/XML over HTTP protocol. This allows many different applications to include determinations, although this will incur a communications overhead, which is necessarily less efficient than direct integration via the Determinations Engine API.

Because the Determinations Server communication is done through XML over HTTP, any application can use this service, regardless of the language it was written in, as long as it can process XML and use the HTTP protocol.

Also, using the Determinations Server allows you to centrally control, update and administer the rules and rulebases deployed as well as allowing for simple scalability through clustering and load balancing across multiple Determinations Servers.

Key attributes for Engine API vs Determinations Server.

Determination Engine API (direct integration)

- Fast, efficient, and simple API
- Java or C#
- Limits to scalability
- Accessed only within the application virtual machine.

Determination Server (determinations as a service)

- Increased overhead XML/HTTP
- Scalable
- Centrally managed
- Service oriented
- Useable by anything that can use XML/HTTP.

For more information on using the Engine API or the Determinations Server, see the documentation for Oracle Policy Automation Java or .NET.

Create a custom interview client

For a description of the process involved in creating a custom interview client, go to the topic, [Create a custom interview user experience](#). For further guidance refer to the API documentation in the Help subdirectory of your Oracle Policy Automation installation (this should be your primary source of information when performing customization).

Take action while rules are being processed

Events are fired by the triggering of an event rule. They are typically used for informing the rule-based application that something important has happened that might require action.

A common use for events is checking values entered by a user. A rule could be written that tests if the applicant's date of birth is after today's date, and if so triggers an event describing the error.

Change the format of returned data values

The Oracle Determinations Engine provides the capability to load the custom function and custom formatters directly into the session itself, thereby creating a unified configuration method for all client applications including the:

- Oracle Determinations Server
- Oracle Policy Modeling Standalone Debugger
- Oracle Policy Modeling Standalone Regression Tester

Go to the [Rulebase configuration file](#) topic for details on how configure the Oracle Determinations Engine to load these custom components. Note that Web Determinations for Java requires custom formatters and custom functions to be configured via its unique configuration method.

Embed determinations inside another application

What do you want to do?

- Load a rulebase into the Determinations Engine
- Understand the Determinations Engine rulebase data model
- Understand the Determinations Engine inferencing cycle
- Find out what data the Determinations Engine needs to make a decision
- Pass data to the Determinations Engine
- Get a decision from the Determinations Engine
- Work with Determinations Engine data that changes over time

Load a rulebase into the Determinations Engine

In the Oracle Determination Engine API, a rulebase must be loaded into the engine, before any of the rules written and compiled in that rulebase can be used. The **com.oracle.determinations.engine.Engine** class is responsible for loading a rulebase.

When loading a rulebase, you should pass the path (relative or absolute) to the rulebase archive (.zip) file. When you build the rulebase in Oracle Policy Modeling, this archive is automatically built in the rulebase project's output directory.

The rulebase archive name is always the name of the rulebase with a .zip extension. Once built, the rulebase archive can be moved out of the output directory of the rulebase. Nothing else is needed other than the Determinations Engine runtime files to execute the rules.

Java example:

```
Rulebase simpleBenefits = engine.getRulebase("SimpleBenefits.zip");
```

C# example:

```
Rulebase simpleBenefits = engine.GetRulebase("SimpleBenefits.zip");
```

Understand the Determinations Engine rulebase data model

When using the Determinations Engine API, it is important to understand how to interact with the engine in order to use a rulebase correctly.

The rulebase

The rulebase contains everything defined in Oracle Policy Modeling project: the rules, the data model used by the rules to reach determinations and screens can all be referenced through the rulebase. Typically, the first thing you do when interacting with the engine API is to load at least one rulebase

The session

A session contains all the data that a rulebase will be used to use in determinations. Entity instances, attribute values for those instances including inferred attribute values and relationships are all represented in the session.

After creating a session for a rulebase, you can set data, including entity instances, attributes and relationships to that session. Any inferred values are got from the session.

You may have several sessions from the same rulebase at the same time. Each session can have its own data, and will reach determinations based only on the data in that session.

The entity and entity instance

Entities are defined when you create a rulebase using Oracle Policy Modeling. It represents a type of "thing" (for example a person). An entity can have attributes, and also relationships to other entities.

At runtime, you can create one or more instances of an entity; for example if your rulebase has a person entity, at runtime you might create one or more instances of the person entity. These entities represent data and will contain values for attributes, and may also be the source or targets of relationships.

All entity instances must be "contained" by another instance; for example, if the entity you want to create is contained by the global entity, then pass the Entity (type) you want to create as the first argument and the instance of the global entity as the second argument.

During runtime, an entity instance is created and held in the session.

In object oriented programming terms, an entity can be thought of as a class. It acts as a template to create instances, and an instance can be thought of as an object; an instantiation of the entity (class).

Example: creating an entity instance *child1* – from the child entity

In this example, the global instance contains the child entity and is therefore required. The **markContainmentComplete** method indicates that all instances of *the child* entity have been collected; this method should be used after all entity instances of the child have been created.

```
EntityInstance global = session.getGlobalEntityInstance();
EntityInstance child1 = session.createEntityInstance(child, global);
global.markContainmentComplete(true, child);
```

The attribute

Like entities, attributes are defined as part of authoring a rulebase in Oracle Policy Modeling. An attribute represents a simple value, like a number, or a text string and is always attached to an entity. When an entity instance is created, value for an attribute can be set for that instance. Each instance has its own value for every attribute defined for

Example: setting the value of the attribute *child_age* on the entity instance *child1*

```
Entity child = simpleBenefits.getEntity("child");
Attribute child_age = child.getAttribute("child_age");
child_age.setValue(child1, 16);
```

Depending on the type of attribute the value set will be different.

Type	Java Type	C# Type
Number	java.lang.Double	System.Double
Currency	java.lang.Double	System.Double

Type	Java Type	C# Type
Text	java lang.String	System.String
Date	java.util.Date	Oracle.Determinations.Masquerade.Util
DateTime	java.util.Date	Oracle.Determinations.Masquerade.Util
Time of Day	com.oracle.determinations.engine. DeterminationsEngineTimeOfDay	Com.Oracle.Determinations.Engine. DeterminationsEngineTimeOfDay

The relationship and relationship instance

Relationships are also defined as part of rulebase authoring. A relationship represents a connection between two types of entities with one end being the "source" and the other end representing the "target"; for example, a parent entity may have a one-to-many relationship to a child entity.

At runtime an instance of a relationship can be created to link entity instances together. Like an entity instance, relationship instances are created (and stored) in a session.

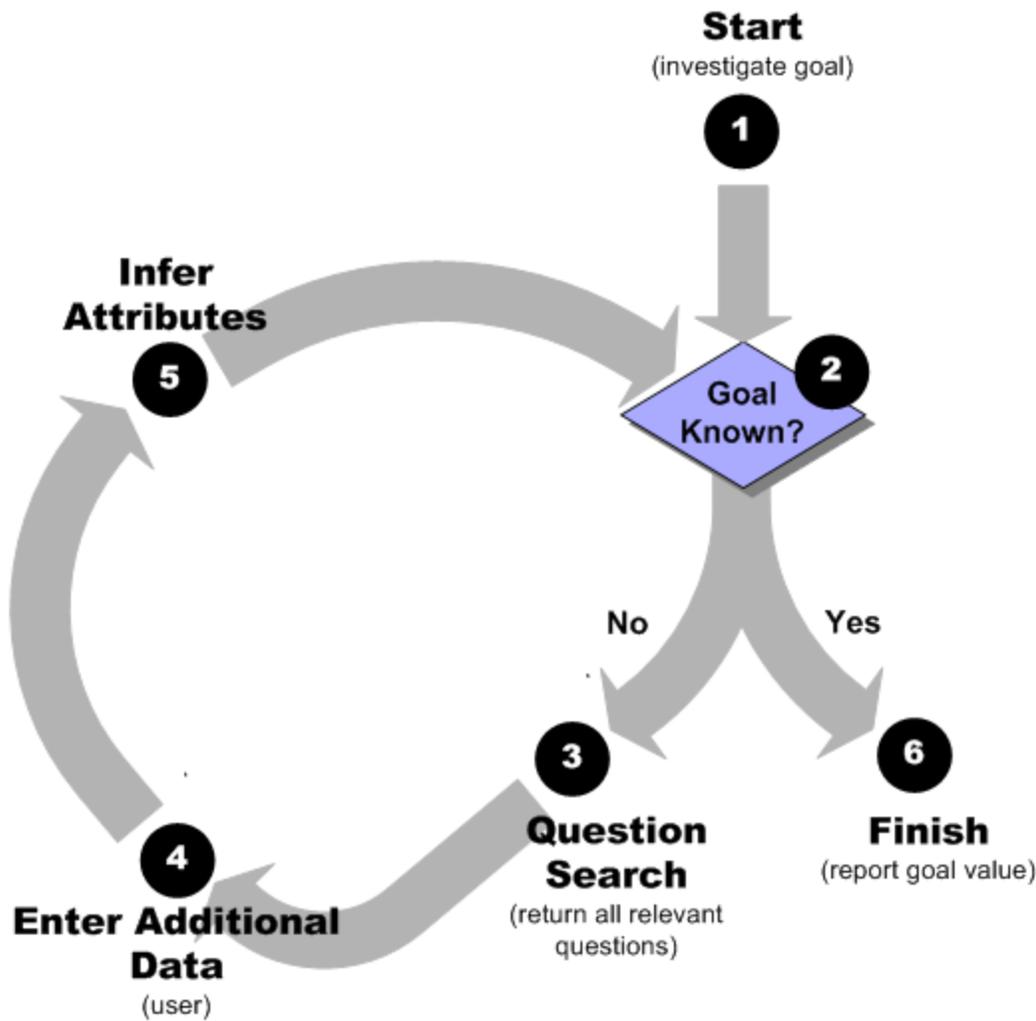
Example: setting a relationship from the global to child entity instances

```
EntityInstance child1 = session.createEntityInstance(child);
EntityInstance child2 = session.createEntityInstance(child);

claimantschildren.setInstance(global,
    Arrays.asList(new EntityInstance[] {child1, child2}));
```

Understand the Determinations Engine inferencing cycle

The Inferencing Cycle is the cycle of question and answer which operates on rules to replicate the decision making process, as shown in the following illustration:



The diagram above shows the following steps:

1. **Start (investigate goal):** An attribute is specified as the goal attribute to be investigated.
2. **Goal Known?:** The Oracle Determinations Engine determines whether or not the goal attribute has a value.
3. **Question Search:** The Oracle Determinations Engine finds all known (or unknown) attributes that influence the goal (an inferencing process known as backward chaining) then reports any influencing attributes that are unknown. Another way of thinking about this is that the Oracle Determinations Engine is asking, "What do I need to find out to prove this attribute?".
4. **Enter Additional Data:** The Oracle Determinations Engine waits for a value(s) to be input for the attribute(s) raised by the question search.
5. **Infer Attributes:** The decision tree is traversed by the Oracle Determinations Engine in the reverse direction, with conclusions drawn from known attributes. This inferencing process is known as forward chaining. Another way of thinking about this is that the Oracle Determinations Engine is asking, "What can I conclude based on the collection of what I know?".

- 6. Finish:** Once the goal attribute is known, the Oracle Determinations Engine reports the value and how it reached that decision (if requested). The Oracle Determinations Engine generates a decision report (if required) using backward chaining, as described above.

The Inference Cycle will repeat steps 2 to 5 until the goal attribute is known.

Find out what data the Determinations Engine needs to make a decision

When using Oracle Policy Automation within an application, you may need to determine what information is required to reach a particular determination or goal; for example, if you have a goal "the claimant is eligible for rent assistance", you may need to determine what information about the claimant is required to determine that outcome, so that you can retrieve it from a data source, or prompt the user to enter that information.

You can use a decision report to examine what data you need to determine a particular goal. A decision report can be used to determine how a particular answer was reached for a goal, but it can also be used to determine what information is needed to reach a decision.

A decision report is basically a tree of attributes and relationships that contributed to a decision. If a goal is Unknown, then one or more contributing attributes or relationships will also be unknown. By examining a decision report, and determining which of the base level attributes are unknown, you can tell what information you need to provide in order to obtain an answer for your goal.

Example: eligible teenage allowance

In the *SimpleBenefits* rulebase there are some simple rules for determining the goal ""

the claimant is eligible for the teenage child allowance if

for at least one of the claimant's children

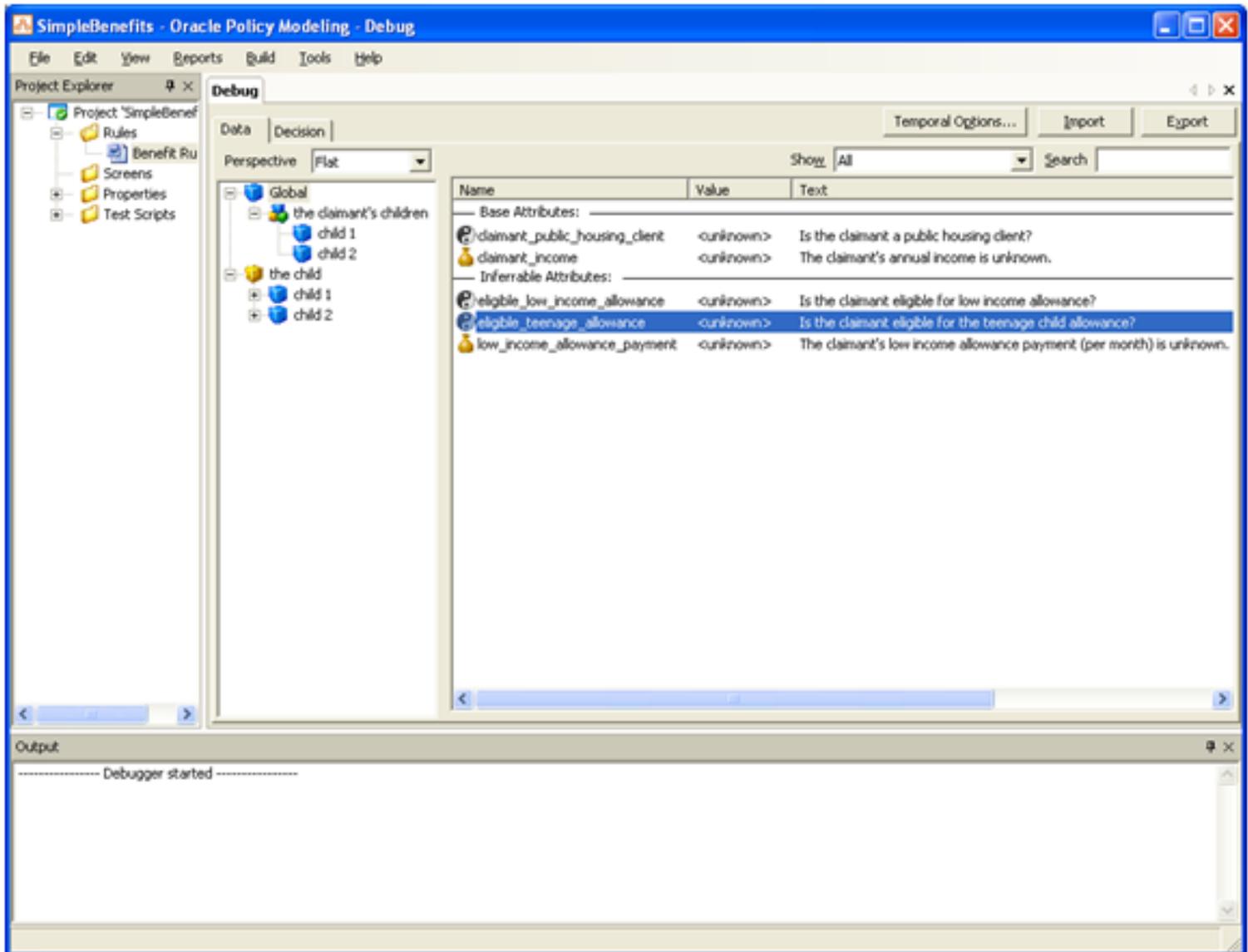
the child is a teenager

The child is a teenager if

The child's age \geq 13 and

The child's age $<$ 20

If we have a simple session with two child entities and no other attributes set, notice that the goal *the claimant is eligible for the teenage child allowance* is unknown.



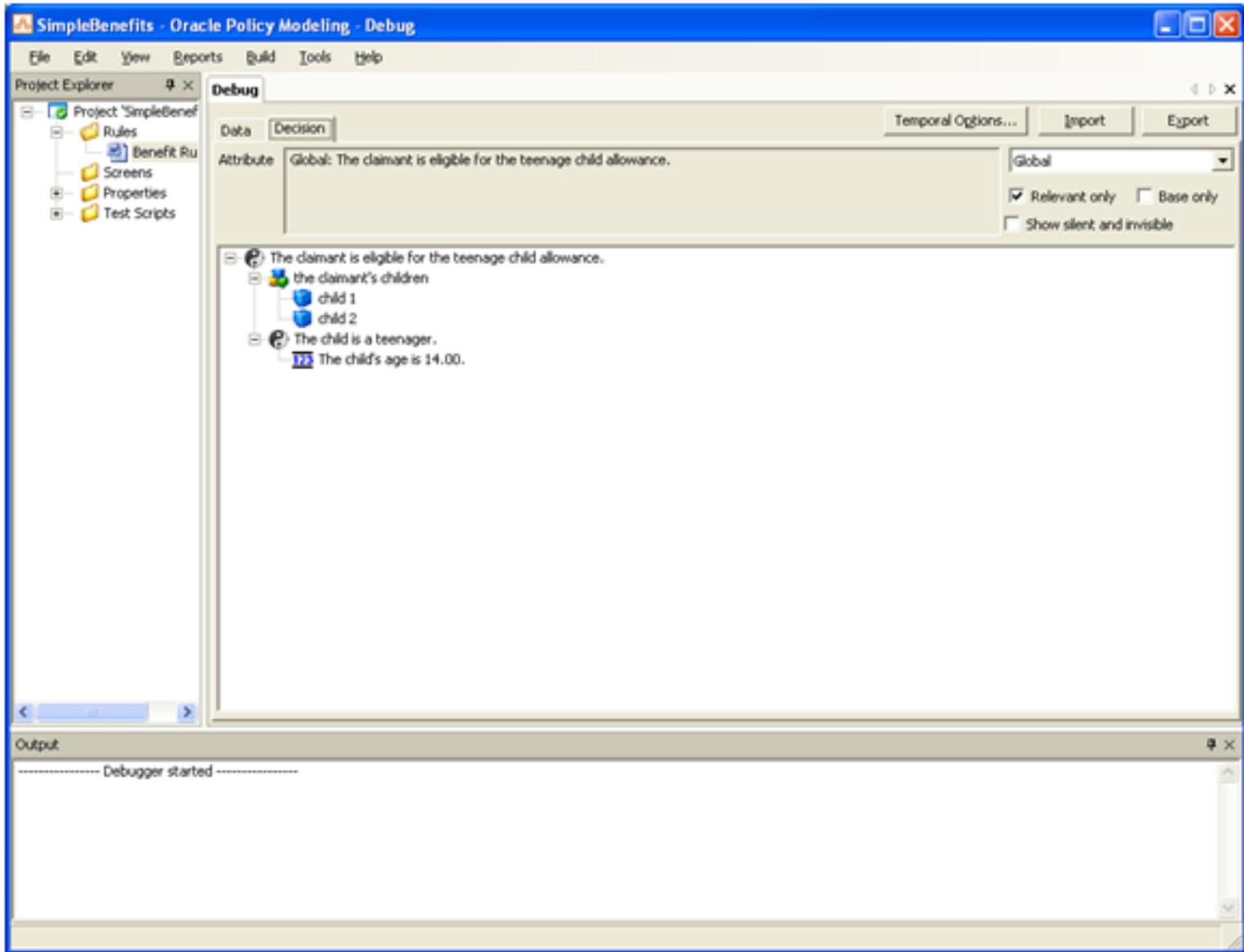
If we investigate the attribute (right click and choose investigate) the decision report for the goal will be displayed. The decision report displays the relevant (contributing) attributes for the goal.

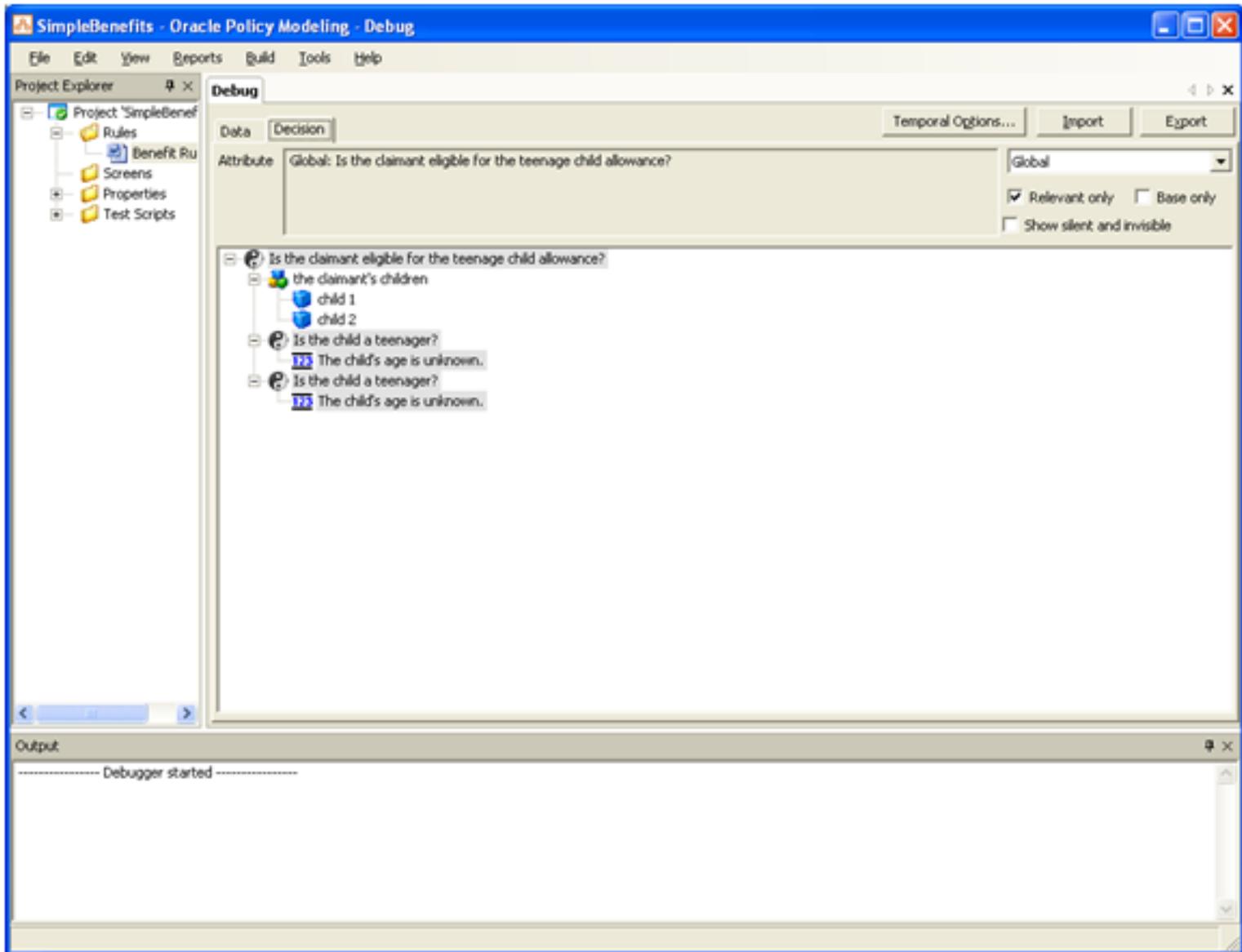
The decision report clearly shows the following:

- There is a contributing relationship with two targets child 1 and child 2.
- For each child, the child is a teenager is unknown, but this attribute is inferred from another attribute on the child entity, the child's age.
- Further, for each child, the child's age is unknown.

So given this information, we can see that we need to provide the ages for the child instances.

If we set one of the children's age to 14, the goal becomes known. If we ask for a decision report (right click and choose **Show Decision**) we can see that the decision was reached because one child is a teenager, because their age is 14. The second child becomes irrelevant because for the goal to be true, we only need one teenage child.





Pass data to the Determinations Engine

For information regarding passing data to the Determinations Engine, refer to the topic, [Understand the Determinations Engine rulebase data model](#).

Get a decision from the Determinations Engine

The main function of the Determinations Engine is to be able to get an answer to a goals attribute (or a goal relationship) from a Determination Engine session. These goals will be inferred from the data that is provided to the session, using the existing rules in the rulebase.

There is a very simple process for obtaining the result (or decision) from a rulebase session; the procedure is as follows:

1. Create a session
2. Add base level data
3. Think (*session*).
4. Ask for the answer or decision.

After the **think()** method is called on the session, the rules will fire, inferring all attributes and relationships that can be proved by the current rules and the data provided. Once the think has finished you can then get the value of the inferred attribute. The value returned will be in one of three states.

State	Description	Return value
Known	A value was successfully inferred	A value appropriate to the type of the attribute: <ul style="list-style-type: none"> • Currency or Number will be a Double • Text will be a String • Date or DateTime will be a Date • TimeOfDay will be <code>com.oracle.determinations.engine.DeterminationsEngineTimeOfDay</code>
Uncertain	The value of the attribute/relationship cannot be determined with the current data.	The value returned will be an instance of the <code>com.oracle.determinations.engine.Uncertain</code> class
Unknown	More information is needed to get a value	The value returned will be null

Example: getting and printing the value `eligible_low_income_allowance` from the global entity instance.

```

session.think();

Attribute eligible_low_income_allowance = globalEntity
    .getAttribute("eligible_low_income_allowance");

if (eligible_low_income_allowance.isUnknown(global)) {
    System.out.println("eligible_low_income_allowance is unknown");
}
else if (eligible_low_income_allowance.isUncertain(global)) {
    System.out.println("eligible_low_income_allowance is uncertain");
}
else {
    Double value = (Double) eligible_low_income_allowance
        .getValue(global);
    System.out.println("eligible_low_income_allowance = "

```

```
} +value);
```

If the value is known, then you have successfully inferred your goal. If the value is unknown, then more information is needed, you can use a decision report to work out what extra information you need to provide to obtain an answer.

If the value is uncertain, that means that because of the rules and data provided there is no way an answer can be provided, no matter how much more information you provide.

An example of an uncertain value might be "the age of the person's oldest child". Where the person has no children, the answer for this would be uncertain, as there is no number value that would make any sense.

Work with Determinations Engine data that changes over time

It is possible to specify the values of an attribute that changes over time; for example, when putting in the values for a claimant's income, you may want to enter that data over a period of time where their income changed.

Engine API

You can do this by setting the value of an attribute to a temporal value, using the classes in the *com.oracle.determinations.engine.local.temporal* package. Using these classes you can set different values of an attribute for different periods of time.

Example Setting data that changes over time via the engine API

If we had a claimant who receives an annual income, we may want to represent that income as it varies over time.

- Initially the claimant's income at 15000
- On the date 13/5/2007 the claimant's income changes to 16000 (a promotion?)
- On the date 21/6/2007 the claimant's income changes to 12000 (no longer working?)

We can represent these changes over time in the following way:

```
Attribute claimantsIncome;  
EntityInstance claimant1;  
  
...  
  
// create the change point dates for the changes  
ChangePointDate date1 = new ChangePointDate(2007, 5, 13);  
ChangePointDate date2 = new ChangePointDate(2007, 6, 21);  
  
// the temporal value has an initial value of 15000, and  
// changes at date1 to 16000, and changes again at date2 to 12000  
TemporalValue incomeVal = new TemporalValue(new Double(15000),  
    date1, new Double(16000),  
    date2, new Double(12000));
```

```
claimantsIncome.setValue(claimant1, incomeVal);
```

Determinations Server

In the same way temporal values can be set through the engine API, you can also set values that vary over time in a Determinations Server request.

In this case we use the change-point element. The initial value for an attribute is the standard XML, but any variations to the value are contained in a change-point element.

The change-point elements must be put in ascending order (oldest to newest).

An empty value (meaning unknown) and a <uncertain-val/> element are also valid values.

Example Setting data that changes over time in a Determinations Server request API

```
<typ:attribute id="claimant_income">  
  <typ:number-val>15000</typ:number-val>  
  <typ:change-point date="2007-05-13">  
    <typ:number-val>16000</typ:number-val>  
  </typ:change-point>  
  <typ:change-point date="2007-06-21">  
    <typ:number-val>12000</typ:number-val>  
  </typ:change-point>  
</typ:attribute>
```

Implement a custom interview client

Create a portlet that embeds an interview experience in a portal

Portals are a widely used deployment among enterprises, both internally and externally. While they are an excellent means of providing a single access point to, and integration of, information across an organization, they do however come with their own unique set of programming problems.

Oracle Web Determinations provides an interactive front-end through which rulebases authored using Oracle Policy Modeling can be accessed; but for those wishing to provide a similar experience for portal users, it is generally not the best option.

While it is possible to expose Web Determinations deployments through a portal by using either iFrames or a more complicated portlet-bridge approach, the best and most scalable option is to produce an Interview Portlet. This provides access to all of the more complex features of the Portlet API as well as providing a properly integrated deployment.

A basic implementation of Web Determinations embedded in a portal can be found here:

[Tutorial: Embed Web Determinations within my portal - a simple implementation](#)

Embed interview logic directly in a client application with the Interview API

Although a portlet is designed around the Interview Service, it is possible that the Interview API will be chosen instead. The changeover of calls to the Interview API from the service only needs to be done in one location, however other changes are required. The JSP and portlet all reference the proxy classes in order to read and write the request and response objects from the Interview Service and these references need to be changed to utilize the API classes; for this reason, it is worthwhile giving consideration to what will be your chosen architecture in order to avoid the need for a switch later in the development cycle.

Where a change may be needed or the possibility of needing to use both the Interview Service and API is present, consider extending the functionality of the Interview Interface to completely wrap all the required classes, instead of directly referencing either the API or proxy classes; in this way it will only be necessary to swap the interview interface, though the development effort required is increased.

Share interview logic across multiple client applications with the Interview Web Service

While the Assess Service provides lightweight stateless access to deployed rulebases, the Interview Service has been built on top of the same platform that powers Web Determinations, providing you with a Web Service enabled method through which to conduct interviews. With the state stored in the Determinations Server, developers are free to concentrate on the specifics of rendering and displaying question and result screens to the user, without having to worry about the specifics of session storage and management.

The following are lists and descriptions of the available methods, as well as an example of a common process and an advanced process:

Available methods

The available methods and a description of each follows:

OpenSession, CloseSession

These methods start or end of any interaction with the Interview Service, all methods require a session token to be provided when then **OpenSession** method is first called. This token uniquely identifies the particular in-memory session. In order to stop the Web Server running out of memory, care should be taken to close sessions once the user has finished.

LoadCase, SaveCase, ListCases

These are the session management methods that provide the means to save and list the available session in the

session store. Because loading and saving are done using the Data Adaptor loaded on the Determinations Server, it is also possible to share the data store with Web Determinations, by using the same Data Adaptor.

ListGoals, ListScreens

These methods list the goals and screens that are available in the rulebase. the screen listing can be filtered by *Entity*, *Entity Instance*, *Attribute* and *Relationship*.

Investigate

The **Investigate** method is the crux of the entire service and is used to investigate a particular goal or flow. The next question screen required is returned by this method in response to either a blank request or a request containing a populated screen.

GetScreen, SetScreen

These methods allow you to get and set screens outside the context of the interview and are often used to retrieve screens such as the *Data Review* screen.

GetDocument, GetDecisionReport

These methods get either a populated document or decision report using the information stored in the session memory.

GetUserSetData

The **GetUserSetData** method retrieves the data in the session that has been set by the user. It is important to note that this concept is separate from the idea of base level, intermediate and goal attributes.

Common process

The following describes the process for a typical interaction with the Interview Service, involving a simple interview such as that for which support is implemented in the portlet as described in the [Tutorial: Embed Web Determinations within my portal - a simple implementation](#).

odsServer/ListRulebases

Request a list of the rulebases loaded into the Determinations Server and present the list to the user. This stage may or may not be recommended depending on the particular usage of the Determinations Server; not all rulebases loaded into the Determinations Server are suitable for running an interactive interview.

odsInterviewService_TestRulebase/OpenSession

The Determinations Server opens a new session in memory and returns a session token (token) that can be used to refer to the aforementioned session.

odsInterviewService_TestRulebase/ListGoals

By providing the token to the Determinations Server, a list of the available goals for the particular rulebase is provided, which can be presented to the user, allowing them to choose which goal to investigate.

odsInterviewService_TestRulebase/Investigate

At this point, a back and forth communication with the Determinations Server will be started, the first of which contains the token and the goal that is to be investigated. The response will contain a definition of the first screen to render to the user for them to complete. Further request and response pairs will contain populated screens and the next screen required respectively, until the goal is reached.

odsInterviewService_TestRulebase/CloseSession

Upon completion of the process, pass the token to a **CloseSession** request in order to "destroy" the session and free up memory for subsequent sessions.

Additional requests for advanced processing

The following additional requests can be made during the life-cycle of a session:

odsInterviewService_TestRulebase/SaveCase

At any point during the investigation process, the user may want to save their session, or you may want to have it saved for them automatically. This is done by calling the **SaveCase** method with the token and an identifier under which the case is to be saved. Loading the case in future is managed by using a **LoadCase** request containing that same identifier.

odsInterviewService_TestRulebase/GetDecisionReport

Once a goal has been reached (through being known, unknown or uncertain), you can then elect to retrieve and render the decision report for the user by passing the token and the goal in which you are interested.

odsInterviewService_TestRulebase/GetDocument

Upon completion of an investigation, it is common for the *Summary* screen to contain references to documents. These documents are dynamically generated based on the session context and can be retrieved by using the **GetDocument** method. This method should only be called when all of the required information is contained in the session, commonly indicated by the presence of a link appearing on the *Summary* screen.

odsInterviewService_TestRulebase/GetScreen

Two of the most commonly used screens are the *Summary* and *Data Review* screens. These can be accessed through the **GetScreen** method and rendered to a user in preference to a custom goal listing or a session data presentation.

See also:

[Tutorial: Embed Web Determinations within my portal - a simple implementation](#)

Load rulebases and rule modules in the Determinations Engine

The following information details how to load rulebases and rulebases that rely on modules, in the Determinations Engine.

The basics of loading a rulebase

The Determinations Engine provides the following two methods for loading a rulebase:

1. **File based:** Rulebases can be loaded by specifying a path to the rulebase archive on the local file system.
2. **Stream based:** Allows rulebases to be loaded by providing an `InputStream` to the rulebase archive. This can be used to load rulebases from remote file systems, databases or other storage.

File based rulebase loading sample

```
String rulebasePath = "/path/to/my/rulebase.zip";
Engine engine = Engine.INSTANCE;
Rulebase rulebase = engine.getRulebase(rulebasePath);
```

Stream based rulebase loading sample

```
String rulebasePath = "/path/to/my/rulebase.zip";
FileInputStream rulebaseStream = new

FileInputStream(rulebasePath); Engine engine = Engine.INSTANCE;
Rulebase rulebase = engine.getRulebase(rulebaseStream);
```

Load a rulebase that references a module

Unlike a standard rulebase where the entire definition of the rulebase is contained in a single archive, a rulebase that uses modules has its definition spread over multiple archives (the rulebase archive plus a separate archive for each module it references).

In order for the rulebase to be loaded correctly, the Determinations Engine must be able to find and load each module; the responsibility for this is given to the **ModuleResolver**.

Loading a rulebase that relies on a module is similar to loading standard rulebase; the appropriate **Engine.getRulebase(...)** method is called and supplies either the path to the rulebase archive or the archive itself as a stream. Additionally, it supplies a **ModuleResolver** which is responsible for locating and providing the required module archive.

If a rulebase containing module references is loaded by supplying a path to the rulebase only (see the *File rulebase loading sample* above), then the engine will attempt to find and load any dependent modules from the same directory as the rulebase archive was loaded in. However, if a rulebase that references modules is loaded from an input stream, the caller must supply a **ModuleResolver**, otherwise the rulebase will fail to load.

The default implementation of the **ModuleResolver** provided by the Determinations Engine is the **FileModuleResolver**, that searches for modules in the specified directory on the local file system. Third parties may provide custom module resolution behavior by creating a class that implements **ModuleResolver**.

Example ModuleResolver

The following module sample illustrates how to write a custom **ModuleResolver** to load modules from a database. In this specific example, the rulebase, and its associated module is located in a simple Derby database. To run the supplied compiled example:

1. Create and configure the database according to the instructions in the [Rulebase Resolver - sample code \(DerbyRulebaseResolver\)](#) example.
2. Load the **IncomeSupportBenefit** into the database (this can be done using the BlobInsert sample in the [Rulebase Resolver - sample code \(DerbyRulebaseResolver\)](#)).
3. Copy the Determinations Engine and its dependencies to your deployment directory.
4. Copy the Derby client library to your deployment directory.
5. Run the jar file from the command line; for example:
`java -jar <jar file name>`

See also:

[Rulebase Resolver - sample code \(DerbyRulebaseResolver\)](#)

Write a Custom Formatter

Note: If using Web Determinations, writing a Formatter plugin may be required instead, to allow interpreting of user input as well as displaying values. See [Formatter plugin overview](#) for more information.

A formatter is used by the Determinations Engine when:

- **Attribute.getFormattedValue** is called
- An attribute's value is substituted into any text retrieved by the determinations engine; for example, "the person's income is \$60,000". This includes decision reports.
- Concatenating text values with other values together in a rule (non-text values are formatted).

A formatter is a class that implements the **com.oracle.determinations.engine.Formatter** interface (in Java) or the **Oracle.Determinations.Engine.Formatter** interface (in .NET).

The **canFormatType** and **canFormatTemporal** methods can be implemented to prevent the formatter being called with unsupported types or with temporal values. In the case of temporal values, a fallback implementation is used that uses the custom formatter to format each individual change point value instead of the whole temporal value.

The **getFormattedValue** method should convert the value, a boxed type (for example, **java.lang.Double** or **Oracle.Masquerade.Lang.Double**), into a string representation of that value. The attribute whose value is being formatted (if available) is also supplied so formatting can be overridden on a per-attribute basis.

Installation of the custom formatter is done by providing a rulebase configuration file that specifies the name of the class to be used when the rulebase is loaded.

See also:

[Rulebase configuration](#)

[Formatter plugin overview](#)

Write a Custom Function extension

Note: This topic describes Custom Function extensions, which have superseded custom function handlers as of 10.2. For information about the old mechanism, see [Maintain a Custom Function Handler](#).

A custom function is a method of extending the set of functions available to a rule developer. It is more limited than an inferencing listener in that it can only return a single value at a time. A custom function cannot modify a session except indirectly by returning a value that is used in a rule.

However, the advantage of custom functions is that they fully participate in backwards chaining, the process by which the return value (the goal) is traced back to its dependent attributes. Essentially this means a custom function can:

- Determine which parameters to show in a decision report.
- Cause attributes to be collected if they are unknown and their value is required for the custom function.

Custom functions can be written in either Java or .NET, depending on the target platform for the deployed rulebase; for example, the Debugger (which is written in .NET) requires a .NET implementation of the custom function in order to debug a rulebase. It is also possible to supply both a Java and a .NET implementation and the rulebase will then work on either platform.

From a rule developer's perspective, a custom function extension is a folder that is copied into the *Extensions* folder of any rulebase project. The custom function is called in exactly the same way as a built-in function, with the compiler verifying correct usage. At build time, the required information and supporting DLLs/JARs are copied into the rulebase zip so that it can be automatically loaded by the engine when the rulebase is loaded.

See also:

[Custom Function extensions](#)

Maintain a Custom Function Handler

A Custom Function Handler is a Java or .NET class that implements the **com.oracle.determinations.engine.CustomFunctionHandler** interface (in Java) or the **Oracle.Determinations.Engine.CustomFunctionHandler** interface (in .NET). A single method `evaluate` (or `Evaluate` in .NET) is implemented to evaluate the function and return the result.

The **CustomFunctionDeclaration** parameter supplies a definition for the custom function, the most useful property of which is the name of the custom function (in case more than one custom function is required). Parameters to the function are supplied as an object array, although these objects will all be string objects in practice.

In a rulebase, a custom function handler is called by using the **CallCustomFunction** function:

The highest number	
<code>Number(calkustomfunction("Maximum","n1,n2,n3"))</code>	The first number is known The second number is known The third number is known
<code>uncertain</code>	<code>otherwise</code>

The first parameter is the name of the function which will come through the **CustomFunctionDeclaration**. The second parameter is a comma-separated list of identifiers, which will be passed to the custom function handler via the object array. Installation of a custom function handler can be accomplished in one of two ways:

- The custom function handler can be installed manually in an application that uses the engine directly (by calling **Session.setCustomFunctionHandler**).
- The handler can be packaged in the compiled rulebase zip file. To do this the compiled .NET assembly DLL or Java JAR file must be copied into the include folder of the rulebase project along with a rulebase configuration file that specifies which class implements the custom function handler.

See also:

[Rulebase configuration](#)

[Migrate a Custom Function Handler to a Custom Function Extension](#)

Migrate a Custom Function Handler to a Custom Function Extension

Custom Functions Handlers (previously known as simply "custom functions") have been deprecated in version 10.2 of Oracle Policy Automation in favour of Custom Function Extensions, a strongly typed method of defining custom functions that is more powerful and allows for easier reuse across projects.

Some comparisons are given below:

	Custom Function Handlers	Custom Function Extensions
<i>Invocation</i>	Invoked using "CallCustomFunction", parameters must be attributes with public names; for example, Result = CallCustomFunction("Foo", "p1,p2,p3")	Invoked just like a regular function; for example, Result = Foo(parameter1, parameter2, etc)
<i>Context</i>	A rule that uses CallCustomFunction needs "known" conditions on all dependent attributes as a hint to the determinations engine which attributes should cause the rule to be re-evaluated if they change. This typically requires the function to be placed in a table rule in Word, and also precludes its use in Excel.	No special operators are required, the custom function can be used anywhere a built-in function can.
<i>Validation</i>	Rule documents that call the custom function are not validated to ensure they use the function correctly.	Name of the function, return type and parameter types are all strongly typed. These are all validated when compiling a rule document that calls the function.
<i>Decision Reports</i>	Does not participate in decision reports, although the "known" clauses mentioned above can simulate this.	By default all parameters appear in decision reports, more complex relevance can also be implemented.
<i>Unknown and Uncertain</i>	Custom function implementation must deal with possible unknown and uncertain values.	Unknown/uncertain is automatically handled, although this can be overridden.
<i>Temporal Reasoning</i>	Custom function implementation must deal with change points or the calling application (and rules) must avoid using them.	Custom function automatically supports change points. Custom behaviour can also be implemented.
<i>Deployment</i>	Java/.NET Implementation can be packaged with compiled rulebase but this requires manual configuration in each project that uses it.	Java/.NET implementation and function definition is packaged together and can be freely copied between projects if desired.

The steps to migrate a custom function handler to a custom function extension are as follows:

1. Create the definition of the functions and the argument they take inside a new *extension.xml* file.
2. Rewrite the custom function class to extend the **CustomFunction** class instead of implementing the **CustomFunctionHandler** interface. The main code changes to note are:
 - a. If the old class handled multiple custom functions, you will probably want to create a separate class for each function.
 - b. Although the **CustomFunction** class has more methods, most of them have default implementations other than evaluate.
 - c. The evaluate has a different signature and slightly different semantics, most notably the arguments passed to the function are no longer the names of attributes to look up, they are now the values passed to the function. Looking up attributes is usually not necessary.
 - d. Custom function handlers need to deal with the possibility of unknown and uncertain values of attributes, but with a custom function extension, this is not necessary unless the **requireKnownParameters** method has been overridden to return false.
3. Package the compiled DLL or JAR files with the *extension.xml* file as an extension and copy it into the rulebase project's "extensions" folder.
4. Rewrite rules that call the custom function; usually this will mean:
 - a. Instead of invoking **CallCustomFunction**, with a pair of quoted strings, the function can be called by name and the arguments directly inserted into the function call.
 - b. "known" conditions used with **CallCustomFunction**, can usually be removed. This often means the function call no longer needs to be in a table rule (although of course that is still permissible).

See also:

[Write a Custom Function extension](#)

[Custom Function extensions](#)

Customize the existing Web Determinations user experience

What do you want to do?

Modify the behavior of Web Determinations

Implement a rulebase-specific look and feel

Modify the behavior of Web Determinations

Oracle Web Determinations allows you to deploy Oracle Policy Modeling rulebases into a configurable and extendable web application out of the box.

There is a comprehensive extension architecture available to technical consultants/system integrators with extension points to accommodate specific project implementation needs.

Web Determinations extensions are custom Java or .Net classes/packages that are developed by system integrators and deployed onto a Web Determinations web application.

There are two main types of interactive extensions:

1. [Plugins](#)
2. [Event Handlers](#)

When extending Web Determinations, both of these extension types do not need to be used exclusively. They both provide complementing extension points, so it is most likely that both plugins and Event Handlers will be used to achieve the desired project-specific behavior.

What is a plugin?

In the context of Oracle Web Determinations, the word **plugin** is ambiguous as it can mean either a single class that implements an exported interface, or the compiled JAR or DLL containing multiple class definitions; from this point on, these are referred to as **plugin class** and **plugin archive** respectively, and the interface they implement is a **plugin interface**.

Plugins are constructed in two stages:

1. At application startup, an instance is constructed by reflection, with a no-argument constructor. This is called the **factory instance**.
2. Plugins are initialized by the factory instance, at different times and with different arguments for each different plugin interface. The factory instance is responsible for returning a fully-initialized **actor instance** which may be a copy of itself, or a newly constructed object. Alternatively, the factory instance may decline to attach by providing no actor instance at all; it is not really necessary to create factory instances as static methods with a specified name and signature could be used instead.

Kinds of plugins

There are several varieties of plugin interface:

- **Engine plugin:**
This is a plugin to the Interview Engine, that could be used in other contexts outside Oracle Web Determinations.
- **Platform plugin:**
This is a plugin to Oracle Web Determinations.

- **Session plugin:**

This is a plugin specific to an Oracle Web Determinations session - each session has its own instance of the plugin class.

- **Singleton plugin:**

This is an engine plugin which is instantiated once at engine startup and shared by all sessions. Singleton plugins must be re-entrant and will be accessed simultaneously by multiple threads.

Some examples:

- the Document Generation plugin is an Engine Session plugin
- the resource loader is a Platform Singleton plugin

General procedure for the creation of a plugin

To create a plugin:

1. Determine what plugin is to be extended
2. Locate the Web Determinations plugin folder inside the web application
 - a. Tomcat (Java) - default is in the `WEB-INF\classes\plugins` path in the interactive web application in the Tomcat webapps folder; for example:
`C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\interactive\WEB-INF\classes\plugins`
 - b. IIS (.Net) - by default, this in `C:\inetpub\interactive\plugins`
3. Determine what plugin interfaces need to be implemented
4. Create a class file, and implement the required interfaces and methods
5. Develop the methods
6. Package the class files
7. Deploy the packaged source into the plugins folder
8. Add needed class libraries in the lib directory
9. Restart the server
10. Test

Configure plugins from the System Administrator's perspective

When deploying Oracle Web Determinations, plugin archives must be placed in a specific location within the deployed web application (classes/plugins). This location cannot be changed by configuration. In Java, plugin classes must be archived into a JAR (no naked .class files). Other non-plugin files, such as dependencies or configuration files for the plugin classes, may also be placed in the plugins directory; they will be ignored by Oracle Web Determinations.

On application servers which allow us to search the plugins directory (such as Tomcat and IIS) every JAR (for Java platforms) or DLL (for .NET platforms) in the plugins directory will be examined to determine if it is a plugin archive. For the convenience of the systems administrator who may wish to deploy large numbers of plugin dependencies without incurring extra delay at startup, sub-directories of the plugins directory will not be searched.

On application servers which do not allow us to search the plugins directory (such as WebLogic and WebSphere) plugin archives must be named explicitly in the Oracle Web Determinations configuration file.

On any application server, even if the plugins directory can otherwise be searched, when plugins are specified in the configuration file, no other unnamed plugins will be loaded. This allows for consistent behavior when the same Oracle Web Determinations deployment is moved between application servers. It also allows an Oracle plugin archive to be overridden by a customer plugin archive while still being available for the customer plugin archive to use.

Plugin initialization order

Plugins are initialized in batches, according to the interfaces they implement. Note that within each batch, the order of initialization is not specified:

- At application startup (generally just once, however web servers may be configured to run several copies of the application)
 1. all Engine Singleton plugins
 2. all Platform Singleton plugins
- At the beginning of each session:
 1. all Engine Session plugins
 2. all Platform Session plugins

Multithreading

Oracle Web Determinations is heavily multithreaded, and attention must be paid to threading issues to produce correct behavior.

- **Session-specific plugins:**

Web Determinations assigns each interactive session to a single thread, so instances of session-specific plugin classes will be single-threaded and require no special attention in normal methods.

Any static members shared between class instances will require synchronization between threads.

If a factory instance returns the same actor instance for multiple different sessions, the plugin must be fully re-entrant!

- **Singleton plugins:**

All singleton plugins must be fully re-entrant and will be run simultaneously in multiple threads.

Implement a rulebase-specific look and feel

The following provides a discussion of the steps required and the available options for implementing a rulebase-specific interview look and feel in Web Determinations.

Rulebase custom properties

Oracle Policy Modeling provides a way to attach key/value property pairs to individual screens and controls at rulebase authoring time. These are then exposed to the templates that render the Web Determinations user interface through the platform's screen and controls API.

Sample code

Take for example, a rulebase in which some controls have a custom property called 'template', which maps onto the path of a template to be used instead of the default template to render that particular control. Additionally, in this example, this functionality is to only be available to this one rulebase, perhaps because access to this rulebase is only granted to a small group of trusted rulebase engineers. Call this rulebase 'alternative'.

The following is a fragment of the default template for cycling through the controls on a screen and calling the template for each control type to render the specific control:

```
#set( $controlList = ${screen.getControls()} ) #foreach( $control in $controlList ) <div class="control-item"> #parse( "controls/${control.getControlType()}.vm" ) </div> <span class="control-clear"></span> #end
```

The name of the template used to render a particular control is retrieved through the **getControlType** method on the control object. This is the logic that will be altered for this example:

```
#set( $rulebase = ${screen.getInterviewSession().getRulebase().getName()} ) #set( $controlList = ${screen.getControls()} ) #foreach( $control in $controlList ) <div class="control-item"> #if( $rulebase == "alternative" && $control.hasProperty("template") ) #set( $alternativeTemplate = ${control.getProperty("template")} ) #parse( "${alternativeTemplate}" ) #else #parse( "controls/${control.getControlType()}.vm" ) #end </div> <span class="control-clear"></span> #end
```

Other methods

Templates can be modified to check for a specific rulebase or set of rulebases, and to alter the functionality and/or look and feel of the application for that particular session. This is what we have done above using the specific case of custom properties attached to controls. Another option would be to parse an alternative/extra CSS template for particular rulebase(s) and not for others. There are, however, a couple of things to bear in mind about rulebase-specific configuration:

1. Be careful about where you insert the rulebase check in the template code; think about which templates may make a `#parse` call into the current template, and about whether there is a chance that the rulebase-specific code may be executed for all rulebases.
2. Keep in mind template and property file caching settings.

The following describes rulebase management in terms of deploying Oracle Determinations Server for Apache Tomcat V6.x or WebSphere Application Server V6.1.

You can only manage individual rulebases for a determinations-server web application on application servers that deploy web applications in expanded form. Apache Tomcat and IBM WebSphere Application Server do deploy their web applications in expanded form, while Oracle WebLogic does not deploy a web application in expanded form by default.

In order to add, update, or remove a rulebase you need to locate the rulebases directory of the determinations-server web application. By default, this is located at: `WEBROOT/WEB-INF/classes/rulebases/`. See below for the WEBROOT depending on the different application server.

If you have specified a custom rulebases directory, you can add, update or remove a rulebase at the location of the specified rulebases directory.

To add a rulebase

Using Oracle Policy Modeling, open and build the rulebase. The rulebase archive, `<rulebase name>.zip` will be built in the output directory of the rulebase project.

Copy the rulebase archive to Oracle Determinations Server's rulebase directory. By default, Oracle Determinations Server will automatically add the rulebase.

Note: In addition to building a rulebase by using Oracle Policy Modeling, it can be done from the command line; for information on how to do this, see the topic *Build the rulebase from the command line*, in the *Reference* section of the *Oracle Policy Modeling User's Guide*.

To update a rulebase

Using Oracle Policy Modeling, open and build the rulebase. The rulebase archive, *<rulebase name>.zip* will be built in the output directory of the Rulebase project.

Copy the rulebase archive to Oracle Determinations Server's rulebase directory. By default, Oracle Determinations Server will automatically update the rulebase.

To remove a rulebase

Delete the rulebase archive from Oracle Determinations Server's rulebase directory. By default, Oracle Determinations Server will automatically remove the rulebase, and its operations will no longer be available

The following terminology applies:

Term	Definition
WAS	Refers to the directory to which the WebSphere Application Server has been installed.
CATALINA_HOME	The Apache Tomcat install directory.
WEBROOT	The path to the determinations-server root directory on the deployed server. By default this is: <ul style="list-style-type: none">• Apache Tomcat (all versions): <i>CATALINA_HOME/WebApps/determinations-server.</i>• WebSphere Application Server 6.1: <i>WAS/profiles/<profile name>/InstalledApps/<node>/determinations-server.war</i> Where:<ul style="list-style-type: none">◦ <i><profile name></i> is the name of the application server instance profile determinations-server is installed to (default is AppSvr01).◦ <i><node></i> the node that HDS is installed to (default is <i><hostname>Node01Cell</i>).

Customize the inferencing cycle with custom functions and inferencing listeners

A rulebase may need to perform a calculation that cannot be done (or cannot be done easily) using the built-in functions of the Oracle Determinations Engine. To work around this, a Java or .NET developer can use custom functions or inferencing listeners to perform calculations that cannot be done in rules.

In general this should be a last resort when rules cannot achieve the desired result, as using custom code can impact the following:

- Transparency – rules written in code are not easily understood by a business user or non-technical rule developer.
- Decision reports – custom inferred values are not fully explored in a decision report as the Determinations Engine does not know what values were relevant to the result.
- Cross-platform compatibility – the Determinations Engine evaluates rules identically on both Java and .NET platforms, but custom code written in one environment cannot be run in the other without a bridging solution of some sort.
- Performance – the Determinations Engine is designed to evaluate rules in the most efficient order and it cannot consider custom code in this analysis.

If custom code is required, it can either be in the form of a custom function or an inferencing listener. The main difference between a custom function approach and an inferencing listener approach is that an inferencing listener is able to 'infer' anything in the session (by modifying the session directly), whereas a custom function can only calculate individual values and is called by the engine when needed.

Custom Function

A custom function is like any other standard function in Oracle Policy Automation, but the behavior is defined with a custom Java or .NET class. Use a custom function when:

- You have a single attribute value to be calculated (custom functions can only return a single values, and they cannot modify the session in any other way).
- The calculation is fairly cheap. It is possible for a custom function to be called during a decision report so it should be a computationally inexpensive calculation.
- The result needs to be included in a decision report. The calculations performed by the custom function will not appear, but if the call to the custom function is guarded by a condition then the condition is included in the decision report.

Inferencing Listeners

An inferencing listener is also defined by a custom Java or .NET class and gets called before, during (when triggered by inferencing events) and after the inference cycle. When used to customize the inferencing, the class should minimally implement the **endInferencing** method to set or update any custom inferred values.

After an inferencing cycle, if an inferencing listener has made changes to the rulebase session then another inferencing cycle is performed.

Use an inferencing listener to customize the inferencing cycle when:

- You need to infer something that isn't a single value, such as creating entity instances or a calculation with multiple results.
- You don't need the results to be explained in a decision report (values set by an inferencing listener appear as base values in the decision report).

See also:

[Rulebase configuration file](#)

Initialize a new session with a Rulebase Listener

Rulebase Listeners are custom objects that are created when a rulebase is loaded, and called every time a new session is created. Developers can create a Rulebase Listener to perform early initialization of new sessions before they are returned to the calling application. For example a Rulebase Listener can:

- Install a formatter, inferencing listener or a legacy custom function handler.
- Preload data into a new session.

For more information, see the following topics:

[Rulebase Listeners](#)

[Example: Create a Rulebase Listener to preload reference data](#)

Create a custom interview user experience

What do you want to do?

Create the interview session

Add data to the interview session

Retrieve Interview Engine screens

Investigate an Interview Engine goal

Integrate Interview Engine with an external data source

Generate an Interview Engine decision document

Retrieve Interview Engine commentary

Handle Interview Engine events

Install an Interview Engine plug-in

Before setting out to create an interview session, you should familiarize yourself with the Interview Engine by reading the topic. [Understand the Interview Engine.](#)

Create the interview session

The first step when working with the Interview Engine is to create an interview session; this is done as follows:

1. Create and configure an **EngineConfiguration** object.
2. Create an instance of the Interview Engine.
3. Get the rulebase to be used.
4. Create the interview session.

This is illustrated by the following code sample:

Code sample

```
import com.oracle.determinations.interview.engine.*;

/**
 * Code sample illustrating how to create an Interview Session
 */
public class SampleSessionCreator {
    //the path to directory containing the rulebases
    private static final String RULEBASE_PATH = "rulebases";
    //The id of the rulebase we want to create a session for
    private static final String RULEBASE_ID = "SuperSimple";
    //The locale that we want to use for this session
    private static final String LOCALE = "en-GB";

    /**
```

```

* Creates and returns an InterviewSession for the SuperSimple rulebase
* @return the InterviewSession
*/
public InterviewSession createSession () {
    /**
     * Step 1. Create the Configuration object
     */
    EngineConfiguration configuration = new EngineConfiguration();
    //set the path to load the rulebases from
    configuration.setRulebaseDirectory(RULEBASE_PATH);
    //signifies that we want to load rulebases from the file system and use hotswapping mode
    configuration.setLoadRulebaseFromClasspath(false);
    configuration.setCacheLoadedRulebases(false);

    /**
     * Step 2. Create an InterviewEngine
     */
    InterviewEngine engine = InterviewEngineFactory.createInstance(configuration);
    SecurityToken token = engine.getSecurityService().authenticateUser("user");

    /**
     * Step 3. Get the rulebase
     */
    InterviewRulebase superSimpleRulebase = engine.getRulebaseService().getRulebase(token,
RULEBASE_ID);

    /**
     * Step 4. Create the Interview Session
     */
    InterviewSession session = engine.createSession(superSimpleRulebase, LOCALE, token);

    return session;
}
}

```

Add data to the interview session

There are two ways of adding/modifying data for an interview session:

[Submit an interview screen](#)

[Add data directly to the session](#)

Submit an interview screen

The most common way of adding data to an interview session is to do so by populating an interview screen and submitting it using the following method:

```
TransactionResult InterviewSession.submit(InterivewScreen screen)
```

Calls to this method trigger a transaction which attempts to submit the data contained on the screen in the rule session. The **TransactionResult** that is returned provides an indication as to whether the transaction succeeded or failed and a list of any errors, warnings and rule events that fired during the transaction.

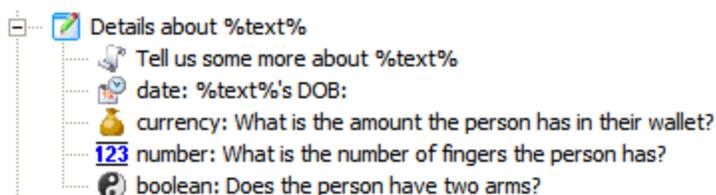
Set input control values

InterviewScreens can contain instances of **InterivewInputControl** which are bound to a particular attribute. Adding values to the screen is simply a matter of calling the following on the **InputInterviewControl** and supplying the value corresponding to that controls attribute:

```
void setValue(Object v);
```

on the **InputInterviewControl** and supplying the value corresponding to that controls attribute.

The sample code that follows, illustrates how to add values to the following screen:



Code sample

```
import com.oracle.determinations.interview.engine.screen.InterviewScreen;
import com.oracle.determinations.interview.engine.screen.InputInterviewControl;
import com.oracle.determinations.interview.engine.InterviewSession;
import com.oracle.determinations.interview.engine.data.TransactionResult;
import com.oracle.determinations.engine.Attribute;

import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.Date;
import java.util.HashMap;

public class ScreenDataExample {

    public void InputValueExample(InterviewScreen screen) {
        Calendar calendar = new GregorianCalendar();
        calendar.set(1999, 5, 1, 0, 0, 0);
        calendar.clear(Calendar.MILLISECOND);
```

```

/*****
 * Input values
 *****/
Date dob = calendar.getTime();
Double money = new Double(150.20);
Double fingers = new Double(10.0);
Boolean hasTwoArms = Boolean.TRUE;

//map values against the ID's of the attributes to which they belong
HashMap<String, Object> values = new HashMap<String, Object>(4);
values.put("date", dob);
values.put("currency", money);
values.put("number", fingers);
values.put("boolean", hasTwoArms);

//Iterate through the controls on the screen and set the appropriate values
for(Object oCtrl : screen.getControls()){
    if(oCtrl instanceof InputInterviewControl){
        InputInterviewControl interviewCtrl = (InputInterviewControl)oCtrl;
        Attribute controlAttribute = interviewCtrl.getAttribute();
        interviewCtrl.setValue(values.get(controlAttribute.getName()));
    }
}
//submit the screen
InterviewSession session = screen.getInteractiveSession();
TransactionResult result = session.submit(screen);

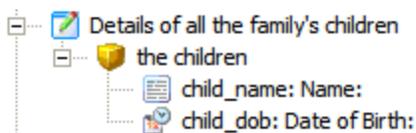
//process result
}
}

```

Add entities

The **ContainmentRelationshipInterviewControl** is the control that represents the creation of new entity instances during an interview. Entities, along with their relationships, can be added, edited and removed using the methods provided by this control. When created, this control will automatically be populated with the relevant entity instances currently held in the session. Each instance of the entity represented on the screen is encapsulated by **EntityInstanceInterviewControl** which, in turn, contains the list of controls bound to that entity instance.

The sample code that follows, illustrates how to add, delete and edit existing entity information for the following screen:



Code sample

```
public void EntityCollectExample(InterviewScreen screen) {
    //Get the Containment Collect Control which is the first control on this screen
    ContainmentRelationshipInterviewControl entityCollect = (ContainmentRelationshipInterviewControl) screen.getControls().get(0);

    //Add an instance of the child 'Fred'
    EntityInstanceInterviewControl fredInstance = entityCollect.addNewInstance("Fred");
    //add values for Fred's name and date of birth which is 25 December 1990
    InputInterviewControl fredNameCtrl = (InputInterviewControl) fredInstance.getControls().get(0);
    fredNameCtrl.setValue("Fred");
    InputInterviewControl fredDobCtrl = (InputInterviewControl) fredInstance.getControls().get(1);
    Calendar calendar = new GregorianCalendar();
    calendar.set(1990, 12, 25, 0, 0, 0);
    calendar.clear(Calendar.MILLISECOND);
    fredDobCtrl.setValue(calendar.getTime());

    //Get the existing instance 'Liz' and correct her name to be 'Elizabeth'
    EntityInstanceInterviewControl lizInstance = entityCollect.getInstance("Liz");
    InputInterviewControl lizNameCtrl = (InputInterviewControl) lizInstance.getControls().get(0);
    lizNameCtrl.setValue("Elizabeth");

    //finally remove the instance of the child walter
    entityCollect.deleteInstance("Walter");

    //submit the screen
    //submit the screen
    InterviewSession session = screen.getInteractiveSession();
    TransactionResult result = session.submit(screen);

    //process result
}
```

This sample illustrates how look up entity instances in the underlying rule session:

```
/**
 * Creates and Delete a set of entity instances on an entity collect screen
 *
 * @param screen - the entity collect screen to be submitted
 * @param instancesToAdd - a list of strings representing the list of instance ids of the entity instance to
add
 * @param instancesToDelete - a list of strings representing the list of instances ids of the entity instances
to delete
 */
public void workingWithEntityCollectSample(InterviewScreen screen, List<String> instancesToAdd,
```

```

List<String>
    instancesToDelete) {
        //Get the Containment Collect Control which is the first control on this screen
        ContainmentRelationshipInterviewControl entityCollectControl = (Con-
tainmentRelationshipInterviewControl)
            screen.getControls().get(0);

        //Navigating between the interview data model's entity instance and that of the underlying rule ses-
sion is done via the means of
        //an EntityInstanceIdentifier, so cache a list of these for each instance we create so that we can refer
back to them later
        List<InterviewEntityInstanceIdentifier> newInstanceIds = new ArrayList(instancesToAdd.size());
        for (String instanceId : instancesToAdd) {
            EntityInstanceInterviewControl newInstanceCtrl = entityCollectControl.addNewInstance
(instanceId);
            newInstanceIds.add(newInstanceCtrl.getContext());

            //set some values for this control...
        }

        //Delete entity instances
        for (String instanceId : instancesToDelete)
            entityCollectControl.deleteInstance(instanceId);

        //submit the screen
        InterviewSession session = screen.getInteractiveSession();
        TransactionResult result = session.submit(screen);

        if(result.isSuccess()){
            //the transaction succeeded so we can now iterate over our newly created entity instances
            for (InterviewEntityInstanceIdentifier identifier : newInstanceIds){
                EntityInstance entityInstance = identifier.findEntityInstance(session);
                if(entityInstance == null)
                    throw new InterviewEngineException("could not find newly created instance in the ses-
sion");

                //do something with the entity instance
            }

            //we can even check to make sure that or deleted instances were actually deleted.
            for (String instanceId : instancesToDelete){
                InterviewEntityInstanceIdentifier identifier = newInterviewEntityInstanceIdentifier
(entityCollectControl.
                    getRelationship().getTargetEntity().getName(), instanceId);

```

```

        EntityInstance entityInstance = identifier.findEntityInstance(session);
        if(entityInstance != null)
            throw new InterviewEngineException("This instance was supposed to be deleted!")
    }
}
}

```

Add reference relationships

The **ReferenceRelationshipInterviewControl** represents a control that allows entity instances to be related together via a specified relationship. Unlike the **ContainmentRelationshipInterviewControl** it does not allow new instances of the relationship's target to be created. When created, the **ReferenceRelationshipInterviewControl** will be populated with the list of all possible targets of relationship, along with all relevant relationships already held in the session. Relationship instances can be added or removed using the methods supplied by this control.

Add data directly to the session

The second way of modifying the data held in the interview session is via the following method:

```
TransactionResult InterviewSession.submit(InterviewUserData dataModel)
```

This method allows for data currently held in the system to be modified independently of a given screen model. Like the [previous method](#) the data contained in the **InterviewUserData** provided to this method is considered a single, atomic transaction.

Understand the InterviewUserData

The **InterviewUserData** is the Interview Engine's abstracted representation of the data currently held in the session. In the context of modifying data within the system, the user data model represents the delta of the changes that are to be performed in the transaction. In addition to the **InterviewUserData** object the Interview Engine's data model contains the following key classes:

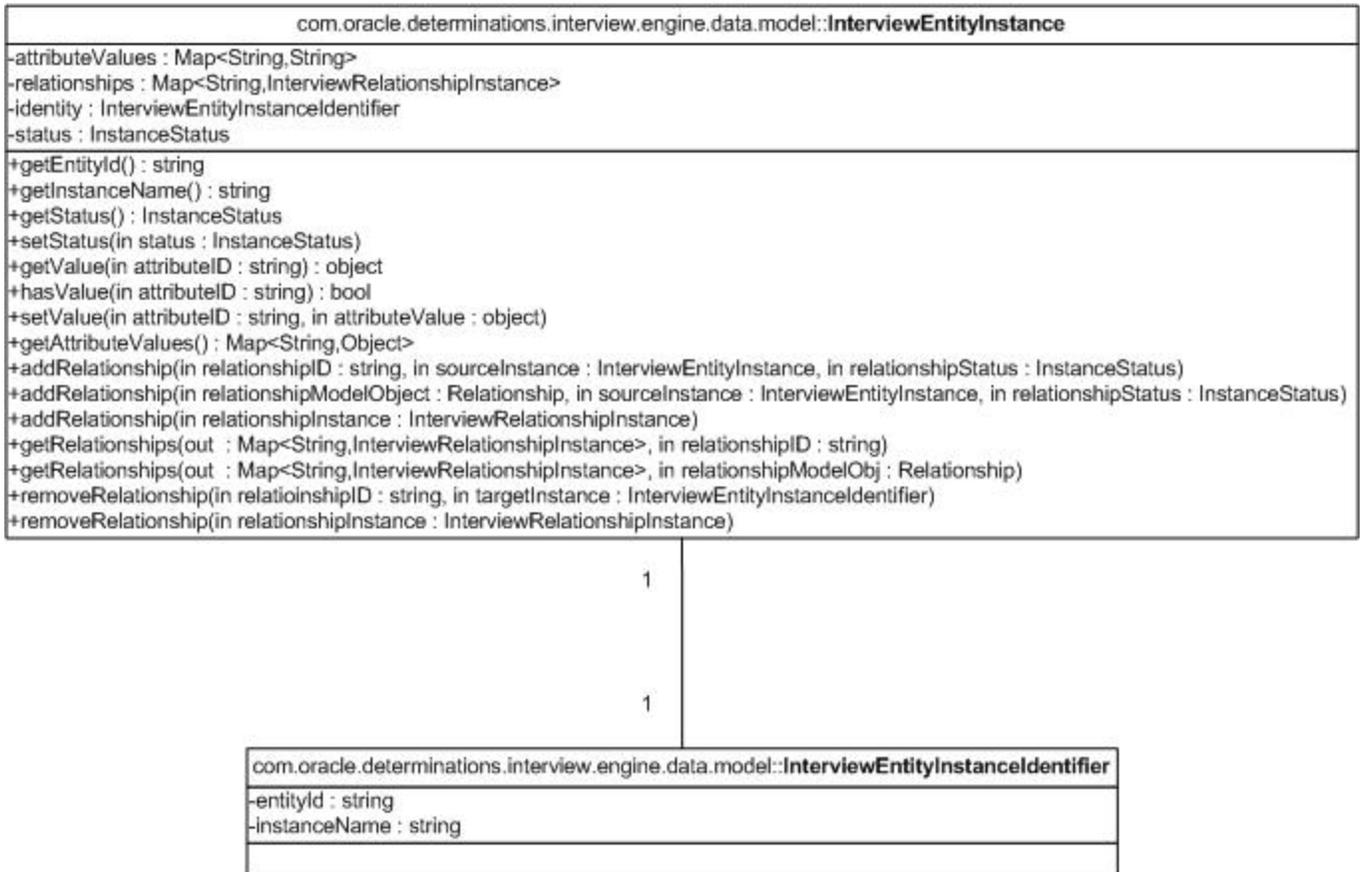
- **InterviewEntityInstance** - this is the main class representing an Entity instance. It contains the list of attribute values to set during the transaction, along with all the relationship instances affected by the transaction; for more information see *Understand the InterviewEntityInstance* below.
- **InterviewEntityInstanceIdentifier** - represents the mapping between the **InterviewEntityInstance** and equivalent Determination's **EntityInstance**.
- **InterviewRelationshipInstance** - represents an instance of a relationship between a given source and target **InterviewEntityInstance**.
- **InstanceStatus** - represents the status of **InterviewEntityInstance** and **InterviewRelationshipInstance** in the transaction. There are three possible statuses:
 - **EXISTING** - signifies that the given instance must already be contained in the session.
 - **ADD** - signifies that the given instance must be added to the session.
 - **DELETE** - signifies that the given instance must be removed from the session.

Understand the InterviewEntityInstance

As demonstrated in [Construct the InterviewUserData](#) below, the **InterviewEntityInstance** object is the key building block when constructing the **InterviewUserData**, and all modification can be done through the **InterviewEntityInstance**.

Below is a class diagram of the **InterviewEntityInstance** with key class members. Note the following points when analyzing it:

- Each **InterviewEntityInstance** is identified by its, **InterviewEntityInstanceIdentifier** which encapsulates the entity ID (for example, "child", "school") and **instanceName** (this is unique to this particular instance when compared to other instances of the same entity ID)
- Attribute values are simply stored as a key-value inside a Map object (key is Attribute ID, and Object value is the value of the Attribute).
- When setting the Object value of an Attribute, the Object type is dependent on the Attribute's rulebase type and the equivalent Java/.NET object type. For example, if the attribute is a rule Number type, its equivalent Java object is the Double. See the topic [Formatter Plugin](#) for ruleengine datatype to Java/.NET mapping.
- The status of the **InterviewEntityInstance** is of the type **InstanceStatus** - which is an enumeration. InstanceStatus is really of the type 'byte', but when getting or setting the status, the **InstanceStatus** enumeration values should be used (**InstanceStatus.ADD**, **InstanceStatus.EXISTING**, **InstanceStatus.DELETE**)
- This object also stores the list of relationship instances for which this **InterviewEntityInstance** is the source



Construct the InterviewUserData

Conceptually, building the **InterviewUserData** is commonly constructed as per the sequence below:

1. An **InterviewUserData** is created
2. Entity instances modifications are created and added into the **InterviewUserData**
 - i. An entity instance is created
 - ii. The 'modification status' of the instance is set - add, delete, or existing
 - iii. If the instance's status is 'add' or 'existing', values of zero or more attributes of the instance is populated to define attribute modifications (no reason to modify attributes if the instance is to be deleted)
 - iv. The instance is added into the **InterviewUserData**
3. Relationship modifications between the entity instances are create, using the source entity instance
 - i. The relationship is determined
 - ii. The source and target entity instance are instantiated and accessible
 - iii. The relationship is created in the source entity instance, passing the name of the relationship, target entity instance and '**modification status**' of add or delete.

When constructing the **InterviewUserData** with the above steps, the following behaviors need to be kept in mind:

- Deleting an entity instance automatically deletes all relationship instances associated with that instance. Therefore there is no need to manually delete those relationships
- When creating the relationship, care must be taken to ensure the source and target entity instances are the correct 'entity' for the relationship. That is, the relationship expects a certain entity for its source and target, and the entity instances must abide to that.
- The 'existing' modification status is for
 - verifying that the instance exists in the Session
 - making it easier to create relationships because the entity instances are instantiated and accessible
 - enabling modification of attribute values of a target existing instance (that is, attributes to modify can only be specified in InterviewUserData by placing it inside an Entity instance and then adding that instance to **InterviewUserData**)

The following sample code demonstrates the steps above. Note that for demonstration's sake, the code is simplified and does not incorporate best practices. See [Modifying the instance data - sample code](#) for best practice techniques for constructing the **InterviewUserData**.

Code sample

Actual code representation of concept

```
//Rulebase Model:
//- Entity 'child' with attributes name and age
//- Entity 'school' with attributes name and address
//- Relationship 'childsschool' that connects 'child' entity to target 'school' entity

//1.
InterviewUserData iuData = new InterviewUserData();
```

```
//2.  
InterviewEntityInstance entInst1 = new InterviewEntityInstance("child", "1"); //2a  
entInst1.setStatus(InstanceStatus.ADD); //2b  
entInst1.setValue("name", "Joe");//2c  
entInst1.setValue("age", "10");//2c  
iuData.addInstance(entInst1);//2d
```

```
InterviewEntityInstance entInst2 = new InterviewEntityInstance("child", "2"); //2a  
entInst2.setStatus(InstanceStatus.ADD); //2b  
entInst2.setValue("name", "Bill");//2c  
entInst2.setValue("age", "12");//2c  
iuData.addInstance(entInst2);//2d
```

```
InterviewEntityInstance entInst3 = new InterviewEntityInstance("school", "1"); //2a  
entInst3.setStatus(InstanceStatus.ADD); //2b  
entInst3.setValue("name", "Summer Heights High");//2c  
entInst3.setValue("address", "6 Summer Heights St, Summer Heights");//2c  
iuData.addInstance(entInst3);//2d
```

```
InterviewEntityInstance entInst4 = new InterviewEntityInstance("school", "2"); //2a  
entInst4.setStatus(InstanceStatus.ADD); //2b  
entInst4.setValue("name", "Old Summer Heights High");//2c  
entInst4.setValue("address", "10 Summer Heights St, Summer Heights");//2c  
iuData.addInstance(entInst4);//2d
```

```
//3.  
entInst1.addRelationship("childsschool", entInst3, InstanceStatus.ADD);//3c
```

```
entInst2.addRelationship("childsschool", entInst4, InstanceStatus.ADD);//3c
```

//The relationship below would be illegal as childsschool relationship expects a 'child' source entity instance and a 'school' target entity instance. This is a fundamental Rulebase authoring concept

```
//entInst1.addRelationship("childsschool", entInst2, InstanceStatus.ADD);
```

Considering the session data now has the new instance data above after adding the entities and relationships, the following sample code demonstrates how to modify existing entity instances, its attributes, and relationships:

Modifying existing entity instances and relationships

```
//Changing Bill's school  
InterviewUserData iuData = new InterviewUserData();
```

```
//no attribute value modifications to the instance below  
InterviewEntityInstance entInst2 = new InterviewEntityInstance("child", "2"); //2a  
entInst2.setStatus(InstanceStatus.EXISTING); //2b  
iuData.addInstance(entInst2);//2d
```

```

//attribute value modification to the instance below
InterviewEntityInstance entInst3 = new InterviewEntityInstance("school", "1"); //2a
entInst3.setStatus(InstanceStatus.EXISTING); //2b
entInst3.setValue("name", "New Summer Heights High");//2c
iuData.addInstance(entInst3);//2d

InterviewEntityInstance entInst4 = new InterviewEntityInstance("school", "2"); //2a
entInst4.setStatus(InstanceStatus.EXISTING); //2b
iuData.addInstance(entInst4);//2d

//The source and target instances have been defined,which makes it much easier to add and
delete relationships
entInst2.addRelationship("childsschool", entInst4, InstanceStatus.DELETE);//Delete old rela-
tionship to old school
entInst2.addRelationship("childsschool", entInst3, InstanceStatus.ADD); //Create new rela-
tionship to new school

//Deleting Joe the child
InterviewEntityInstance entInst1 = new InterviewEntityInstance("child", "1"); //2a
entInst1.setStatus(InstanceStatus.DELETE); //2b
iuData.addInstance(entInst1);//2d

```

Technical information when constructing InterviewUserData

The following are key points to keep in mind when constructing the **InterviewUserData**:

- When defining an entity instance to be retrieved (**InstanceStatus.EXISTING** or **InstanceStatuts.DELETE**), ensure that the identifiers **entityID** and **instanceName** are valid.
- Setting an entity instance with 'existing' status is a good way to check and ensure that an **InterviewEntityInstance** exists in the session, before trying to modify values, or add/delete relationships to that particular instance.
- When defining an attribute to modify:
 - The target entity instance that contains the attribute must be identified (that means a valid **entityID** and **instanceName**).
 - The attribute ID of the attribute to modify must be valid; that is, the attribute belongs to the entity type of the target entity instance.
- When defining a relationship between two instances to be modified:
 - The source and target entity instances must be identified (that means valid **entityID** and **instanceName**).
 - The relationship ID provided must be valid. That is, the relationship ID must exist. (this is not a problem if using the rulebase model 'Relationship' object).
 - The relationship's expected source and target entity type must match the entity type of the source and target entity instances.


```

//Get the underlying Rulebase object
Rulebase detEngineRulebase = rulebase.getRulebase();

//Check that an Entity ID 'child' exists first before creating the InterviewEntityInstance
Entity checkEntity = rulebase.getEntity("child");
if(checkEntity != null)
{
    //continue ceating InterviewEntitInstance
    InterviewEntityInstance entityInstance = new InterviewEntityInstance("child", "1");
    entityInstance.setStatus(InstanceStatus.ADD);
    ...
}
else
{
    //the entity 'child' doesn't exist, do something to fix it or throw an error
}

```

Verify attribute value modifications

When adding attribute values to an **InterviewEntityInstance**, check that the attribute exists for the entity instance.

Code sample

Verify attribute value modifications - sample code

```

//DataAdaptor load()
public InterviewUserData load(SecurityToken token, String caseID,
                             InterviewRulebase rulebase)

//Get the underlying Rulebase object
Rulebase detEngineRulebase = rulebase.getRulebase();

//Check that an Entity ID 'child' exists first before creating the InterviewEntityInstance
Entity checkEntity = rulebase.getEntity("child");
if(checkEntity != null)
{
    //continue ceating InterviewEntitInstance
    InterviewEntityInstance entityInstance = new InterviewEntityInstance("child", "1");
    entityInstance.setStatus(InstanceStatus.EXISTING); //Checking that this already exists, and also the entity
instance needs to be instantiated so attribute values can be added to it

    //Attempting to add attribute values to 'entityInstance'
    Attribute checkAttribute = checkEntity.getAttribute("name");
    if(checkAttribute != null)
    {
        entityInstance.setValue("name", "Joe");
    }
    else

```

```

    {
        //the attribute 'name' does not exist in the entity 'child', do something to fix it or throw an error
    }
}
else
{
    //the entity 'child' doesn't exist, do something to fix it or throw an error
}

```

Adding the above attribute value was simple, because the attribute 'name' is a Text ruleengine datatype, and the corresponding Java/.NET type is String (see [Formatter Plugin](#)).

In the following sample code, we check the ruleengine datatype (**HaleyType**) of the attribute, and parse it accordingly from **String format -> Object**. A basic parser method (**simpleParse**) has been created which converts the String value to the correct Object value using the **HaleyType** of the attribute.

Code sample

Verify attribute value modifications - sample code

```

public InterviewUserData load(SecurityToken token, String caseID,
                             InterviewRulebase rulebase)
{

    //Get the underlying Rulebase object
    Rulebase detEngineRulebase = rulebase.getRulebase();

    //Check that an Entity ID 'child' exists first before creating the InterviewEntityInstance
    Entity checkEntity = rulebase.getEntity("child");
    if(checkEntity != null)
    {
        //continue ceating InterviewEntitInstance
        InterviewEntityInstance entityInstance = new InterviewEntityInstance("child", "1");
        entityInstance.setStatus(InstanceStatus.EXISTING); //Checking that this already exists, and also the entity
        instance needs to be instantiated so attribute values can be added to it

        //Attempting to add attribute values to 'entityInstance'
        Attribute checkAttribute = checkEntity.getAttribute("name");
        if(checkAttribute != null)
        {
            entityInstance.setValue("dateofbirth", simpleParse("10/10/1990", checkAttribute.getValueType()));
        }
        else
        {
            //the attribute 'name' does not exist in the entity 'child', do something to fix it or throw an error
        }
    }
}
else
{

```

```

    //the entity 'child' doesn't exist, do something to fix it or throw an error
}
...
} //end function

//Really simple parser to convert a String to the correct Object value, does not handle Time of Day
(Haley.TIMEOFDAY) types
public Object simpleParse(String value, byte haleyType)
{
    if(haleyType == HaleyType.BOOLEAN)
    {
        if (value.toLowerCase().equals("true") || value.toLowerCase().equals("yes")) {
            return Boolean.TRUE;
        } else if (value.toLowerCase().equals("false") || value.toLowerCase().equals("no")) {
            return Boolean.FALSE;
        }
    }
    else if(haleyType == HaleyType.TEXT)
    {
        return value;
    }
    else if(haleyType == HaleyType.NUMBER || haleyType == HaleyType.CURRENCY)
    {
        return Double.valueOf(value);
    }
    else if(haleyType == HaleyType.DATE || haleyType == HaleyType.DATETIME)
    {
        return Date.valueOf(value);
    }
    return null;
}

```

Verify relationship modifications

When adding relationship modifications, check that the entity of the source and target entity instances match the expected entity source/target of the relationship.

Code sample

Verify relationship modifications - sample code

```

//DataAdaptor load()
public InterviewUserData load(SecurityToken token, String caseID,
                             InterviewRulebase rulebase)

//Get the underlying Rulebase object
Rulebase detEngineRulebase = rulebase.getRulebase();

Entity childEntity = rulebase.getEntity("child");

```

```

Entity schoolEntity = rulebase.getEntity("school");

if(childEntity != null && schoolEntity != null)
{
    //continue creating InterviewEntitInstance
    InterviewEntityInstance childEntityInstance = new InterviewEntityInstance("child", "1");
    childEntityInstance.setStatus(InstanceStatus.EXISTING); //Checking that this already exists, and also the
    entity instance needs to be instantiated so relationship modifications can be added to it

    InterviewEntityInstance schoolEntityInstance = new InterviewEntityInstance("school", "1");
    schoolEntityInstance.setStatus(InstanceStatus.EXISTING); //Checking that this already exists, and also the
    entity instance needs to be instantiated so relationship modifications can be added to it

    //Attempting to add a new relationship between child and school
    Relationship relationship = childEntity.getRelationship("childsschool");
    String sourceEntityName = childEntity.getName();
    String targetEntityName = schoolEntity.getName();

    //Check
    if(relationship.getSourceEntity().getName().equals(sourceEntityName) &&
        relationship.getTargetEntity().getName().equals(targetEntityName)
        )
    {
        childEntityInstance.addRelationship("childsschool", schoolEntityInstance, InstanceStatus.ADD);
        //Or could have been childEntityInstance.addRelationship(relationship, schoolEntityInstance,
InstanceStatus.ADD);
    }
    else
    {
        //the entity of thesource and/or target instance is invalid for the relationship "childsschool"
    }
}
else
{
    //the entity 'child' or 'school' doesn't exist, do something to fix it or throw an error
}
}

```

Best practice sample code

The following code samples show a conversion of the simple sample code found in [Constructing InterviewUserData](#) samples above, to best-practice.

Code sample

Modifying the instance data - sample code - add some data

```

//Rulebase Model:
//- Entity 'child' with attributes name and age

```

```

//- Entity 'school' with attributes name and address
//- Relationship 'childschool' that connects 'child' entity to target 'school' entity

InterviewUserData iuData = new InterviewUserData();
Rulebase rulebase = interviewSession.getRulebase().getRulebase();

//Add child entity with name=Joe and age=10
Entity checkEntity = rulebase.getEntity("child");
if(checkEntity != null)
{
    //continue creating InterviewEntitInstance
    InterviewEntityInstance entInst1 = new InterviewEntityInstance("child", "1");
    entInst1.setStatus(InstanceStatus.ADD);
    iuData.addInstance(entInst1);

    //Attempting to add attribute values
    Attribute checkAttribute = checkEntity .getAttribute("name");
    if(checkAttribute != null)
    {
        entInst1.setValue("name", simpleParse("Joe", checkAttribute.getValueType()));
    }
    else
    {
        //the attribute 'name' does not exist in the entity 'child'
    }
    checkAttribute = checkEntity .getAttribute("age");
    if(checkAttribute != null)
    {
        entInst1.setValue("age", simpleParse("10", checkAttribute.getValueType()));
    }
    else
    {
        //the attribute 'name' does not exist in the entity 'child'
    }
}
else
{
    //the entity 'child' doesn't exist, do something to fix it or throw an error
}

//Add a school instance with name=Summer Heights High, address=1 Summer St, Summer
Heights
checkEntity = rulebase.getEntity("school");
if(checkEntity != null)
{

```

```

//continue creating InterviewEntitInstance
InterviewEntityInstance entInst2 = new InterviewEntityInstance("school", "1");
entInst2.setStatus(InstanceStatus.ADD);
iuData.addInstance(entInst2);

//Attempting to add attribute values
Attribute checkAttribute = checkEntity .getAttribute("name");
if(checkAttribute != null)
{
    entInst2.setValue("name", simpleParse("Summer Heights High", check-
Attribute.getValueType()));
}
else
{
    //the attribute 'name' does not exist in the entity 'school'
}
checkAttribute = checkEntity .getAttribute("address");
if(checkAttribute != null)
{
    entInst2.setValue("address", simpleParse("1 Summer St, Summer Heights", check-
Attribute.getValueType()));
}
else
{
    //the attribute 'name' does not exist in the entity 'school'
}
}
else
{
    //the entity 'school' doesn't exist, do something to fix it or throw an error
}

//Add a relationship between Joe and Summer Heights High
Relationship relationship = entInst1.getRelationship("childsschool");
String sourceEntityName = entInst1.getName();
String targetEntityName = entInst2.getName();

//Check
if(relationship.getSourceEntity().getName().equals(sourceEntityName) &&
    relationship.getTargetEntity().getName().equals(targetEntityName)
)
{
    entInst1.addRelationship("childsschool", entInst2, InstanceStatus.ADD);
}
else

```

```

    {
        //the entity of thesource and/or target instance is invalid for the relationship "childsschool"
    }

...
...

//Really simple parser to convert a String to the correct Object value, does not handle Time of
Day (Haley.TIMEOFDAY) types
public Object simpleParse(String value, byte haleyType)
{
    if(haleyType == HaleyType.BOOLEAN)
    {
        if (value.toLowerCase().equals("true") || value.toLowerCase().equals("yes"))
        {
            return Boolean.TRUE;
        } else if (value.toLowerCase().equals("false") || value.toLowerCase().equals("no")) {
            return Boolean.FALSE;
        }
    }
    else if(haleyType == HaleyType.TEXT)
    {
        return value;
    }
    else if(haleyType == HaleyType.NUMBER || haleyType == HaleyType.CURRENCY)
    {
        return Double.valueOf(value);
    }
    else if(haleyType == HaleyType.DATE || haleyType == HaleyType.DATETIME)
    {
        return Date.valueOf(value);
    }
    return null;
}

```

Modifying the instance data - sample code (modify existing data from first sample code)

```

//Rulebase Model:
//- Entity 'child' with attributes name and age
//- Entity 'school' with attributes name and address
//- Relationship 'childsschool' that connects 'child' entity to target 'school' entity

InterviewUserData iuData = new InterviewUserData();
Rulebase rulebase = interviewSession.getRulebase().getRulebase();

```

```

//Check child entity '1' exists, change his age to 14
Entity checkEntity = rulebase.getEntity("child");
if(checkEntity != null)
{
    //continue creating InterviewEntitInstance
    InterviewEntityInstance entInst1 = new InterviewEntityInstance("child", "1");
    entInst1.setStatus(InstanceStatus.EXISTING);
    iuData.addInstance(entInst1);

    //Attempting to add attribute values to be modified
    Attribute checkAttribute = checkEntity .getAttribute("age");
    if(checkAttribute != null)
    {
        entInst1.setValue("age", simpleParse("14", checkAttribute.getValueType()));
    }
    else
    {
        //the attribute 'age' does not exist in the entity 'child'
    }
}
else
{
    //the entity 'child' doesn't exist, do something to fix it or throw an error
}

//Add a school instance with name=Summer Heights College, address=1 College St, Summer
Heights
checkEntity = rulebase.getEntity("school");
if(checkEntity != null)
{
    //Check that Summer Heights High still exists
    InterviewEntityInstance entInst2 = new InterviewEntityInstance("school", "1");
    entInst2.setStatus(InstanceStatus.EXISTING);
    iuData.addInstance(entInst2);

    InterviewEntityInstance entInst3 = new InterviewEntityInstance("school", "2");
    entInst3.setStatus(InstanceStatus.ADD);
    iuData.addInstance(entInst3);

    //Attempting to add attribute values
    Attribute checkAttribute = checkEntity .getAttribute("name");
    if(checkAttribute != null)
    {
        entInst3.setValue("name", simpleParse("Summer Heights College",

```

```

checkAttribute.getValueType());
    }
    else
    {
        //the attribute 'name' does not exist in the entity 'school'
    }
    checkAttribute = checkEntity .getAttribute("address");
    if(checkAttribute != null)
    {
        entInst3.setValue("address", simpleParse("10 Summer St, Summer Heights", check-
Attribute.getValueType()));
    }
    else
    {
        //the attribute 'name' does not exist in the entity 'school'
    }
}
else
{
    //the entity 'school' doesn't exist, do something to fix it or throw an error
}

//Move Joe from Summer Heights High to Summer Heights College
Relationship relationship = entInst1.getRelationship("childsschool");
String sourceEntityName = entInst1.getName();
String targetEntityName = entInst2.getName();

//Delete relationship of Joe to Summer Heights High and add relationship to Summer Heights
College
if(relationship.getSourceEntity().getName().equals(sourceEntityName) &&
    relationship.getTargetEntity().getName().equals(targetEntityName) )
{
    entInst1.addRelationship("childsschool", entInst2, InstanceStatus.DELETE);
    entInst1.addRelationship("childsschool", entInst3, InstanceStatus.ADD);
}
else
{
    //the entity of thesource and/or target instance is invalid for the relationship "childsschool"
}

...
...

//Really simple parser to convert a String to the correct Object value, does not handle Time of

```

```

Day (Haley.TIMEOFDAY) types
public Object simpleParse(String value, byte haleyType)
{
    if(haleyType == HaleyType.BOOLEAN)
    {
        if (value.toLowerCase().equals("true") || value.toLowerCase().equals("yes"))
        {
            return Boolean.TRUE;
        } else if (value.toLowerCase().equals("false") || value.toLowerCase().equals("no")) {
            return Boolean.FALSE;
        }
    }
    else if(haleyType == HaleyType.TEXT)
    {
        return value;
    }
    else if(haleyType == HaleyType.NUMBER || haleyType == HaleyType.CURRENCY)
    {
        return Double.valueOf(value);
    }
    else if(haleyType == HaleyType.DATE || haleyType == HaleyType.DATETIME)
    {
        return Date.valueOf(value);
    }
    return null;
}

```

Retrieve Interview Engine screens

Screens in the Interview Engine are managed the Screen Service which provides all the methods to retrieve each of the different screens found in an Interview.

Retrieve a question screen

The screen service provides the ability to return question screens by id, by attribute or by relationship, which is illustrated in the following example:

Code sample

```

import com.oracle.determinations.interview.engine.data.model.InterviewEntityInstanceIdentifier;
import com.oracle.determinations.interview.engine.screen.InterviewScreen;
import com.oracle.determinations.engine.Rulebase;
import com.oracle.determinations.engine.Attribute;
import com.oracle.determinations.engine.Relationship;

public class GetQuestionScreenSample {
    public void getQuestionScreens(InterviewSession session){
        /*****

```

```

    * Get Screen by ID
    *****/
    //The ID of the screen. The first parameter indicates that we want a question screen and the second
is the rulebase ID of the screen. The final two
    //parameters identify the entity instance context of the screen, in this case the instance named 'bob'
in the entity 'person'.
    String screenId = "qs$details$person$bob";
    InterviewScreen screen = session.getScreenService().getScreen(screenId);

    /*****
    * Get All Screen that collect the attribute 'person_name'
    * in the instance 'bob' of the 'person' entity
    *****/
    InterviewEntityInstanceIdentifier personBobContext = new InterviewEntityInstanceIdentifier("per-
son", "bob");
    Rulebase determinationsRulebase = session.getRulebase().getRulebase();
    Attribute personNameAttr = determinationsRulebase.getAttribute("person_name", "person");
    InterviewScreen[] personNameScreens = session.getScreenService().getAllScreens(per-
sonNameAttr, personBobContext);

    /*****
    * Get All Screen that collect the relationship 'children'
    * in the instance 'fred' of the 'person' entity
    *****/
    InterviewEntityInstanceIdentifier personFredContext = new InterviewEntityInstanceIdentifier("per-
son", "fred");
    Relationship relationshipChildren = determinationsRulebase.getRelationship("children", "person");
    InterviewScreen[] childrenScreens = session.getScreenService().getAllScreens(rela-
tionshipChildren, personFredContext);
    }
}

```

Retrieve a summary screen

Every interview has a default summary screen which can be accessed by calling:

```
ScreenService.getDefaultSummaryScreen()
```

If no default summary screen has been created, the Interview Engine will automatically generate one containing all the top level goals found in the Global Entity Instance. In the case where multiple summary screens have been authored, then non-default summary screens can be retrieved by screen Id.

Retrieve a data review screen

Like summary screens, every interview also has a default data review screen which can be accessed by calling:

```
ScreenService.getDefaultDataReviewScreen()
```

The Interview Engine will automatically generate a default data review screen containing all user set data, arranged by entity instance, if one was not explicitly authored. If multiple data review screens have been authored, the non-default screens can be accessed by Id.

Retrieve a decision report screen

Retrieving decision reports in the Interview Engine are done by asking for a decision report screen for a particular attribute or relationship. The Screen Service provides the following methods retrieve a decision report screen:

Code sample

```
/**
 * Returns a decision report screen for the specified attribute instance
 *
 * @param attr the attribute
 * @param context the entity instance context of the attribute
 * @return a screen containing the decision report for the specified attribute
 */
InterviewScreen getDecisionReportScreen(Attribute attr, InterviewEntityInstanceIdentifier context,
DecisionReportFlag flags);

/**
 * Returns a decision report screen for the specified attribute instance
 *
 * @param attr the attribute
 * @param context the entity instance context of the attribute
 * @param flags the decision report flags
 * @param ignoreInvisible set to <code>true</code> if the invisible operator should be ignored
 * @param ignoreSilent set to <code>true</code> if the silent operator should be ignored
 * @return a screen containing the decision report for the specified attribute
 */
InterviewScreen getDecisionReportScreen(Attribute attr, InterviewEntityInstanceIdentifier context,
DecisionReportFlag flags, boolean ignoreSilent, boolean ignoreInvisible);

/**
 * Returns a decision report screen for the specified relationship instance
 *
 * @param rel
 * @param context the entity instance context of the relationship's source
 * @param flags the decision report flags
 * @return a screen containing the decision report for the specified relationship
 */
InterviewScreen getDecisionReportScreen(Relationship rel, InterviewEntityInstanceIdentifier context,
DecisionReportFlag flags);

/**
 * Returns a decision report screen for the specified relationship instance
 *

```

```

* @param rel the relationship
* @param context the entity instance context of the relationship's source
* @param flags the decision report flags
* @param ignoreInvisible set to <code>true</code> if the invisible operator should be ignored
* @param ignoreSilent set to <code>true</code> if the silent operator should be ignored
* @return a screen containing the decision report for the specified attribute
*/
InterviewScreen getDecisionReportScreen(Relationship rel, InterviewEntityInstanceIdentifier context,
DecisionReportFlag flags, boolean ignoreSilent, boolean ignoreInvisible);

/**
* Returns a decision report screen for a specific goal
* @param goal the goal
* @param flags the decision report flags
* @return the decision report screen for the goal, or <code>null</code> if no such screen exists.
*/
InterviewScreen getDecisionReportScreen(InterviewGoal goal, DecisionReportFlag flags);

/**
* Returns a decision report screen for a specific goal
* @param goal the goal
* @param flags the decision report flags
* @param ignoreInvisible set to <code>true</code> if the invisible operator should be ignored
* @param ignoreSilent set to <code>true</code> if the silent operator should be ignored
* @return the decision report screen for the goal, or <code>null</code> if no such screen exists.
*/
InterviewScreen getDecisionReportScreen(InterviewGoal goal, DecisionReportFlag flags, boolean
ignoreSilent, boolean ignoreInvisible);

```

Investigate an Interview Engine goal

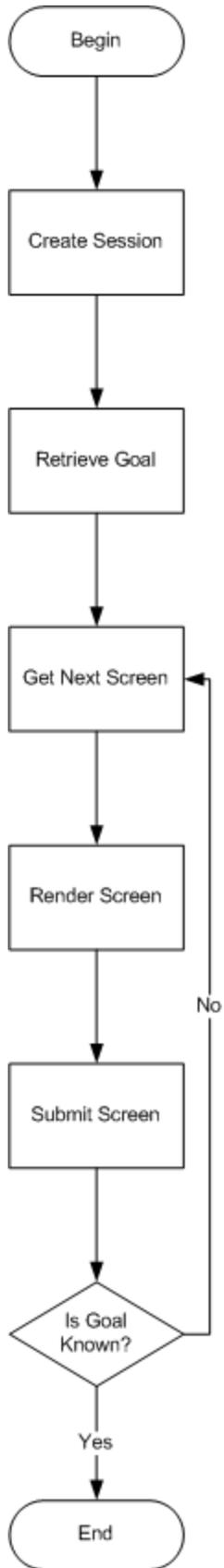
The list of goals that are able to be investigated in Interview Session are managed by the Goal Service. The Goal Service provides the ability to return a goal based on ID or goal state. The goal service can also provide a list of all top level goals in the Interview Session. A top level goal is either a top level attribute instance or a flow goal.

Goals can also be accessed via the goal control(s) on a summary screen.

Conduct the interview

Having understood the fundamentals of how the Interview Engine works, conducting an interview is a straight forward process as shown in the following flow diagram:

Interview flow diagram



Code sample

This code sample illustrates how to conduct a basic investigation for the rulebase found in the *SuperSimple.zip* sample.

```
import com.oracle.determinations.interview.engine.local.AttributeGoal;
import com.oracle.determinations.interview.engine.data.TransactionResult;
import com.oracle.determinations.interview.engine.data.model.InterviewEntityInstanceIdentifier;
import com.oracle.determinations.interview.engine.screen.*;
import com.oracle.determinations.interview.util.StringUtils;
import com.oracle.determinations.engine.Attribute;
import com.oracle.determinations.engine.DecisionReportFlag;

import java.util.*;

/**
 * Simple Example of how to drive a basic investigation
 */
public class InterviewExample {
    //the path to directory containing the rulebases
    private static final String RULEBASE_PATH = "rulebases";
    //The id of the rulebase we want to create a session for
    private static final String RULEBASE_ID = "SuperSimple";
    //The locale that we want to use for this session
    private static final String LOCALE = "en-GB";

    public static void main(String[] args) {
        /*****
         * Input values
         *****/
        String name = "Bob";
        Calendar calendar = new GregorianCalendar();
        calendar.set(1999, 5, 1, 0, 0, 0);
        calendar.clear(Calendar.MILLISECOND);

        Date dob = calendar.getTime();
        Double money = new Double(150.20);
        Double fingers = new Double(10.0);
        Boolean hasTwoArms = Boolean.TRUE;

        //map values against the ID's of the attributes to which they belong
        HashMap<String, Object> values = new HashMap<String, Object>(5);
        values.put("text", name);
        values.put("date", dob);
        values.put("currency", money);
        values.put("number", fingers);
        values.put("boolean", hasTwoArms);
    }
}
```

```

//Create the session
InterviewSession session = createSession();

//Get the default summary screen and extract the goal from it
InterviewScreen summary = session.getScreenService().getDefaultSummaryScreen();
GoalInterviewControl goalControl = (GoalInterviewControl)summary.getControls().get(0);
InterviewGoal goal = goalControl.getGoal();

//conduct the interview
while(!goal.isKnown()){
    InterviewScreen screen = session.getNextScreen(goal);
    renderScreen(screen);
    mapValues(values, screen);
    TransactionResult result = session.submit(screen);

    if(!result.isSuccess()){
        //process errors and warnings
    }
}

//Now that the interivew is complete, re-render the summary screen
System.out.println();
System.out.println("Interview complete...");
summary = session.getScreenService().getDefaultSummaryScreen();
renderScreen(summary);

//Render a decision report for the goal
System.out.println();
System.out.println("Showing Decision Report...");

if(goal instanceof AttributeGoal){
    Attribute goalAttr = ((AttributeGoal)goal).getAttribute();
    InterviewEntityInstanceIdentifier goalContext = ((AttributeGoal)goal).getContext();
    InterviewScreen decisionReportScreen = session.getScreenService().getDecisionReportScreen
(goalAttr, goalContext, DecisionReportFlag.RELEVANT);
    renderScreen(decisionReportScreen);
}

//show the data review screen
System.out.println();
System.out.println("Showing Data Review...");
renderScreen(session.getScreenService().getDefaultDataReveiwScreen());

//finally clean up the session and exit

```

```

        session.destroy();
        System.exit(0);
    }

    /**
     * Simple method to render a screen to standard out
     * @param screen the screen to render
     */
    public static void renderScreen(InterviewScreen screen) {
        System.out.println();
        System.out.println(screen.getTitle());
        System.out.println("-----");
        System.out.println();

        renderControls(screen);
    }

    /**
     * Recursive function to render a set of controls in the given control container
     * @param container the container to render
     */
    private static void renderControls(InterviewControlContainer container) {
        for (Object oCtrl : container.getControls()) {
            if (oCtrl instanceof GroupInterviewControl) {
                System.out.println(((GroupInterviewControl)oCtrl).getText());
                renderControls((InterviewControlContainer)oCtrl);
            } else if (oCtrl instanceof DataReviewScreenInterviewControl){
                DataReviewScreenInterviewControl dataReviewScreenCtrl = (DataReviewScreenIn-
terviewControl)oCtrl;
                System.out.println(dataReviewScreenCtrl.getText());
                renderControls(dataReviewScreenCtrl.getUnderlyingScreen());
            } else if (oCtrl instanceof GoalInterviewControl) {
                GoalInterviewControl goalCtrl = (GoalInterviewControl) oCtrl;
                System.out.println("Goal: " + goalCtrl.getGoal().getId() + ": " + goalCtrl.getText());
            } else if (oCtrl instanceof InputInterviewControl){
                InputInterviewControl inputCtrl = (InputInterviewControl)oCtrl;
                System.out.println("Input: " + inputCtrl.getAttribute().getName() + ": " + inputCtrl.getText()
+ ": " + StringUtils.getDefaultFormattedValue(inputCtrl.getValue()));
            } else {
                System.out.println(((InterviewControl)oCtrl).getText());
            }
        }
    }
}

/**

```

```

* Maps the input values to the screen
* @param values the input values hashed by Attribute ID
* @param screen the screen
*/
public static void mapValues(Map values, InterviewScreen screen){
    System.out.println();
    System.out.println("Mapping values...");
    //Iterate through the controls on the screen and set the appropriate values
    for (Object oCtrl : screen.getControls()) {
        if (oCtrl instanceof InputInterviewControl) {
            InputInterviewControl interviewCtrl = (InputInterviewControl) oCtrl;
            Attribute controlAttribute = interviewCtrl.getAttribute();
            interviewCtrl.setValue(values.get(controlAttribute.getName()));
            System.out.println("Setting value of attribute " + controlAttribute.getName() + ", to " +
StringUtils.getDefaultFormattedValue(values.get(controlAttribute.getName())));
        }
    }
}

/**
* Creates and returns an InterviewSession for the SuperSimple rulebase
*
* @return the InterviewSession
*/
public static InterviewSession createSession() {

    /**
    * Step 1. Create the Configuration object
    */
    EngineConfiguration configuration = new EngineConfiguration();
    //set the path to load the rulebases from
    configuration.setRulebaseDirectory(RULEBASE_PATH);
    //signifies that we want to load rulebases from the file system and use hotswapping mode
    configuration.setLoadRulebaseFromClasspath(false);
    configuration.setCacheLoadedRulebases(false);

    /**
    * Step 2. create an InterviewEngine
    */
    InterviewEngine engine = InterviewEngineFactory.createInstance(configuration);
    SecurityToken token = engine.getSecurityService().authenticateUser("user");

    /**
    * Step 3. Get the rulebase
    */

```

```

        InterviewRulebase superSimpleRulebase = engine.getRulebaseService().getRulebase(token,
RULEBASE_ID);

        /*****
        * Step 4. Create the Interview Session
        *****/
        InterviewSession session = engine.createSession(superSimpleRulebase, LOCALE, token);

        return session;
    }
}

```

This code produces the following output:

Person's name

Input: text: What is the person's name?: unknown

Mapping values...

Setting value of attribute 'text, to 'Bob

Details about Bob

Tell us some more about Bob

Input: date: Bob's DOB:: unknown

Input: currency: What is the amount Bob has in their wallet?: unknown

Input: number: What is the number of fingers Bob has?: unknown

Input: boolean: Does Bob have two arms?: unknown

Mapping values...

Setting value of attribute 'date, to '1999-06-01 00:00:00

Setting value of attribute 'currency, to '150.2

Setting value of attribute 'number, to '10

Setting value of attribute 'boolean, to 'true

Interview complete...

Super Awesome Assessment Summary

Goal: Attribute~goal~global~global: Bob is happy.

Showing Decision Report...

Bob is happy.

Bob is happy.
Bob is happy.
The person is Bob.
Bob's date of birth is 01/06/99.
The amount Bob has in their wallet is £150.20.
Bob has two arms.
The number of fingers Bob has is 10.

Showing Data Review...

Data Review

Default Screen Order
Person's name
Input: text: What is the person's name?: Bob
Details about Bob
Tell us some more about Bob
Input: date: Bob's DOB:: 1999-06-01 00:00:00
Input: currency: What is the amount Bob has in their wallet?: 150.2
Input: number: What is the number of fingers Bob has?: 10
Input: boolean: Does Bob have two arms?: true

.Net code sample

This sample (*CustomInterview.cs*) also illustrates how to manually install a default data adaptor.

```
using System;
using System.IO;
using Oracle.Determinations.Interview.Engine;
using Oracle.Determinations.Interview.Engine.Data;
using Oracle.Determinations.Interview.Engine.Data.Error;
using Oracle.Determinations.Interview.Engine.Local;
using Oracle.Determinations.Interview.Engine.Screens;
using Oracle.Determinations.Web.Platform.Plugins.Data;
using Boolean = Oracle.Determinations.Masquerade.Lang.Boolean;

namespace InterviewSample
{
    internal class Program
    {
        private const string rulebaseDir = @"..\..\rulebase";
        private const string sampleRulebase = "sample";
        private const string locale = "en-US";
        private const string dataBaseDir = "data";
```

```

private static void Main(string[] args)
{
    System.Console.WriteLine("Sample Rulebase runner");

    //we'll use most of the defaults for the engine configuration
    EngineConfiguration engineConfig = new EngineConfiguration(rulebaseDir);

    //create the engine
    InterviewEngine engine = InterviewEngineFactory.CreateInstance(engineConfig);

    //generate a default security token and create the session
    SecurityToken token = engine.GetSecurityService().AuthenticateUser("guest");

    InterviewRulebase rulebase = engine.GetRulebaseService().GetRulebase(token, sampleRule-
base);

    //To use the existing XSDDataAdaptor we can manually register it by creating one and then
adding it to the configuration
    if (!Directory.Exists(dataBaseDir))
        Directory.CreateDirectory(dataBaseDir);

    XSDDataAdaptor dataAdaptor = new XSDDataAdaptor(dataBaseDir);
    InterviewSessionConfiguration sessionConfig = new InterviewSessionConfiguration(dataAd-
aptor, null);

    InterviewSession session = engine.CreateSession(rulebase, locale, token, sessionConfig);

    //List the currently available cases
    Console.WriteLine("Currently Available Saved Cases: ");
    foreach (string name in dataAdaptor.ListCases(token, rulebase))
        Console.WriteLine(name);

    Console.WriteLine();

    //Load a case
    Console.Write("Enter the case to load [blank to skip]: ");
    string caseId = Console.ReadLine();
    if (caseId.Length > 0)
    {
        try
        {
            //this will invoke whatever data adaptor is registered on the session
            //to load the case
            session.LoadData(token, caseId);
        }
    }
}

```

```

    }
    catch (Exception ex)
    {
        Console.WriteLine("Sorry, case " + caseId + " could not be loaded. Reason: " +
ex.Message);
        Environment.Exit(-1);
    }
}

//Get the summary screen and investigate the first goal on it
InterviewScreen summaryScreen = session.GetScreenService().GetDefaultSummaryScreen();
InterviewGoal goal = null;
foreach (InterviewControl ctrl in summaryScreen.GetControls())
{
    if (ctrl is GoalInterviewControl)
    {
        goal = ((GoalInterviewControl) ctrl).GetGoal();
        break;
    }
}

//now do the investigation
while (!goal.IsKnown())
{
    InterviewScreen screen = session.GetNextScreen(goal);
    renderScreen(screen);
    string input = System.Console.ReadLine();
    submitScreen(session, screen, input);
}

Console.WriteLine("Interview is complete!");
Console.WriteLine("Goal value: " + goal.GetValue());

//Save a case
Console.Write("Enter the name of the case to save this session as [blank to skip]: ");
caseId = Console.ReadLine();
if (caseId.Length > 0)
{
    try
    {
        session.SaveData(token, caseId);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Sorry, case " + caseId + " could not be saved. Reason: " +

```

```

ex.Message);
        Environment.Exit(-1);
    }
}

Console.WriteLine("Hit Enter to exit.");
Console.ReadLine();
}

private static void renderScreen(InterviewScreen screen)
{
    Console.WriteLine("Screen Title: " + screen.GetTitle());
    //in the 10.x engine, an entity collect control will always be the first control on the screen
    bool isEntityCollect = screen.GetControls().Get(0) is ContainmentRelationshipInterviewControl;

    if (isEntityCollect)
    {
        //if it's an entity collect screen, ask for a ';' delimited list of instance names
        Console.Write("Please Enter a ';' list of instances to create: ");
    }
    else
    {
        //in this sample we only have one input control per screen so we'd just ask for a value for
this screen
        foreach (InterviewControl ctrl in screen.GetControls())
            Console.WriteLine(ctrl.GetText());
    }
}

private static void submitScreen(InterviewSession session, InterviewScreen screen, String input)
{
    bool isEntityCollect = screen.GetControls().Get(0) is ContainmentRelationshipInterviewControl;
    if (isEntityCollect)
    {
        ContainmentRelationshipInterviewControl containmentControl = (Con-
tainmentRelationshipInterviewControl)
            screen.GetControls().Get(0);
        String[] instanceNames = input.Split(';');

        //create our entity instances
        foreach (String instanceName in instanceNames)
        {

            EntityInstanceInterviewControl instanceCtrl = containmentControl.AddNewInstance

```

```

(instanceName);
        //each entity instance also has in input control on it to store the child's name,
        //so we'll just reuse the same instance name
        InputInterviewControl ctrl = instanceCtrl.GetControls().Get(0) as InputIn-
terviewControl;
        ctrl.SetValue(instanceName);
    }
}
else
{
    foreach (InterviewControl ctrl in screen.GetControls())
    {
        //this example only support text and statement controls
        if (ctrl is TextInputInterviewControl)
        {
            ((TextInputInterviewControl) ctrl).SetValue(input);
        }
        else if (ctrl is StatementInputInterviewControl)
        {
            try
            {
                Boolean value = bool.Parse(input) ? Boolean.TRUE : Boolean.FALSE;
                ((StatementInputInterviewControl) ctrl).SetValue(value);
            }
            catch
            {
                Console.WriteLine("Can not submit screen because value is not a valid boolean
value: " + input);
            }
            return;
        }
    }
}

TransactionResult result = session.Submit(screen);
if (!result.IsSuccess())
{
    //The transaction failed, so we write out the list of errors
    Console.WriteLine("ERROR: could not submit screen due to the following error(s)");
    foreach (Error error in result.GetErrors())
        Console.WriteLine(error.GetMessage());
    Console.WriteLine("Since the transaction failed, the same screen will now be re-dis-
played.");
}
else

```

```

    {
        //The transactions succeeded but we may have warnings or events
        Console.WriteLine("Transaction succeeded with " + result.GetWarnings().Size() + " warning
(s) and " +
        result.GetEvents().Size() + " event(s) fired");
        Console.WriteLine("The next screen will now be displayed.");
    }
}
}
}
}

```

Integrate the Interview Engine with an external data source

For information relating to the integration of the Interview Engine with an external data source, refer to [Data Adaptor - Common Scenarios](#) where pre-seeding and interview with data, saving data, auto-saving data and using the Data Adaptor to handle data integration are discussed.

See also:

[Data Adaptor plugin overview](#)

[Create a Data Adaptor](#)

Generate an Interview Engine decision document

Users require a transcript when going through any interview, so that they have:

- a record of the interview
- the answers they provided
- the outcome.

The client system running the Web Determinations interview needs to be able to provide this, and also be able to define what data from the interview session is displayed in the transcript, and how it is displayed (layout).

Oracle Policy Modeling allows rulebase authors to create reports of the Web Determinations interview, accessible by the user from the *Summary* screen. With the report links, users can view and download information transcript about the interview session as a *document*.

The Document Generator plugin is an Engine Session plugin to the Interview Engine that allows plugin developers to generate those documents in any document type, and also define the document layout and which session data is displayed. A Document Generator is known by the *document type* it provides; for example, HTML, PDF, XML and so on.

By default, Web Determinations ships with two Document Generators for HTML and PDF. For more information about using these, see the topic, [Use the default Document Generator](#).

Multiple Document Generators can be used together during a Web Determinations interview session, allowing users to view the interview transcript in various formats. This is different to other plugins where only one plugin can be used during an interview session; for example, Data Adaptor plugins. Each Document Generator provides a specific document type, therefore it is imperative that all Document Generators registered during a Web Determinations interview, must provide unique document types.

Go to:

[Document Generator plugin overview](#)

See also:

[Document Generator - sample code](#)

Retrieve Interview Engine commentary

Some screens and attributes during a Web Determinations Interview are complex or ambiguous and need further explanation. The Commentary functionality in Web Determinations allows the rulebase author to provide extra information about screens and attributes, which can be accessed by the user during the interview when more information or explanation is required. The Commentary functionality also allows an external web page to be displayed as commentary.

When commentary is available for a screen or attribute, the attribute/screen label is rendered as a link that the user can click on to view the commentary information. By default, commentary is displayed in a frame to the right of the interview screen. Optionally, it is possible to configure Web Determinations to display commentary information in a new window (more about the configuration in [Configuration files](#)). Attributes and screens do not have to have commentary, and it is possible for a Web Determinations interview to have commentary available for some screens/attributes only.

Commentary plugins are Engine Session plugins that manage commentary in a Web Determinations interview. The main functionality is to determine whether commentary for a screen/attribute is available, and to be able to provide the commentary content to those screens/attributes when requested. Commentary plugins have the flexibility of being able to retrieve content from data-sources accessible in a Java/.NET class, or returning a URL to be displayed in the commentary frame/window. The ability to return a URL allows Commentary plugins that can make use of a client's existing information webpages.

Web Determinations ships with a default Commentary plugin that uses commentary HTML files generated from Oracle Policy Modeling by the rulebase author. More about the default Commentary in [Use the default commentary](#).

Like many other plugins, only one Commentary plugin can be registered per Web Determinations interview.

Go to:

[Commentary plugin overview](#)

See also:

[Use the default commentary](#)

[Pseudo code](#)

[Sample code \(DerbyCommentary\)](#)

[Sample code \(RedirectCommentary\)](#)

[Configuration files](#)

Handle Interview Engine events

For a description of the specific engine and platform events, including details about the objects contained by the events (accessible and modifiable by associated event handler implementations) and the mapping between the events and the event handler interfaces to be implemented by plugin classes in order to be registered as valid event handlers for the respective events, go to the topic, [Events and event handlers](#).

Install an Interview Engine plug-in

For information regarding the installing and registering of Web Determinations extensions, refer to the topic [Plugin loading, invocation and discovery](#).

See also:

[Introduction to Web Determinations extensions](#)

[Web Determinations extensions - the technical details](#)

[Plugin loading, invocation and discovery](#)

Extensions

There are two main types of Extensions:

[Plugins](#)

[Event Handlers](#)

See also:

[Plugins - general technical information](#)

[Plugin loading, invocation and discovery](#)

[Create a plugin](#)

[Report error messages from plugins](#)

Plugins

Web Determinations and the Determinations Server provide a plugin architecture to enable the default behavior of the application to be customized on a per project basis.

For information on creating a plugin, go to [Create a plugin](#); this will help you understand the processes involved in creating the more specific extensions listed below.

Each of the following plugin *types* allows a specific component in the interview to be extended:

- [Data Adaptor](#) - allows you to override the way an interview is saved and loaded.
- [Document Generator](#) - provides new ways for Web Determinations to generate documents about an Interview.
- [List Provider](#) - provides a different way for Web Determinations to obtain the list data of a List Control.
- [Formatter](#) - overrides the way data is displayed.
- [Custom Control](#) - overrides the way a default control is rendered.
- [Custom Screen](#) - overrides the way a screen is displayed.
- [Custom Commentary](#) - provides new ways for Web Determinations to generate commentary information.
- [Rulebase Resolver](#) - uses a custom datasource to store and retrieve rulebases.
- [Custom Service](#) - allows the leveraging of Oracle Policy Automation technology to create a custom SOAP/HTTP based web service (for the Determinations Server).

There are also the following Custom Validators that can be used for validating controls and screens:

- [Custom Validator for control validation](#)
- [Custom Validator for screen validation](#)

Only the Document Generator plugin can have multiple plugin implementations within the same Web Determinations interview; for example, it is possible to write several Document Generators (PDF, custom XML, MSEXcel, and so on) to provide different options in generating documents of the current Interview.

All other plugin types can only have one implementation in an interview; for example, the interview can only use one **DataAdaptorPlugin** implementation, so if there are several **DataAdaptorPlugins** in Web Determinations, the plugin developer must ensure that they register themselves to an Interview exclusively to each other. More information on plugin authoring can be found in [Understanding the InterviewSession](#)

Plugins are created by extending Java/.Net interfaces and deploying them into a specific folder in the Web Determinations web application. It is important to note that while default component behavior is being overridden by writing and providing custom plugins, it is necessary to understand how Web Determinations uses plugins because they cannot alter the way each plugin type is used.

More information about each plugin type can be found in their respective plugin information articles, including their 'variety' class; for example, Data Adaptors are Engine Session plugins.

See also:

[Plugins - general technical information](#)

Event Handlers

Event Handlers provide the most flexibility for extending Web Determinations, since plugin extensions only override a certain set of components. Web Determinations provides a set of events fired at specific points in an Interview that can be hooked into to provide project-specific functionality.

When Web Determinations goes through an interview, events are fired at certain points throughout the session. When these events are fired, the Event Handlers for that specific event are called to provide their added functionality. Each event is different to each other in terms of what interview data it exposes and what point in the Web Determinations session it is called.

Each event will expose data to its Event Handler/s so that a handler has enough knowledge about the *context* of when the event was fired, and also enough accessibility/permission to modify certain parts of Web Determinations session data and thus provide functionality.

An Event Handler might be written for a specific event by extending the **EventHandler** (Java/.Net) Interface of the event and deploying it to Web Determinations which will then call that Event Handler when the event is fired.

It is important to be familiar with when a particular event is fired, what the expected usage of the event is, and the interview data it exposes.

For more details about each event and event handling, go to [Events and Event Handlers](#).

Plugins - general technical information

The following information provides guidelines and details for programmatically setting up Plugin classes, and loading them into either a Web Determinations server or the Determinations Server.

From a technical viewpoint, extensions are Java or .Net classes that implement Java/.Net interfaces to provide plugins to either Web Determinations or the Determinations Server. As illustrated in the topic [Introduction to extensions](#), there are two main types: Plugins and Event Handlers; each type has several subtypes.

The interfaces that need to be implemented for each subtype differ; this information can be found in the plugin-specific topics.

The general approach to developing and implementing a plugin is:

1. Determine what the plugin (or group of plugins) needs to be able to do.
2. Determine what specific plugin/s need to be implemented to achieve the required solution.
3. For each plugin, create and develop the class that implements the required subtype's interface; for example, a List Provider plugin implements the **ListProviderPlugin** interface.
 - Steps to develop the class can be found in [Create a plugin](#).
4. Load the plugins together with the code libraries that it depends on in the appropriate place (that is, either Web Determinations server or Determinations Server).

Creating a Web Determinations plugin class

Programmatically, Event Handlers are actually a type of plugin in that an Event Handler's interface inherits from the plugin interface.

Each Web Determinations plugin class implements at least one interface; for example:

DataSubmissionAborter class

```
.. public class DataSubmissionAborter implements BeforeSubmitDataEventHandler { ..
```

A Web Determinations plugin class can also implement more than one interface for convenience, although it might have an effect on maintainability.

Plugins

Below is a list of specific plugin types (and the corresponding interface).

Plugin Type	Interface	Ancestor Interface
Data Adaptor	DataAdaptorPlugin	EnginePlugin
Document Generator	DocumentGenerationPlugin	EnginePlugin
Commentary Provider	CommentaryProviderPlugin	EnginePlugin
List Provider	ListProviderPlugin	EnginePlugin
Custom Screen Provider	CustomScreenProvider	PlatformPlugin
Custom Control Provider	CustomControlProvider	PlatformPlugin
Web Determinations Formatter	WebDeterminationsFormatter	PlatformPlugin

The Web Determinations web application comes with default Plugin implementations for each type of plugin; for example, **XSDDataAdaptor** for **DataAdaptorPlugin**.

As mentioned in [Introduction to plugins](#), only one plugin can be loaded per Web Determinations interview (with the exception of Document Generator plugins). This means that if the developer chooses to develop a custom plugin for one of the plugin types, the custom plugin overrides the default plugin implementation.

The plugin developer can still access the functionality provided by each of the default plugin implementation as each implementation is in the Web Determinations library (the .jar or .dll file, see [Create a plugin](#)). For example, the developer can create a custom **DataAdaptorPlugin** that simply calls the corresponding **XSDDataAdaptor** method except for specific circumstances which then uses the custom **DataAdaptorPlugin** logic.

Plugin rules

Regardless of the specific interface implemented, all plugin implementations must conform to the following to be loaded and invoked properly by the Oracle Web Determinations plugin framework:

- they must implement the **getInstance(args : RegisterArgs) : Plugin** method. This method allows a Web Determinations plugin class to read the **RegisterArgs** and its contents to determine whether it should be registered to the current Web Determinations interview (by returning an instance of itself).
 - As an example, this allows the plugin to only become loaded on specific rulebases, or locale, and so on if the rule-base or locale data is in the passed **RegisterArgs**.
 - This is critical for ensuring that only one plugin is registered per Web Determinations interview.
Note: The above **RegisterArgs** is more specific depending on the plugin; for example, DataAdaptorPlugins receive InterviewSessionRegisterArgs, CustomScreenProviders receive PlatformSessionRegisterArgs, and so on
- they must have a parameter free constructor.
- they must correctly implement all methods defined by the specific interface that it implements. This includes methods defined in super-interfaces of the specific interface implemented by the plugin.
- See the following sample code:

Sample Plugin - RulebaseSpecificDataAdaptor

```
public class RulebaseSpecificDataAdaptor implements DataAdaptorPlugin { //TIP: Each Plugin implements a
Plugin Interface

    /**
     * Only provide this plugin if the Web Determinations session's rulebase is the 'SpecialRulebase' rulebase.
     * The specific RegisterArgs object we get here is an instance of InterviewSessionRegisterArgs.
     */
    public Plugin getInstance(InterviewSessionRegisterArgs args) { //TIP: Each Plugin interface implemented
        have different getInstance() argument
        if (args.getSession().getRulebase().getIdentifier().equals("SpecialRulebase")){
            return new RulebaseSpecificDataAdaptor ();
        } else {
            return null;
        }
    }
}
```

```
/**
 * Parameterless constructor to conform to the plugin requirements
 */
public RulebaseSpecificDataAdaptor (){
}
...

```

For a more comprehensive example of sample code, see [Data Adaptor - pseudo code](#).

Event Handlers

There are much more Event Handler interfaces to implement than plugins. To see the full list of Event/Event Handlers to implement, see [Events and Event Handlers](#)

As mentioned before, Event Handlers are also plugins and so must conform to the same rules outlined in *Plugin rules* above.

Below are the steps in which an Event is fired and an Event Handler consumes it:

1. All Event Handlers are first registered as a normal plugin - that is it can choose if it gets registered based on the **getInstance()** args such as current rulebase, and so on.
2. Each Event Handler is then registered to the Events that it is supposed to handle.
3. When an Event is fired, all Event Handlers registered to handle it receives the fired Event.
4. The Event Handler receives the fired Event, and can provide functionality.

Each Event type is fired at different times during the Web Determinations interview, and expose different data to each other; it is therefore important to understand:

- the intended usage of the Event and why it is being exposed by the Web Determinations Server.
- when the Event is fired.
- what objects/data are available in the Event object.
- what the sender object is.

In addition to the plugin rules, all Event Handlers must implement the **handleEvent(Event event)** function. The **Event** input argument is specific to each Event Handler since each Event Handler is setup to receive a specific Event.

See the sample code below; because it is an **OnInvestigationEndedEventHandler**, the **handle()** method receives an **OnInvestigationEndedEvent** object.

Sample Event Handler - RulebaseSpecificHandler

public class RulebaseSpecificHandler implements OnInvestigationEndedEventHandler{ //TIP: Similar to Plugins, each Event Handler implements an interface based on the Event type it handles

```
/**
 * Check the type of event and output a message accordingly.
 */
public void handleEvent(OnInvestigationEndedEvent event, Object sender) { //TIP: Each Event Handler
handleEvent() has different Event and sender objects passed in //handler logic

}

/**
 * Only provide this plugin if the Web Determinations session's rulebase is the 'SpecialRulebase' rulebase.
 * The specific RegisterArgs object we get here is an instance of InterviewSessionRegisterArgs.
 */
public Plugin getInstance(PlatformSessionRegisterArgs args) { //TIP: Similar to Plugins, each Event Handler
interface implemented have different getInstance() argument
if (args.getContext().getInterviewSession().getRulebase().getIdentifier().equals("SpecialRulebase")){
return new RulebaseSpecificHandler();
} else {
return null;
}
}

/**
 * Parameterless constructor to conform to the plugin requirements
 */
public RulebaseSpecificHandler(){

}
}
```

Installing Web Determinations plugins into a Web Determinations server

To find out what is involved in installing Web Determinations plugins into a Web Determinations server, go to the topic, [Install and register plugins](#).

Report error messages from plugins

If your plugin is passed a **TransactionResult** object, it may have been passed by one of the following events:

- **BeforeSubmitDataEvent**
- **OnCommitEvent**
- **OnSubmitDataEvent**
- **OnSubmitRollbackEvent**
- **OnValidateControlEvents**
- **OnValidateScreenEvent**

For these events, you attach an **Error** object to the transaction result, in order to describe the error using **addError()**. Other objects that it would be appropriate to attach are the **GenericError** class and its subclasses.

For more information, see the **com.oracle.determinations.interview.engine.data.error** package in the Determinations Interview Engine API documentation.

Other events and plugins aren't passed a **TransactionResult**, so in those cases, an **Exception** object must be thrown instead. In this case, the **PluginException** class and its subclasses are appropriate exceptions to be raised.

See the **com.oracle.determinations.interview.engine.exceptions** package in the Determinations Interview Engine API documentation.

Localize messages

Messages need to be localized when they are intended for users that speak different languages. Error messages from events and plugins can be localized in a manner similar to that in which rulebases are localized.

The best way to localize an error message is to supply a message code, which is then looked up within *messages.XX.properties* to supply the localized message. The XX here refers to the user's locale.

How web determinations determines the message code

Web Determinations will determine whether the error object implements **CodedMessage**. Both **GenericError** and **PluginException** implement this interface. If this is the case, it will retrieve a message code from the **getMessageCode()** method. If this message code can be found in its *messages.XX.properties* file, then it is done.

Otherwise, it will look at the class name of the error, and look that up in *messages.XX.properties*. If it cannot find the class name, it will recursively search for superclass names until a matching name can be found. If no name can be resolved, it will use the message provided in the **GenericErrorMessage** template.

How to pass parameters to a localized error message

Parameters are passed in the **Error** or **Exception** object by way of getter methods; for example, **UnknownAttributeError** provides the **getAttributeId()** and **getEntityInstanceIdentifier()** methods. Because the message text is interpreted by Web Determinations's templating engine, these can be interpreted within the message text.

As an example, for **UnknownAttributeError**, the line in the English localization property file reads:

```
Can not find attribute '{message.getAttributeId()}' in entity '{message.getEntityInstanceIdentifier().getEntityId()}'
```

The **getAttributeId()** call accesses the **Error** or **Exception** object to provide the actual **Attribute ID** that failed to load. If you want to pass additional parameters, the best way is to subclass the appropriate error object, and include getter methods for the parameters you want to pass. These can be called from the template supplied in *messages.XX.properties*.

Interview Engine Plugins

CommentaryProviderPlugin

An exception must be thrown.

getCommentaryURL() and **getCommentaryContent()** can throw **PluginException** or subclasses.

Other methods should only return boolean values; that is, **hasCommentary()**, **isCommentaryEnabled()**, **isCommentaryRedirect()**.

See the [examples\interview-engine\derby-commentary](#) example in Interview Engine customizations.

DataAdaptorPlugin

An exception must be thrown.

listCases(), **load()**, and **save()** can throw **PluginException** or subclasses.

One other method should only return a boolean; that is, **dataAdaptorProvidesCaseID()**.

See the [examples\interview-engine\data-adaptor](#) example in Interview Engine customizations.

ListProviderPlugin

An exception must be thrown.

getListOptions() can throw **PluginException** or subclasses.

See the [examples\interview-engine\list-provider](#) examples in Interview Engine customizations.

DocumentGeneratorPlugin

An exception must be thrown

generateDocument() can throw **PluginException** or subclasses.

See the [examples\interview-engine\document-generator](#) example in Interview Engine customizations.

OnValidateControlEventHandler

Provides a **TransactionResult**.

handleEvent() can attach **GenericError** and subclasses to the provided **TransactionResult**.

See the [examples\interview-engine\custom-control-validator](#) example in Interview Engine customizations.

OnValidateScreenEventHandler

Provides a **TransactionResult**.

handleEvent() can attach **GenericError** and subclasses to the provided **TransactionResult**.

See the [examples\interview-engine\custom-screen-validator](#) example in Interview Engine customizations.

Web Determinations plugins

CustomInputControl

An exception must be thrown

mapValues(), **extractValue()**, **getDisplayValue()**, **getValue()**, **getListDisplayForValue()** can throw **PluginException** or subclasses.

See the [examples\web-determinations\custom-control](#) and [examples\web-determinations\calendar-date-control](#) examples in Web Determinations customizations.

CustomScreen

An exception must be thrown

processAndRespond() and **submit()** can throw **PluginException** or subclasses.

See the [examples\web-determinations\custom-screen](#) example in Web Determinations customizations.

WebDeterminationsFormatterPlugin

An exception must be thrown

parse() and **getFormattedValue()** can throw **PluginException** or subclasses.

See the [examples\web-determinations\custom-formatter](#) example in Web Determinations customizations.

OnGetScreenEventHandler

An exception must be thrown

handleEvent() can throw **PluginException** or subclasses.

See the [examples\web-determinations\auto-save](#) example in Web Determinations customizations.

OnInvestigationEndedEventHandler

An exception must be thrown.

handleEvent() can throw **PluginException** or subclasses.

See the [examples\web-determinations\auto-save](#) example in Web Determinations customizations.

Create a plugin

The following information describes the steps required to create a plugin, install it onto a Java webserver, run the plugin, and debugging tips. While this specifically describes a Java implementation, similar steps apply for .Net

Assets and information needed

The following provide information that will assist in understanding what is happening in the steps below:

Web Determinations plugin folder - this is where the Web Determinations webserver will pick up the plugins. The user needs to place the plugin packages (.jar for Java).

1. Tomcat (Java) - default is in the `WEB-INF\classes\plugins` path in the Web Determinations web application in the Tomcat webapps folder,
 - for example: `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\classes\plugins`
2. IIS (.Net) - default is in `C:\inetpub\web-determinations\plugins`

Web Determinations library jars - these can be found in the Web Determinations 'lib' folder; for example: `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\lib`

- `determinations-engine.jar`
- `determinations-interview-engine.jar`
- `determinations-utilities.jar`
- `web-determinations.jar`

Java IDE - this is the Java development application you are using. While this is optional, it will make it much easier when referring to objects in the `web-determinations.jar` and `determinations-interview-engine.jar`.

Steps

To develop a plugin, do the following:

1. Determine what the plugin is extending (the plugin type), and the Java interface the Java plugin needs to extend. See [Plugins - general technical information](#).
2. If you are using a Java IDE - set it up with a Java project and load the Web Determinations library jars as Java libraries. The Web Determinations library jars have Javadoc, which will make it much easier when using Web Determinations Java objects in the jars.
3. Create a Java Class for the Plugin in the new Java project, and implement the required interface and its methods in the new Class.
 1. More information can be found on the specific pages of each plugin type.
 2. Technical details about plugin authoring can be found in [Plugins - general technical information](#).

To install the plugin, do the following:

1. Package the Plugin Class and any other Java Class developed with it in a .jar file.
2. Deploy the jar file into the Web Determinations plugin folder, together with any other libraries that were required (except the Web Determinations library jars - they are already in the 'lib' folder as mentioned previously).
3. Restart the Tomcat server to enable the plugin.

To test the plugin, do the following:

1. The first step to check is if it was recognized and loaded by the Tomcat server. The Web Determinations web application prints out data about plugins that were loaded into the runtime successfully. Any errors when loading a plugin is also printed out. Check the webserver's output file (for example: *stdout.txt* for Tomcat).
2. The next step is to check if the plugin is registered. Depending on the plugin type - sometimes it might be loaded when the main web application root is accessed, or others when a Web Determinations interview is started (that is, rulebase and locale has been selected). If you encounter problems getting the plugin registered, ensure that your plugin **getInstance** method simply returns the plugin object, for example: "return new MyDataAdaptor()".
3. Finally, if the plugin is loaded and registering correctly, test its functionality.

Plugin loading, invocation and discovery

This topic is intended as a guide for implementing plugins that are loaded and invoked by the extension framework.

Plugin interface

All extension implementations must implement an interface that is descended from the super plugin interface known as **Plugin**. Typically, you are required to implement the most specific interface in the hierarchy, according to the type of plugin that you are writing; for example, if you wanted to implement a plugin that was an event handler for the **OnSessionCreatedEvent** engine event, you would implement the **OnSessionCreatedEventHandler**, **NOT** the generic **EngineEventHandler** interface.

Regardless of the specific interface implemented, all plugin implementations must conform to the following to be loaded and invoked properly by the extension framework:

- Must implement the **getInstance(args : RegisterArgs) : Plugin** method. This method is to return an instance of the plugin object if the plugin has determined (using the contents of the specific **RegisterArgs** implementation instance passed) that it is usable.
- Must have a parameter free constructor.
- Must correctly implement all methods defined by the specific interface that it implements. This includes methods defined in super-interfaces of the specific interface implemented by the plugin.

Extension types

The point at which an extension is registered and the registration arguments that are passed to it, is determined by its type. There are several types of plugins:

- **Engine plugin:** Interview Engine plugins are available to customize the behavior of the Interview Engine. Interview Engine plugins are registered when the instance of an engine is create.
- **Platform plugin:** plugins that are available in the Web Determinations web application, as opposed to the Interview Engine or session. These are registered when an application session is created, as distinct from an Interview Session.
- **Service plugin:** Determinations Server plugin that allows the leveraging of the Web Determinations technology to create a custom SOAP/HTTP based web service.
- **Session plugin:** plugins that attach to a particular Interview Session. Interview Session plugins are registered when the instance of a session is created.

What do you want to do?

[Understand what installation methods are available](#)

[Install extensions to work on Oracle Policy Modeling Build and Run instances](#)

[Register an extension](#)

Understand what installation methods are available

There are two methods of installing extensions:

1. Auto-discovery
2. Manual configuration

Auto-discovery

It is possible to automatically discover and register extensions when the applications start. Using this method, plugin installation is simply a matter of compiling the plugin libraries (.jar in Java or .dll in .NET) and deploying them to the plugins folder and restarting the application. The plugins folder is located in `<web root>/WEB-INF/classes/plugins` in Java or `<web root>/bin/plugins` in .NET. Multiple plugins can be compiled into a single library or multiple plugin libraries can be used. Any third party dependencies must be copied into a location that will cause them to be automatically loaded when the applications starts such as `<web root>/WEB-INF/lib` in Java or `<web root>/bin/` in .NET.

Note for Java Deployments:

Because the Auto-discovery mechanism requires the ability to introspect the class path, this method may not be available in certain application servers such as Oracle WebLogic Application server, when running the war file unexpanded. In such cases, plugins must be loaded using the manual configuration option

Manual configuration

Manually configuring which plugins to load involves deploying the extensions, and any third party dependencies to a location that will automatically be loaded when the application starts such as `<web root>/WEB-INF/lib` in Java or `<web root>/bin/` in .NET. The list of plugin classes to be loaded must then be manually specified using the `plugin.libraries` property in `application.properties`

Install extensions to work on Oracle Policy Modeling Build and Run instances

Installing extensions to enable Oracle Policy Modeling Build and Run instances to use those extensions, is the same process as installing either onto the Determinations Server or onto a Web Determinations server.

The Oracle Policy Modeling Build and Run command invokes a Java web server as well, and its plugin folder can be found in the 'Release' folder directly in the project folder; for example, if we have a rulebase named 'MyRulebase', located in `C:\-projects\MyRulebase`, the plugins folder will be located in the path:

`C:\projects\MyRulebase\Release\web-determinations\WEB-INF\classes\plugins`

Register an extension

Regardless of the specific interface implemented, all extensions must conform to the following to be loaded and invoked properly by the extension framework:

- Must implement the **getInstance(args : RegisterArgs) : Plugin** method. This method allows an extension class to read the **RegisterArgs** and its contents to determine whether it should be registered to the current Web Determinations interview (by returning an instance of itself). In addition to that, because the plugin has control on what instance of itself it returns, it is able to instantiate an instance of itself with arguments from the **RegisterArgs** args.
 - This allows the plugin to only become loaded on specific rulebases, or locale, and so on if rulebase or locale data is in the passed **RegisterArgs**
 - This also allows the plugin to choose which constructor to use for instantiation; for example, new **CustomPlugin (SessionContext ctx)**
 - This is critical for ensuring that only one plugin is registered per Web Determinations interview

Note:

The above **RegisterArgs** is more specific depending on the plugin; for example, **DataAdaptorPlugins** receive **InterviewSessionRegisterArgs**, **CustomScreenProviders** receive **PlatformSessionRegisterArgs**, and so on.

- In is permissible to implement singleton plugins by returning the same instance of the plugin for multiple invocations of **getInstance()**, however, it is the implementer's responsibility to ensure thread safety in such situations.
- Must have a public parameter free constructor.
- Must correctly implement all methods defined by the specific interface that it implements. This includes methods defined in super-interfaces of the specific interface implemented by the plugin.

Code Sample - Sample Plugin - RulebaseSpecificDataAdaptor

```
public class RulebaseSpecificDataAdaptor implements DataAdaptorPlugin { //TIP: Each Plugin implements a
Plugin Interface

    /**
     * Only provide this plugin if the Web Determinations session's rulebase is the 'SpecialRulebase' rulebase.
     * The specific RegisterArgs object we get here is an instance of InterviewSessionRegisterArgs.
     */
    public Plugin getInstance(InterviewSessionRegisterArgs args) { //TIP: Each Plugin interface implemented
have different getInstance() argument
        if (args.getSession().getRulebase().getIdentifier().equals("SpecialRulebase")){
            return new RulebaseSpecificDataAdaptor ();
        } else {
            return null;
        }
    }

    /**
     * Parameterless constructor to conform to the plugin requirements
     */
    public RulebaseSpecificDataAdaptor (){

    }

    ...
}
```

Notes about Hot-swapping mode

If Hot-swapping mode is turned on, then:

- When a new rulebase is added, the Custom Service plugin will be given an opportunity to register a service against that rulebase.
- When a rulebase is deleted, any service registered against it will be automatically unloaded.
- Updating an existing rulebase is equivalent to a delete and load.

Plugins and RegisterArgs

The following table indicates which specific **RegisterArgs** implementation objects may be passed to the **getInstance** method of descendant interfaces of the base **Plugin** interface.

REGISTERARGS implementation	Encapsulated objects	Plugin Descendent interface (s)
InterviewEngineRegisterArgs	- The Interview Engine in use.	
InterviewSessionRegisterArgs	- The current InterviewSession .	<ul style="list-style-type: none"> • EngineEventHandler • DataAdaptorPlugin • DocumentGenerationPlugin • CommentaryProviderPlugin • ListProviderPlugin
PlatformSessionRegisterArgs	- The current SessionContext .	<ul style="list-style-type: none"> • CustomScreenProvider • CustomControlProvider • PlatformEventHandler • WebDeterminationsFormatter
InterviewServletRegisterArgs	- The InterviewServletContext object in use.	

Plugin type	Interface	Ancestor interface
Data Adaptor	DataAdaptorPlugin	EnginePlugin
Document Generator	DocumentGenerationPlugin	EnginePlugin
Commentary Provider	CommentaryProviderPlugin	EnginePlugin
List Provider	ListProviderPlugin	EnginePlugin
Custom Screen Provider	CustomScreenProvider	PlatformPlugin
Custom Control Provider	CustomControlProvider	PlatformPlugin
Web Determinations Formatter	WebDeterminationsFormatter	PlatformPlugin

Do's and don'ts

- As part of the initialization logic implemented in your implementation of the **getInstance** method or the parameter free constructor, **DO NOT** set a static (class-scope) reference to any of the objects encapsulated by the given **RegisterArgs** parameter.
- It is not permissible for plugins to directly alter the state of any of **RegisterArgs** members passed in the get instance method.

- Do not make assumptions about the number of times that the **getInstance** method is called. It is up to the logic implemented in this method to determine whether the plugin is to be available in a particular context. Every call to this method is another chance to the plugin to inform the application that it is ready to be used or to invalidate itself for use.
- Once the first call to the **getInstance** method returns, the plugin's code may be called at any point. That is, *you may not make any assumptions about the order and number of calls into your code.*

Formatter plugin overview

There will be times where a Web Determinations interview will have an attribute that requires custom input data, or custom output format of the value of the attribute.

For example, say we have two variable 'number' attributes called 'longitude' and 'latitude'. While it is stored in Web Determinations as a double, during the Web Determinations interview it should allow the user to enter proper longitude and latitude formats; that is, angular measurements that consist of degrees/minutes/seconds and direction (for example, 66 33' 39" N).

Formatter plugins are Platform Session plugins that allow Web Determinations to correctly parse custom input data formats, and correctly 'format' custom output data formats.

With the above example:

- The Formatter will handle processing of input attribute values that are of angular measurement format and convert them to a double (number).
- The Formatter will handle formatting and displaying of attribute values stored in double (number) that should be displayed in angular measurement format.

See the topic, [Formatter - sample code](#).

Web Determinations ships with a default Formatter. There are no special steps involved in using the default Formatter as it is automatically used to process input and output formats based on the attribute type. The default Formatter does have some very basic configuration to modify formats for Date, DateTime, Currency. More information about configuring these formats can be found in [Input and output formats](#).

Like many other plugins, only one Formatter plugin can be registered per Web Determinations interview. The default Formatter provides important core functionality that needs to be extended (rather than overridden) - therefore custom Formatter plugins in almost all cases, delegate the core functionality to the default Formatter plugin except for functionality that needs to be customized/overridden. For more information about this, refer to the topic below, [Developing a Formatter for a specific project/implementation](#).

Common scenarios

Ability to accept custom input format

A Web Determinations interview has a 'Number' variable attribute (or several attributes) that requires a standardized input format, for example angular measurements such as Latitude and Longitude(degrees/minutes/seconds and direction). The user needs to be able to enter standard angular measurement format(s) for fields that gather input data for those attributes in the Web Determinations interview, and those input values need to be translated to the number value (the format in which Web Determinations stores it).

Ability to display custom output format

Following custom input format above, a Web Determinations interview has a 'Number' variable attribute (or several attributes) it needs to display in the aforementioned Latitude/Longitude format. When the value of the attribute(s) needs to be displayed to the user, it needs to be converted from a number value (the format in which Web Determinations stores it) into the standardized angular measurement with direction .

Formatter and the Web Determination architecture

This section details how the Formatter fits into the Web Determinations architecture, and how to use it in the Web Determinations environment.

How Web Determinations use Formatter plugin by default

The default Formatter (DefaultFormatter) plugin is used automatically as it provides core functionality to Web Determinations in terms of parsing all input data and formatting all output data. All input data for attributes provided during the Web Determinations interview; for example, in interview question screen is processed by the registered Formatter plugin. All output data displayed from attribute values; for example, Decision Report for an attribute also uses the registered Formatter plugin.

While other plugins are optional (for example, a Web Determinations interview does not need to have a Data Adaptor), or a List Provider plugin registered to the current interview session, a Formatter must always be registered to a Web Determinations interview. Furthermore, if the registered Formatter is a custom Formatter, it should also provide all the functionality the default Formatter provides together with custom formatting.

Formatter plugin and other Web Determination extensions

Most Web Determinations extensions are provided with input data that has already passed through the registered Formatter plugin, and thus has no need to directly access and use the Formatter. Also, output data returned by Web Determinations extensions to Web Determinations for display, usually passes through the registered Formatter for formatting before being displayed, and thus has no need to directly access and use the Formatter.

Formatter plugins are useful only to Web Determinations extensions that have to handle raw input and output data. These extensions are the Custom Screen and Custom Control plugins. Custom formatting of input and/or output data that is used by multiple Custom Screens/Controls should be contained in a Formatter.

Formatter class methods - functional methods

A Formatter plugin implements the interface **WebDeterminationsFormatterPlugin**. The **WebDeterminationsFormatterPlugin** interface in turn extends the **PlatformSessionPlugin** and **Formatter** interface.

Below are important methods the **WebDeterminationsFormatterPlugin** interface requires when implemented. For details on other methods required, see the API documentation for the **WebDeterminationsFormatterPlugin**.

Method signature	Description
Object parse(byte type, String value, Attribute attr)	<p>Parses an input data value (String value) and returns its object representation using the input format(s) for the specified attribute value. Used by the Web Determinations to parse input data into a Ruleengine datatype (byte type).</p> <ul style="list-style-type: none">'type' relates to the value's Ruleengine datatype; for example, Boolean, Number, Date, and so on. See the HaleyType object (com.oracle.determinations.engine.HaleyType) for more information.'value' is the actual value of the input data, as a String (since all data from the Web Determinations interview webpages are submitted as String).'attr' is the Attribute object to which the value belongs.the returned Object is a standard Java/.NET data type. The calling method knows the HaleyType, and can therefore cast this object to the corresponding Java/.NET datatype.

Object parse(byte type, String value)	<p>Parses an input data value (String value) and returns its object representation using the input format(s) for the specified attribute value. Used by the Web Determinations to parse input data into a Ruleengine datatype (byte type).</p> <ul style="list-style-type: none"> 'type' relates to the value's Ruleengine datatype; for example, Boolean, Number, Date, and so on. See the HaleyType object (com.oracle.determinations.engine.HaleyType) for more information. 'value' is the actual value of the input data, as a String (since all data from the Web Determinations interview webpages are submitted as String). the returned Object is a standard Java/.NET data type. The calling method knows the HaleyType, and can therefore cast this object to the corresponding Java/.NET datatype.
String getFormattedValue(byte type, Object value, Attribute attribute)	<p>Formats a Ruleengine data value into output data (its formatted display representation) according to the specified Ruleengine datatype (byte type).</p> <ul style="list-style-type: none"> 'type' relates to the value's Ruleengine datatype; for example, Boolean, Number, Date, and so on. See the HaleyType object (com.oracle.determinations.engine.HaleyType) for more information. 'value' is the Ruleengine data value, represented by an Object. This Object is a standard Java/.NET datatype based on the Ruleengine datatype. 'attr' is the Attribute object to which the value belongs. the returned String value is the value to be displayed.
String getFormattedValue(byte type, Object value)	<p>Formats a Ruleengine data value into output data (its formatted display representation) according to the specified Ruleengine datatype (byte type).</p> <ul style="list-style-type: none"> 'type' relates to the value's Ruleengine datatype; for example, Boolean, Number, Date, and so on. See the HaleyType object (com.oracle.determinations.engine.HaleyType) for more information. 'value' is the Ruleengine data value, represented by an Object. This Object is a standard Java/.NET datatype based on the Ruleengine datatype. the returned String value is the value to be displayed.

Ruleengine datatypes

Below is a table listing the various **Ruleengine** datatypes (which correspond to the various attribute datatypes that exist when authoring rules in Oracle Policy Modeling) and the Java/.NET Object that corresponds to the datatype.

Note that the **Ruleengine** datatype column is derived from HaleyType values (see API documentation for com.oracle.determinations.engine.HaleyType).

Ruleengine datatype	Java/.NET object
Boolean	Boolean
Statement	Boolean
Text	String
Number	Double
Currency	Double

Date	Date
DateTime	Date
TimeOfDay	com.oracle.determinations.engine.DeterminationsEngineTimeOfDay

Developing a Formatter for a specific project/implementation

This section explains the various approaches on designing and developing a Formatter plugin for a specific project/implementation.

Common design approach when building a custom Formatter plugin

A custom Formatter is usually required when specific and custom input and/or output format of a particular attribute's data (or attributes') need to be supported.

As was previously mentioned, because Formatters perform core functionality for Web Determinations (that is, parsing/formatting most or all input and output data), custom Formatters usually only need to extend the default Formatter (or override formatting of an existing **Ruleengine** datatype).

Therefore a custom Formatter should delegate function calls to the **DefaultFormatter** for normal (core) processing of input/output data, except in cases where the input/output data requires custom formatting (by inspecting the attribute members).

Using attribute members to determine custom formatting needs

A common way to determine if the input/output data needs custom formatting is by accessing members of the attribute object.

- The attribute's (custom) properties, set in Oracle Policy Modeling by the rule author, is the best member to use. It is much more maintainable and flexible since the attribute custom properties were designed for custom functionality like this.
- The attribute's name and other members can also be used if necessary.

As an example, where Web Determinations needs to be able to process standardized angular movement formats (latitude and longitude), attributes can have a custom property with the key 'latlongformat' and value of 'longitude' or 'latitude'. Attribute custom properties are usually added by the rulebase author in Oracle Policy Modeling.

Note: Longitude can only be East(E) or West(W) directions while Latitude can only be North(N) or South(S) - hence the need for different values in the 'latlongformat' property

In the custom Formatter plugin, the existence of the key in the attribute's properties can trigger the custom formatting, and the key's value determines whether the attribute should be parsed/formatted as a Longitude (which is degrees/minutes/seconds EorW) or Latitude (same as longitude but expects NorS). See the sample code below for code implementation of this design.

About the data and objects passed into the Formatter methods

Below are descriptions of the input data for the main methods.

parse(byte type, String value, Attribute attribute)

- byte type - the value of the byte corresponds to byte values in the HaleyType object (com.oracle.determinations.engine.HaleyType). See Ruleengine datatype
- String value - the input value from the Web Determinations interview, which is always a String value as data passed in from webpages are always String

- Attribute attribute - the Attribute object linked to the input data value being passed in for parsing. See API documentation for the Attribute object (com.oracle.determinations.engine.Attribute).

getFormattedValue(byte type, Object value, Attribute attribute)

- byte type - the value of the byte corresponds to byte values in the HaleyType object (com.oracle.determinations.engine.HaleyType). See Ruleengine datatype
- Object value - the output value from Web Determinations that needs to be formatted to the return String value for display. See Ruleengine datatype for the Java/.NET to HaleyType mapping.
- Attribute attribute - the Attribute object linked to the output data value being passed in for formatting. See API documentation for the Attribute object (com.oracle.determinations.engine.Attribute).

How to use the data passed into the Formatter methods

The sample code below demonstrate the following

- usage of the input arguments passed in the **parse()** and **getFormattedValue** methods.
- implementation of a custom Formatter plugin delegating to the **DefaultFormatter** for core formatting, and diverting to a custom formatting method if the input/output data to be formatted needs custom formatting.

The **needsCustomFormat()** method encapsulates logic that checks whether a parse/formatting request needs custom formatting, or simply needs the default core formatting. The sample below checks the properties of the Attribute object, by taking in the Attribute object, and the HaleyType byte data. It checks whether the attribute has a property with a certain key value to determine if the input/output data needs custom formatting.

Sample Formatter Codeblock

```
public class CustomFormatter implements WebDeterminationsFormatterPlugin {

    private DefaultFormatter defaultFormatter;

    //REQUIRED - for Plugin architecture
    public CustomFormatter ()
    {

    }

    public CustomFormatter (DefaultFormatter defaultFormatter)
    {
        this.defaultFormatter = defaultFormatter;
    }

    public PlatformSessionPlugin getInstance(PlatformSessionRegisterArgs args) {
        //Demonstration of a Plugin only registering if the current rulebase is 'CustomFormatter'
        if (args.getContext().getInteractiveSession().getRulebase().getIdentifier().equals("CustomFormatter")){
            DefaultFormatter defaultFormatter = new DefaultFormatter(args.getContext());
            return new CustomFormatter (defaultFormatter);
        } else {
```

```

        return null;
    }
}

public String getFormattedValue(byte type, Object value, Attribute attr) {
    if(!needsCustomFormat(type, attr).equals(""))
    {
        //Perform custom formatting
        return customFormatting(type, value, attr);
    }
    else
    {
        //Perform normal (core) formatting - delegate to DefaultFormatter
        return defaultFormatter.getFormattedValue(type, value, attr);
    }
}

```

```

public Object parse(byte type, String value, Attribute attr) {
    if(!needsCustomFormat(type, attr).equals(""))
    {
        //Perform custom parsing
        return customParsing(type, value, attr);
    }
    else
    {
        //Perform normal (core) parsing - delegate to DefaultFormatter
        return defaultFormatter.parse(type, value, attr);
    }
}

```

```

}
/**

```

* Checks if the Attribute needs the custom formatting that this class provides, using the (custom) Properties of the Attribute

```

*

```

```

* @param type the HaleyType

```

```

* @param attr

```

```

* @return

```

```

*/

```

```

private String needsCustomFormat(byte type, Attribute attr)

```

```

{

```

```

    //Perform logic that checks if the Attribute meets the requirements for a custom parse/formatting

```

```

    Map<String, String> attributeProperties = attr.getProperties();

```

```

    Set attributePropKeys = attributeProperties.keySet();

```

```

    Iterator keyIterator = attributePropKeys.iterator();

```

```

    while(keyIterator.hasNext())

```

```

    {

```

```

        String propertyKey = (String)keyIterator.next();

```

```
        String propertyValue = (String)attributeProperties.get(propertyKey);
        if( propertyKey.equals("<expected key value>"))
        {
            return propertyValue;
        }
    }

    return "";
}
/**
 *
 * @param type
 * @param value
 * @param attr
 * @return
 */
private Object customParsing(byte type, Object value, Attribute attr)
{
    //Perform custom parsing
    return null;
}

private String customFormatting(byte type, Object value, Attribute attr)
{
    //Perform custom formatting
    return "";
}
```

Create a Rulebase Resolver plugin

The Rulebase Resolver plugin is responsible for locating and providing access to the archive streams (and associated module streams) of the rulebases that are available from the Oracle Determinations Interview Engine. By default the Interview Engine provides two methods for loading rulebases, either via the local file system using the **FileRulebaseResolverPlugin** or (for Java deployments only) via the Java Classpath API using the **ClassLoaderRulebaseResolverPlugin**. This default behavior, however, can be customized by implementing the **RulebaseResolverPlugin**.

Overview

The responsibility for loading and maintaining rulebases is shared between two components: the **RulebaseService** and the **RulebaseResolverPlugin**. Simply put, the difference between the two is that the **RulebaseService** is responsible for reading the contents of the rulebase archives and managing the resulting **InterviewRulebases**, whereas responsibility for providing access to the actual archives is delegated to the **RulebaseResolverPlugin**.

More specifically the **RulebaseService** is responsible for:

- Creating and caching the available **InterviewRulebase** objects.
- Providing access to the list of available rulebases for this instance of the **InterviewEngine**.
- Applying updates to the rulebase cache.

The **RulebaseResolverPlugin** is responsible for:

- Determining where rulebases and modules are loaded from.
- Providing access to the rulebase and module archive streams.
- Notifying the rulebases services of any updates to be made to the rulebase cache.

The **RulebaseResolverPlugin** is created and initialized when the rulebase is created, and like all other Interview Engine plugins, there can only be one per instance of the engine.

Determining which style of RulebaseResolverPlugin to implement

When implementing a custom **RulebaseResolverPlugin**, the implementer must elect between one of the two following styles:

- The Push Method: where rulebase (and associated modules) are "pushed" into the **RulebaseService**. Further updates may be dynamically pushed into the **RulebaseService** at the plugins discretion without the need to restart the **InterviewEngine**.
- The Pull Method: where rulebases (and associated modules) are not loaded until specifically requested (that is, "pulled"). Rulebases loaded in this manner cannot be subsequently updated or removed from the cache.

It is recommended that **RulebaseResolverPlugin** implementations use the Push Method wherever possible. The difference between these two methods of loading rulebases are discussed in more detail below.

The Push Method

Under the Push Method, the plugin pushes the set of rulebases to be loaded as a **RulebaseChangeSet**, using the **RulebaseService.applyChangeSet(...)** method. Generally, the initial set of rulebases to be loaded should be pushed when the plugin's **initialise()** method is called.

When calling **applyChangeSet(...)** the provided change set will be applied on a best efforts basis. This method will return with a list of **RulebaseChangeSetError** to indicate any errors (as well as any existing rulebases that were unloaded as a result) that may have occurred while attempting to apply the change set. Furthermore, any changes provided in a given **RulebaseChangeSet** are considered complete. This means that where a rulebase is dependant on one or more modules, both the rulebase and all of its associated module archives must be provided in the same change set for the rulebase to be able to load correctly.

Source code

To view the source code for the **RulebaseResolver** sample, refer to [examples\interview-engine\rulebase-resolver](#) in the Java runtime zip file.

Dynamically updating rulebases (hotswapping)

Another advantage of using the Push Method is that it allows the rulebase cache to be dynamically updated after it is initialized, allowing rulebases modules to be added, removed or updated. Making such changes is simply a matter of making subsequent calls to **RulebaseService.applyChangeSet(...)** with a **RulebaseChangeSet** that contains the set of changes to be applied.

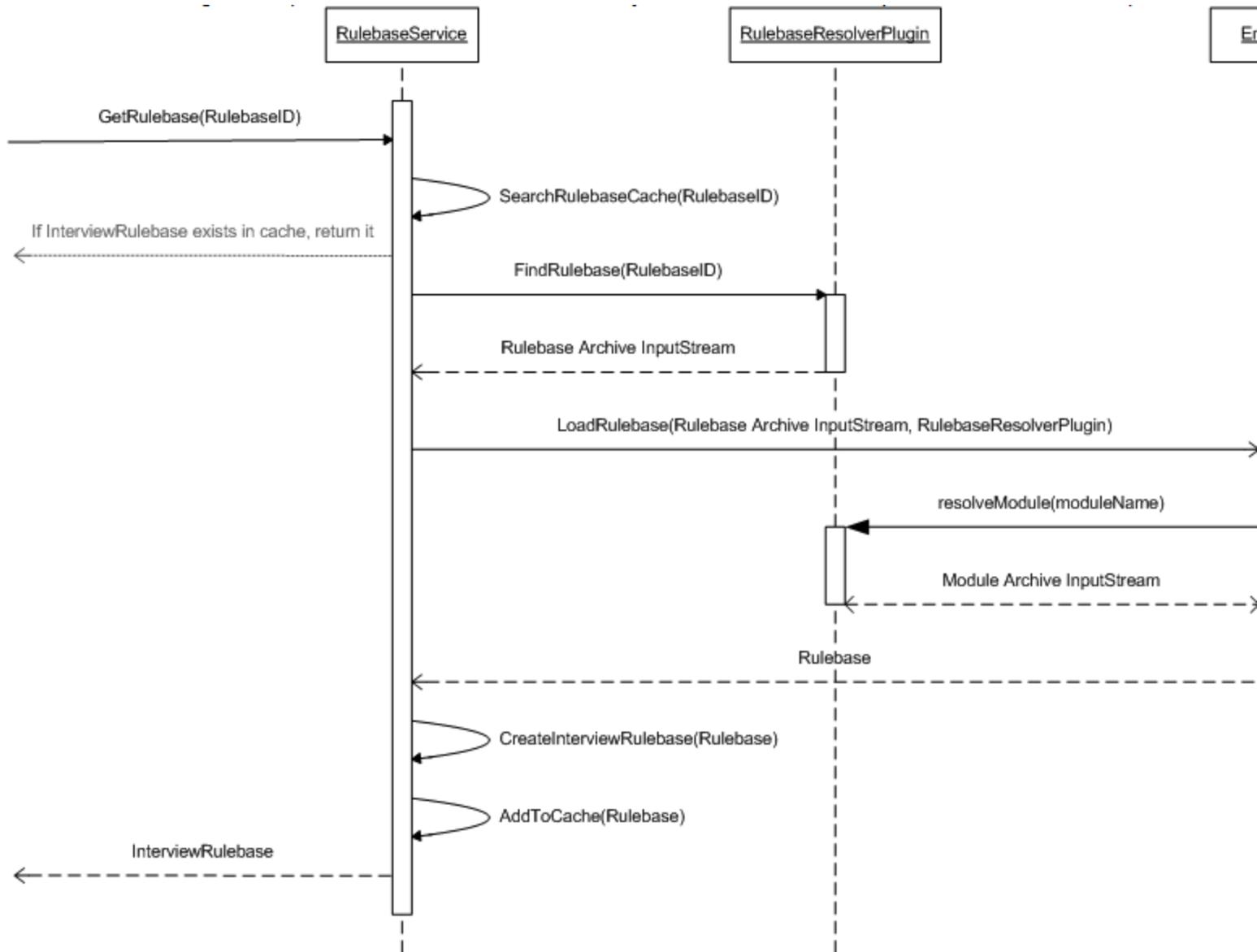
There are some other additional things to be aware of when implementing a **RulebaseResolverPlugin** that provides dynamic updating capability:

- When updating an existing rulebase, it is the responsibility of the **RulebaseResolverPlugin** implementation to ensure that all applicable updates are provided in the same change set. For example, if a rulebase R relies on two modules A and B, both of which have been updated, then both those modules must be supplied in the same change set for the changes for R to be effected correctly.
- Modules can be updated independently of anything that may rely on them, however, updating a module will cause every rulebases that relies on it to be re-loaded. Similarly removing a module will cause all rulebases that depend on it to be removed.
- If an error occurs attempting to update a rulebase or module archive, any existing rulebases in the cache that relate to that archive will be automatically unloaded. The **RulebaseChangeSetError** will indicate which rulebases were unloaded for a given error.

The Pull Method

The Pull Method of loading rulebases is designed for situations where it is not possible to load a rulebase or module archive without being specifically told to which resource it needs to load. For example, a specific implementation stores their rulebase and/or module archives in a data store that only allows them to request a specific resource but provides it with no way to list all the available resources.

Under this method, rulebases are loaded on demand ("pulled") when the Interview Service gets a request for a rulebase that is not already stored in the cache. The process flow for such a request is:



Implementing a Pull style **RulebaseResolverPlugin** is a matter of implementing the **findRulebase(...)** and **resolveModule(...)** methods to return the corresponding rulebase and module archive InputStream.

There are a few disadvantages to implementing this style of resolver:

- Once a rulebase or module is loaded it cannot be dynamically updated or removed.
- Client applications cannot provide a complete list of available rulebases. Instead, rulebases will need to be loaded manually via the **LoadRulebase** action for the Determinations Server or through the **StartSession** URL for Web Determinations.

Source code

To view the source code for the **RulebaseResolver** sample, refer to [examples\interview-engine\rulebase-resolver](#) in the Java runtime zip file.

Example

See [Rulebase Resolver sample code \(DerbyRulebaseService\)](#) for a worked example of implementing a rulebase resolver plugin.

Data Adaptor Plugin overview

The Data Adaptor is an Engine Session plugin to the Web Determinations Interview Engine which allows the current Web Determinations session to use an arbitrary data source for loading and saving session data.

- Web Determinations session data can be saved to any data sources that a Java/.NET app can connect to. This ranges from simple file systems to databases, and also large applications that expose their data source via API's.
- A new Web Determinations session can be created and pre-seeded with rule data from data sources.
- A user's current Web Determinations session can be persisted to data sources. The user is able to specify the Save ID, or save to the current Save ID.

The Web Determinations server only loads one Data Adaptor plugin when a Web Determinations session is started. The Web Determinations server ships with a Data Adaptor plugin, called XDSDataAdaptor. By default this plugin is loaded unless another Data Adaptor plugin has been registered in the current Web Determinations session, in which case the new (custom) Data Adaptor plugin will be loaded.

The Data Adaptor is a plugin, therefore it follows plugin implementations and its benefits/restrictions. For more information about plugins, see [Introduction to plugins](#). For more information about Interview Engine, see [Understand the Interview Engine](#).

The XDS Data Adaptor loads and saves data with the .xds format. This is useful for testing purposes as a Web Determinations session can load data that was saved from, for example, a Debugger session, and likewise a Debugger session can load data from a Web Determinations session.

The examples are for Data Adaptor plugins based on Java programming, but the concepts apply to both Java and .NET.

For more information about implementing the Data Adaptor to common integration scenarios, please see [Data Adaptor - common scenarios](#).

Go to:

[Data Adaptor and the Web Determination architecture](#)

[Developing a Data Adaptor plugin for your specific project/datasource](#)

[General Data Adaptor functionality and design concepts](#)

[Miscellaneous notes](#)

See also:

[Introduction to plugins](#)

[Plugin loading, invocation and discovery](#)

[Plugins - general technical information](#)

[Create a plugin](#)

[Create a Data Adaptor](#)

[Data Adaptor - common scenarios](#)

[Data Adaptor - pseudo code](#)

[Data Adaptor - sample code \(DerbyDataAdaptor\)](#)

[Data Adaptor - sample code \(Autosave with Derby\)](#)

Data Adaptor and the Web Determination architecture

This section details how the Data Adaptor fits into the Web Determinations architecture, and how to use it in the Web Determinations environment.

Go to:

[How Web Determinations uses a Data Adaptor plugin by default](#)

[Data Adaptor plugin and other Web Determinations plugins](#)

[Error Handling](#)

[Data Adaptor Class Methods](#)

How Web Determinations uses a Data Adaptor plugin by default

A Data Adaptor plugin is used whenever the user attempts to save or load Web Determinations session data. By default, the user saves/loads by clicking on the default '**Save/Load/Save As**' buttons during a Web Determinations session.

When the user loads data using the default method, the following occurs:

1. A list of available case ID's to load is displayed to the user.
2. The user selects the caseID to load from a list.
3. Web Determinations server retrieves the Data Adaptor registered to the current Web Determinations interview.
4. Web Determinations Server calls the Data Adaptor **load()**, passing the caseID in together with the current Interview's rule-base.
5. Data Adaptor **load()** returns an object that contains session data back to Web Determinations server.
6. Web Determinations server starts a new session and loads the returned data into it.

When the user saves using the default method, the following occurs:

1. The user clicks on either **Save** or **Save As**.
 - i. If the user chooses to **Save** and there is no CaseID assigned to the current session, the user specifies the CaseID using the *Save As* screen. Otherwise the current session Case ID is used.
 - ii. If the user chooses to **Save As**, the user specifies the CaseID to use for the save in the *Save As* screen.
2. Web Determinations server retrieves the Data Adaptor registered to the current Web Determinations interview.
3. The CaseID from Step 1a/b is used by Web Determinations server to call the Data Adaptor **save()**. The Web Determinations server also passes in the Web Determinations interview session data.
4. The Data Adaptor performs the save onto the datasource, and returns a Case ID.
5. The Web Determinations server uses the Case ID returned and assigns it onto the current Web Determinations interview session.

Loading by URL

Web Determinations also allows a case to be loaded automatically via a URL template. Using the URL skips the step where the user has to choose which Case ID to load. Instead the user starts the Web Determinations interview with the data loaded already. This is useful for integrating Web Determinations with other web applications.

`http://<webdeterminations_
webapp>/startsession/<rulebase>/<locale>/<goalID>?caseID=<caseID#>&user=guest`

- `webdeterminations_webapp` - the address of the Web Determinations web application for example, `localhost/webdeterminations/`.
- `rulebase` - the name of the target rulebase installed in the current Web Determinations (see Web Determinations Server setup to locate the rulebase folder).
- `locale` - the locale for the Interview session, for example, `en-US`, `es-CR` for Spanish - Costa Rica.
- `goalID` - optional, If it is provided, an investigation on the provided goalID is started and thus skips the Summary screen.
 - Goal IDs are structured like, for example, for the Attribute 'eligible' in global, it is 'Attribute~eligible~global~global'.
- `caseID` - the Case ID to be provided by the client system. The client system may already have the data needed to retrieve/construct the Case ID, or the client system may authenticate the user retrieve or generate the Case ID.

Data Adaptor plugin and other Web Determinations plugins

Other ways in which the Data Adaptor is used in Web Determinations is through other [Web Determinations plugins](#). These plugins can call the Data Adaptor's save and load methods, thus allowing loading data into the Web Determinations interview/saving data in the current Web Determinations interview to be controlled by plugins such as Event Handlers. Saving of session data can happen automatically without the user needing to explicitly issue a save command (which would be the most common case when Web Determinations is integrated within a web application/workflow).

Web Determinations plugins can access the Data Adaptor registered to the current Web Determinations session if the plugin has access to the `SessionContext` object or its member the **InterviewSession** object. The Web Determinations plugin can call the Data Adaptor **load()** via the **InterviewSession** object to retrieve **InterviewUserData** based on the provided caseID, or call the Data Adaptor **save()** to persist the current **InterviewSession**; for example:

Calling the Data Adaptor load

```
InterviewUserData InterviewUserData = sessionContext.getInterviewSession().getDataAdaptor().load(token, caseIDToLoad, sessionContext.getInterviewSession().getRulebase());
```

Calling the Data Adaptor save

```
interviewSession.getDataAdaptor().save(token, caseIDToSave, interviewSession);
```

Notes

- Web Determinations extensions should start a new session before loading the data into the Web Determinations session (this is the same as how the default load process works). Loading session data into an existing Web Determinations session that already has data will result in abnormal behavior.
- The Data Adaptor **load()** and **save()** only have rulebase model data (for load) and rulebase model + instance data (for save) passed in. This means that Web Determinations extensions that call the Data Adaptor **load()** and **save()** cannot pass

in additional data, it can only trigger the Data Adaptor to load or save. Therefore the Data Adaptor needs to be designed in such a way that it can correctly load data based on the rulebase model data available, or correctly save data based on rulebase model plus instance data.

- The Web Determinations extension calling the Data Adaptor must provide the Case ID for **load()**, and also for **save()** if the Data Adaptor is not configured to provide Case ID's. Keep in mind that there times where the **save()** may be called by a Web Determinations Extension and the Case ID has not been provided yet (for example, the Web Determinations interview was not started with a Data Adaptor **load()**)

Error handling

Data Adaptor plugins can handle errors that occur within it (for example, SQLExceptions), and then throw an exception for the calling method to handle (or in Java, an unchecked exception).

When the Data Adaptor load method is called and an error occurs:

- A specific 'Failed to Load' screen is displayed by Web Determinations, together with the attempted Case ID if the Data Adaptor **load()** throws an exception.

When the Data Adaptor save method is called:

- A specific 'Failed to Save' screen is displayed by Web Determinations if the Case ID returned by **save()** is null or empty.
- Generic 'error' page if the **save()** throws an exception.

Data Adaptor class methods

A Data Adaptor plugin implements the **DataAdaptorPlugin** interface. The **DataAdaptorPlugin** interface in turn extends the **DataAdaptor** interface and also the **InterviewSessionPlugin** interface.

The **DataAdaptor** interface requires the following when implemented:

Method Signature	Description
InterviewUserData load(SecurityToken token, String caseID, InterviewRulebase rulebase)	<ul style="list-style-type: none"> • The SecurityToken can be used by the plugin developer for authorization. • The caseID is the ID used by the developer to retrieve the Web Determinations session data to load. • The InterviewRulebase allows the plugin developer to access the session's rulebase, and its contents; that is, entities, attributes, relationships, screens, flows, and so on. • an object of InterviewUserData is returned, which in turn is loaded by the calling process.
String save(SecurityToken token, String caseID, InterviewSession session)	<ul style="list-style-type: none"> • The SecurityToken can be used by the plugin developer for authorization. • The caseID is used as the ID to save the Web Determinations Session data. This is optional if the dataAdaptor is generating it's own ID during the save. • The InterviewSession contains the Web Determinations session, which has the data to save and also data about the current session; for example, the rulebase being used. • The caseID used by the save() is returned.
boolean dataAd-	<ul style="list-style-type: none"> • This is set by the plugin developer to true if the dataAdaptor will provide the

<p>apTORProvidesCaseID</p>	<p>caseID (usually for first-time save). When this is true, the Save As button is not displayed, therefore the user can only initiate Save action.</p> <ul style="list-style-type: none"> • This is used for setups where the datasource of the Data Adaptor will generate the CaseID; for example, SQL insert into a table with autoincrement primary key. • For default setups where the Web Determinations server calls the Data Adaptor save() (and not another Web Determinations extension), the CaseID passed will be null/empty since the user was not able to provide the Case ID.
<p>String[] listCases(SecurityToken token, InterviewRulebase rulebase)</p>	<ul style="list-style-type: none"> • The SecurityToken can be used by the plugin developer for authorization. • The InterviewRulebase allows the plugin developer to access the session's rulebase, and its contents; that is, entities, attributes, relationships, screens, flows, and so on. • The String array returned is a list of Case ID's to display to the user as options of cases available to be loaded.

For requirements from the plugin interface, see [Plugin loading, invocation and discovery](#) See the sample code section for practical examples

Developing a Data Adaptor plugin for your specific project/datasource

This section explains the various approaches on designing and developing a Data Adaptor plugin for a specific project/rulebase/datasource.

Go to:

[Factors to consider when designing a Data Adaptor plugin](#)

[About the data passed into the Data Adaptor methods](#)

[How to use the data passed into the Data Adaptor methods](#)

[How to build the InterviewUserData object in Data Adaptor load\(\)](#)

Factors to consider when designing a Data Adaptor plugin

When designing a Data Adaptor, the following factors needs to be analyzed

- datasource type and schema/setup.
- rulebase complexity and rule data required to be loaded/saved.
- the need to support flexibility/maintainability for both a changing rulebase and datasource schema.

Essentially, the Data Adaptor needs to be able to:

- for loading - retrieve data from the datasource so it can be transformed into Web Determinations session instance data, and loaded onto the session.
- for saving - manipulate the Web Determinations session instance data so it can be persisted into the datasource.

The design of a Data Adaptor plugin varies because how it will map the datasource data to its equivalent Web Determinations session data depends on the datasource and rulebase setup/complexity. Also the maintainability aspect of the Data Adaptor is

important - because for both rulebase and datasource change it is important to design the Data Adaptor to match the anticipated changes both will have in the future.

For example - the Data Adaptor might have to deal with a very complex rulebase and database system that is anticipated to change. It will probably be designed with an architecture, using other tools/libraries, etc to ensure that the it can meet various requirements such as authentication, datasource transaction, errors, as well as non-functional such as performance. On the other hand the rulebase and datasource might be small and simple in which case the Data Adaptor simply needs to be a small implementation, with a lot of hard coded data to access the datasource.

About the data passed into the Data Adaptor methods

It is important to understand the available data that is passed to the Data Adaptor **load()** and **save()**. The input arguments objects that are of interest for both are the **InterviewSession** and **InterviewRulebase** objects. The SecurityToken and caseID are both sufficiently explained in the previous section *Programming Constructs*.

But first, it is important to understand the two main types of data that exists during Determination Engine runtime:

- The rulebase model data- The 'metadata'. This is the entities, attributes, and relationships created when a rule author creates a rulebase in Oracle Policy Modeling. It provides the data structure that allows the Determinations Engine to inference, get next question, and so on.
- The rulebase instance data - this is the actual data provided by the user during a Web Determinations session, and maps to items in the rulebase data model.

The following table shows the difference between the Model and Instance:

Rulebase Model data	Rulebase Instance data
the child (entity)	child1 (identifier value = Wayne)
the school (entity)	school2 (identifier value = Summer High)
the child's name (attribute)	Wayne
the child's school (relationship)	<child1 connects to school2>

The **InterviewSession** object contains many members, but with relation to the Data Adaptor it's useful members are:

- **InterviewRulebase** - this object provides access to the rulebase model data.
- **(Rule)Session** - this object provides access to the rulebase instance data.

How to use the data passed into the Data Adaptor methods

The Data Adaptor **load()** receives the **InterviewRulebase**, therefore it only has access to the rulebase model data.

- the model data is used together with the data retrieved from the data source to construct the **InterviewUserData** object to be loaded into the Web Determinations session.
- the model data can be used to drive the retrieval process from the datasource via mapping/configuration data for data integrity and better maintainability and flexibility of the Data Adaptor.

Sample code to demonstrate access of the InterviewRulebase model data

```

public InterviewUserData load(SecurityToken token, String caseID,
                             InterviewRulebase rulebase)
{
    ...
    //Go through the rulebase entities, and get all instances of each entity in the database together with
    //attributes
    Rulebase rulebaseDE = rulebase.getRulebase();
    //note you need to get the Determination Engine rulebase first
    for(Object entObj : rulebaseDE.getEntities())
    {
        //The code below accesses an Entity in the List of Entities, its Attributes, and Relationships
        Entity entityModel = (Entity)entObj;
        List<Attribute> entityAttributes = entityModel.getAttributes();
        List<Relationship> entityRelationships = entityModel.getRelationships();
    }
    ...
}

```

- More sample code at [Data Adaptor - Pseudo Code](#), [Data Adaptor - Sample Code \(Autosave with Derby\)](#) and [Data Adaptor - Sample Code \(DerbyDataAdaptor\)](#)

The Data Adaptor **save()** receives the **InterviewSession** object, so it has access to both the rulebase model and instance data. Note that it does not receive the Web Determinations session user data in a **InterviewUserData** object - the return object of the Data Adaptor **load()**.

- the model data must be used to extract the instance data from the **(Rule)Session**.
- the model data is used to persist the extracted instance data to the data source.
- the model data can also be used with mapping/configuration data for better Data Adaptor maintainability and flexibility.

Sample code to demonstrate usage of the InterviewSession model data

```

public String save(SecurityToken token, String caseID,
                  InterviewSession iSession)
{
    ...
    Session sessionDE = session.getRuleSession(); //note you need to get the Determination Engine session
    Rulebase rulebaseDE = sessionDE.getRulebase(); //this gets the Determination Engine rulebase
    ...

    for(Object entObj : rulebaseDE.getEntities())
    {
        //Get all instances of the current entity from the session and loop through it
        Entity currentEntity = (Entity)entObj;
        Collection<EntityInstance> currentEntityInstances = currentEntity.getEntityInstances(sessionDE);
        //note retrieval of instance using the Entity model object
        Iterator iterateCurrentEntityInstances = currentEntityInstances.iterator();
    }
}

```

```

while(iterateCurrentEntityInstances.hasNext())
{
    EntityInstance currEntInstance = (EntityInstance)iterateCurrentEntityInstances.next();

    //Access attribute instance values of the current Entity Instance
    for(Object attrObj : currentEntity.getAttributes())
    {
        Attribute currAttr = (Attribute)attrObj;

        //Attributes in an entity instance is accessed through the Attribute object and passing the Entity
        //instance
        Object attributeValue = currAttr.getValue(currEntInstance); //note retrieval of instance using the
        Attribute model object
    }
    //Get source or target instance for each relationship of the current Entity model by passing the
    //instance to the Relationship model data
    for(Object relObj : currentEntity.getRelationships())
    {
        Relationship currRel = (Relationship)relObj;
        //note retrieval of relationship source/targets using the Relationship model object
        if(currRel.isSingleTarget) EntityInstance targetInstance = currRel.getTarget(currEntInstance);
        if(currRel.isSingleSource) EntityInstance sourceInstance = currRel.getSource(currEntInstance);
    }
}
}
...
}

```

- More sample code at [Data Adaptor - Pseudo Code](#), [Data Adaptor - Sample Code \(Autosave with Derby\)](#) and [Data Adaptor - Sample Code \(DerbyDataAdaptor\)](#)

How to build the InterviewUserData object in Data Adaptor load()

The **InterviewUserData** is much simpler than the **InterviewSession** or **InterviewRulebase**. The **InterviewUserData** has a List of Entities and Relationship instances.

- The **InterviewUserData** is made up of **InterviewEntityInstance**, which in turn is made up of relationships and attributes.
- To add an Entity instance to the InterviewUserData, call the method **addInstance(InterviewEntityInstance entInstance)** of the InterviewUserData
- To add a relationship to an Entity instance, call the **addRelationship(Relationship rel, InterviewEntityInstance target, byte status)** on the **InterviewEntityInstance** object.
- An Entity instance stores its attributes in a map as key/value pairs, where the key is the Attribute's public name. Adding an attribute to the Entity instance is as simple as calling its **setValue(String key, Object value)** method.

- Note that for the Entities and Relationships to be added into the session properly, they must be set to the correct status.
 - For relationships, it is done when calling the **addRelationship()** by passing the 'Add' status; for example, `entInstance.addRelationship(relationship, targetEntInstance, InterviewInstanceStatus.ADD);`
 - For entities, the entity instance's status is set to 'Add'; for example, `entInstance.setStatus(InterviewInstanceStatus.ADD);`
- Note that when adding a relationship to an Entity Instance; for example, `entInstance.addRelationship(relationship, targetEntInstance, InterviewInstanceStatus.ADD);`
 - the reverse relationship is also created for the target instance - **targetEntInstance**.

Sample code to demonstrate usage of the InterviewUserData

```

public InterviewuserData load(SecurityToken token, String caseID,
                             InterviewRulebase rulebase)
{
    ...
    InterviewUserData returnUserData = new InterviewUserData();

    //Go through the Entity model, and create an InterviewEntityInstance of each, add some
    //attributes and relationship, and add to user data
    Rulebase rulebaseDE = rulebase.getRulebase();
    //note you need to get the Determination Engine rulebase first
    int count = 1;
    for(Object entObj : rulebaseDE.getEntities())
    {
        Entity entityModel = (Entity)entObj;

        //create an Entity instance and add it to the user data
        InterviewEntityInstance entInt = new InterviewEntityInstance(entityModel.getName(),
            count);
        count++;
        entInt.setStatus(InterviewInstanceStatus.ADD);
        returnUserData.addInstance(entInt);

        //add an attribute to the entity instance
        entInt.setValue("attributePublicID", attributeValue);
    }

    //add a relationship - lets assume there's an entity called 'child' and 'school', with a 1-m relationship
    InterviewEntityInstance child = new InterviewEntityInstance("child", 100);
    InterviewEntityInstance school = new InterviewEntityInstance("school", 2);
    Relationship childSchoolRel = rulebaseDE.getRelationship("the child\'s school", "child");
    child.addRelationship(childSchoolRel, school, InterviewInstanceStatus.ADD);
    //Note that the reverse relationship e.g. the school's student is created/added to 'school'

```

```
...  
}
```

More sample code at [Data Adaptor - Pseudo Code](#), [Data Adapter - Sample Code \(Autosave with Derby\)](#) and [Data Adaptor - Sample Code \(DerbyDataAdapter\)](#)

General Data Adaptor functionality and design concepts

Authentication

A SecurityToken is provided to the **save()**, **load()**, and **listCases()** for the Data Adaptor to use if it needs to authenticate the user; for example, the Data Adaptor may need to use the token to authenticate and connect to the datasource as well.

It is optional to use the SecurityToken.

Identifying Entity instances from each other

There are a couple of uses for instance Identifiers in the Data Adaptor:

- managing the instance ID of the Entity instances being created (for load) or read (for save) to be able to create or persist relationships properly.
- in cases where the instance ID is required to check if it exists in the datasource for a Data Adaptor **save()**.
- use the instance ID for complex Data Adaptor designs/architectures such as mapping the rulebase model data to the database schema.

The **instanceName** of an Entity instance object (which is **InterviewEntityInstance** or **EntityInstance**) is its unique identifier amongst all other Entity instances in a Web Determinations session; for example, a School entity instance can have an instanceName of 's1'. Therefore no other instance, regardless of whether it is a School entity or not, may have an **instanceName** of 's1'. The **instanceName** can be anything from a simple number to composite text generated from attributes of the instance; for example, `instanceName = firstName + lastName + dob.toString()`.

The **instanceName** of Entity instances loaded into the Web Determinations session is maintained - so any instanceNames from the Data Adaptor **load()** will still be intact for those same instances that are processed in Data Adaptor **save()**

For complicated setups, the Data Adaptor developer can implement their own method of ID'ing instances. The Data Adaptor can implement a complex/composite **instanceName** property generated from various data values of the instance or the Data Adaptor developer can create an Identifier object that can assist in ID management of instances.

Ways to make the Data Adaptor flexible to rulebase and datasource changes

Making the Data Adaptor flexible to rulebase and datasource changes varies wildly from each implementation, but there are approaches and tools that should work for most setups. One thing to note is that datasource, especially legacy database systems does not change often due to the number of other systems that depend on it. Rulebase changes can vary from project to project.

Using a tool that simplifies the Data Adaptor's data collection from the datasource will be useful for both managing datasource changes and also rulebase-datasource mapping; for example, Object-Relational Mapping for relational databases.

Ideally the Data Adaptor load and save can be driven via a rulebase-datasource mapping as opposed to hard coding rulebase and datasource mapping within the Data Adaptor logic. The mapping can be separate for Data Adaptor load and save.

- the mapping can contain rulebase entity/attribute/relationships that should be retrieved, and how to retrieve it in the data-source
- having a mapping as a configuration means it can also be easily updated, and integrated with rule authoring workflow for automatic update

Other Data Adaptor load() specific concepts

The following is a list of things to note when developing the Data Adaptor **load()**:

- check that the caseID exists in the datasource.
- the Global instance is automatically created when an instance of InterviewUserData is created.
 - retrieved via **InterviewUserData.getGlobalInstance()**;
 - also the Global instance doesn't need its **InterviewInstanceStatus** set to ADD.
- it's a good idea to drive the data retrieval process from the data source via the rulebase model data; that is, loop through the model data (entities, its attributes, relationships), through the loop if an expected model comes up then perform logic.
 - note that for quick setups (for example, prototypes and proof of concepts), it's possible to just hard code rulebase and datasource references so that it simply retrieves data, builds the user data, and returns it.

Relationships

Entity Relationships is the more complex part of loading data into a Web Determinations session.

- adding a relationship to an entity instance is done via the call: **sourceEntityInstance.addRelationship(Relationship rel, InterviewEntityInstance target, InterviewEntityStatus byte)**.
- adding the relationship is usually done to the **source** entity instance, and doing so automatically creates the reverse relationship for the target.
- relationships between the Global entity and other entities must always be created. Otherwise entities not attached to the Global will not appear in the Web Determinations session.
 - relationships between entities can be left out to give the user a pre-set choice when linking them (with the help of screens and screenflow); for example, say a child is connected to a school entity via an m-1 a child's school.
 - child and school entities can be loaded into the session, but not the relationships between them
 - during the interview, the user will be able to select which school each child goes to if the specified school was loaded in session, rather than having to create instances of the school first.
- both source and target entity instances and the Relationship object (model data of the relationship) are needed to add a relationship.
- as mentioned before, the status needs to be **InterviewInstanceStatus.ADD** so it is added to the session
- when creating instance relationships, it might be handy to create a list that contains five datamembers for each item - source entityID, source instanceName, Relationship object or name, target entityID and target instanceName.
 - this list can be populated while retrieving entity instances and their attributes.
 - the list can be processed after all instance objects have been created, to create the relationships and link the source/target entities.

Other Data Adaptor save() specific concepts

The following is a list of handy notes for reference when developing the Data Adaptor save()

- The instance identifier is important here.
 - This is important for implementations where the Data Adaptor **save()** needs the identifier to know where to save the data (for example, saving the data to a specific table row).
 - Therefore it is imperative that the Data Adaptor **load()** correctly assigns instanceNames; for example, Columns 1-5 from table 'User', userID '1' is loaded, and inferred data is saved on columns 6-10 of the same userID '1' in the table 'User'.
- Its more convenient to process the relationships first, then process entities and attributes.
 - Go through all relationship instances and store it in a List object, then while going through entities and persisting them any relationship data that needs to be persisted can be done together with the instance.
 - Opposite to Data Adaptor **load()**, where the relationship is processed last.
 - Similar to the Data Adaptor **load()**, a list with 5 datamembers (source ID/name, target ID/name, and Relationship object) or similar list would do.
 - After the list is populated, while saving the entities and attributes the relationship data can also be saved, thus it's only one insert/update SQL instead of two in the case of databases.
- A tool or intermediate architecture/API between the Data Adaptor and the datasource would be handy here, to make data persistence easier to manage - especially with changing rulebase and datasource; for example, for SQL databases, an ORM would reduce the amount of maintenance work on SQL query building.

Miscellaneous Notes

- If multiple Data Adaptor plugins have been registered for a Web Determinations session, there is no way to specify which one will be used by the Interview Engine. It is up to the developer to ensure that only one Data Adaptor is loaded for a Web Determinations session.
- A developer can deploy multiple Data Adaptor's into a Web Determinations server, and can use the Data Adaptor's plugin method implementation **getInstance()** to ensure that only one Data Adaptor is registered for a Web Determinations session.
- Entity instances of the target for an m-m relationship without creating the relationship between the source/target allows the source to select a predefined list of the target -> the effect is a poor man's List Provider.
- Link template to load a Case ID - <http://<webserver address>/web-determinations/startsession/<rulebase name>/<- locale ID>?caseID=<caseID#>&user=guest>.
- Web Determinations Extensions can only call Data Adaptor load and save, it cannot pass in additional data.

Data Adaptor - common scenarios

The following are common implementation scenarios that use the Data Adaptor, listed from basic to more complex scenarios. Note that some of the scenarios need to use other Web Determinations extensions, and also other Web Determinations extensions; for example, customizing UI, controls, and so on.

These scenarios document only how to implement the Data Adaptor in terms of each scenario.

Go to:

[Pre-seeding a Web Determinations interview with data from an external data source](#)

[Saving data from a Web Determinations interview to an external data source](#)

[Auto-saving data during a Web Determinations interview - using the Data Adaptor and Event Handlers](#)

[Integrating the Web Determinations interview into the client system - using the Data Adaptor to handle data integration](#)

Pre-seeding a Web Determinations interview with data from an external data source

This scenario is useful when:

- Data for base attributes in a Web Determinations interview is already available, and accessible from a datasource; for example:
 - the user has provided data to an existing client system, and that system stores the data in a datasource accessible to Web Determinations.
 - data about the subject matter of the Web Determinations interview is available from a datasource that is accessible to Web Determinations.
- The Web Determinations interview needs to be integrated into a client system, and the client system wants to hand over the user to the Web Determinations interview for determination.
 - it is expected that questions in the Web Determinations interview where the data is already known by the client system should be pre-populated.
- The Web Determinations interview is expected to allow the user to stop the interview, and continue later.

Pre-seeding a Web Determinations interview with existing data has the following advantages:

- better user experience - the user does not need to re-enter the same data that has already been provided to the client's other system in the past.
- ability to provide data that the user may not know - this is useful for Web Determinations interviews where some of the base attributes of a goal is not known by the user, but is needed for the determination of a goal.

Example scenario:

A client system is a case management system used for customer support, and needs determinations functionality. The user of the system is a customer support representative. When the user is in the case management system and needs to access Web Determinations for determinations, the Web Determinations interview that is started will have user data pre-filled, as well as other data required by the interview for determinations. Thus, the user does not need to manually enter existing data, and the Web Determinations interview process is streamlined, efficient, and will have less chances of input error from the user.

How to implement pre-seeding functionality using Data Adaptor plugin.

The general steps to enable pre-seeding are:

1. Use or create a Data Adaptor that has a working **load()** action for the Web Determinations interview (note - a Web Determinations interview has specific rulebase and locale).
2. The Data Adaptor is loaded onto the Web Determinations web application.
3. The user is directed to the Web Determinations interview via a constructed link that defines the Case ID to be used as the key to retrieve data.
4. The Data Adaptor **load()** is called, with the Case ID passed in.
5. The Data Adaptor uses the Case ID to retrieve the necessary data from the external data source, and hands it over to the Web Determinations interview for data pre-seeding.
6. The user starts the Web Determinations interview with pre-seeded data.

The URL that will direct the user to the pre-seeded Web Determinations interview is through a URL constructed by the client's system.

For developing Data Adaptors, see [Data Adaptor plugin overview](#).

Saving data from a Web Determinations interview to an external data source

This scenario covers cases where data in a Web Determinations interview session needs to be saved to an external data source.

- Useful for saving inferred data (the main power of Oracle Policy Automation).
- Also can be used together with auto-saving to enhance user experience with Web Determinations interview - they can save provided data and continue later, don't need to complete the interview in one go.

The general steps to enable saving data from a Web Determinations interview are:

1. User or create a Data Adaptor that has a working **save()** action for the Web Determinations interview (note - a Web Determinations interview has specific rulebase and locale).
2. The Data Adaptor is loaded onto the Web Determinations web application.
3. When the save action is triggered in the Web Determinations interview, it calls the Data Adaptor **save()**.
4. The Data Adaptor reads the Web Determinations interview session data and saves it into the external data source.
5. Other systems can now access the data entered or inferred during the Web Determinations interview.

The steps above use default functionality for the Data Adaptor save method. As long as the Data Adaptor plugin is loaded for the Web Determinations interview, it will be used for any save action within that Web Determinations interview.

The action for saving data should also be determined; the following are two common ways of calling the **Save** action in a Web Determinations interview:

- Most likely the data is automatically saved for the user by the Web Determinations session. Automatic saving of data requires Event handlers that call the Data Adaptor to save the current data.
- The user is also able to save by clicking on the default **Save/Save As** command available in a Web Determinations interview session. This also calls the Data Adaptor to save the current data.

For developing Data Adaptors, see [Data Adaptor plugin overview](#)

This scenario also has variations/extensions, which are:

- Auto-saving of Web Determinations interview data when a certain event occurs or requirements are met.
- Loading base instance data and saving inferred instance data

Note that when calling the Data Adaptor save, only the Case ID and interview session data are passed in.

Auto-saving data during a Web Determinations interview - using the Data Adaptor and Event Handlers

Auto-saving data during a Web Determinations interview can be driven by Event Handler plugins. For more information about Event Handlers, see [Events and Event Handlers](#).

Auto-saving will use the 'save' capability of the loaded Data Adaptor, triggered by an Event Handler. To persist Web Determinations interview data into an external datasource see "Saving data from a Web Determinations Interview to an external data source" (above).

The following are common triggers for auto-save, and the Event to use:

- when the user completes a screen (**OnGetScreenEvent**).
- when a goal is reached (**OnInvestigationEndEvent** plus checking the Session data that the target goal attribute value has been provided).
- when certain attributes are inferred (**OnGetScreenEvent** plus checking that the particular attribute has a value that is not null, unknown, uncertain).

More complex scenarios can combine the above so that, for example, auto-save per screen is performed as well as being performed again when the goal is reached.

The general steps to setup Auto-saving of data during a Web Determinations Interview to an external datasource (that is, using Data Adaptor):

1. User or create a Data Adaptor that has a working **save()** action for the Web Determinations interview (note - a Web Determinations interview has specific rulebase and locale).
2. Determine which Event/s need to be used to trigger the Data Adaptor **save()**.
3. Create Event Handler/s that triggers the Data Adaptor **save()**.
4. The Data Adaptor together with the Event Handler/s are loaded onto the Web Determinations web application.
5. When a user goes through a Web Determinations interview, event/s for the autosave Event Handler/s may be triggered.
6. When an autosave Event Handler is triggered, it will process the Event arguments and call the Data Adaptor **save()** if conditions are met.
7. When the Data Adaptor **save()** is called, it will save data from the Web Determinations interview.

For Autosave sample code, see [Data Adaptor - sample code \(Autosave with Derby\)](#)

Notes

- Because Case ID and Interview Session data are the only data passed through to the Data Adaptor, design of the Event Handler and Data Adaptor must take this into account. The Data Adaptor must be able to operate based on information available

in the Web Determinations session data passed - that is, the Event Handler cannot tell the Data Adaptor if the save is for a completed screen (autosave per screen), or a completed goal.

- When analyzing which Events to use, check both sender Object and the event args. Both expose data that can be used to fine-tune when the Data Adaptor save is triggered
- The Case ID that the Event Handler needs to pass to the Data Adaptor **save()** is in the **SessionContext** object. Therefore ensure that the Event to trigger the autosave has access to the **SessionContext** object (either the sender object or from one of the Event args)
- Ensure that the Event Handler calling the Data Adaptor provides a Case ID if the Data Adaptor is not generating the ID.

Integrating the Web Determinations interview into the client system - using the Data Adaptor to handle data integration

It's very common for Web Determinations to be integrated into the client system, usually as part of a workflow where a Web Determinations interview session is required to inference information from provided data.

The client system usually has existing data about the user, and this existing data can be loaded into the Web Determinations interview session to pre-fill fields. In many cases the client system will integrate to the Web Determinations interview by being able to pre-load that existing data into a Web Determinations interview.

To load data into a Web Determinations interview, a Data Adaptor needs to be created so it can load the data from an external data-source. The URL that the client system uses to forward the user to the Web Determinations interview will contain the Case ID. For more information about loading data via the Data Adaptor see the *Pre-seeding a Web Determinations interview with data from an external data source* section above.

Once the user has provided all the necessary base attribute data to reach a conclusion, some or all of the Web Determinations interview session data usually needs to be passed back to the client system. The most effective way to do this is to persist the data into a datasource first so the client system can access the new data from the datasource.

The persisting or 'saving' of data to a datasource during the Web Determinations interview can be persisted via the Data Adaptor. The action of saving the data can be manual (user-triggered) or automatic (event-driven).

For more information about saving Web Determinations interview session data into a datasource, see [Saving data from a Web Determinations interview to an external data source](#).

Create a Data Adaptor

The following information describes the steps specific to creating a Data Adaptor plugin. For steps on creating and installing a Plugin, see [Create a plugin](#). Although it focuses on Java development, the same steps can be followed for .Net (with slight implementation differences).

Assets and information needed

The following assets are required to develop the Data Adaptor Plugin, or information that will be referred to in the Steps section.

Web Determinations library jars

- *web-determinations.jar* - this can be found in the Web Determinations folder 'lib' folder, for example: `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\lib`.
- *determinations-interview-engine.jar* - this can also be found in the Web Determinations folder 'lib' folder, for example: `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\lib`.

Java IDE - this is the Java development application you are using. While this is optional, it will make it much easier when referring to objects in the *web-determinations.jar* and *determinations-interview-engine.jar* files.

Data Adaptor business case - the reason why the Data Adaptor needs to be implemented for this specific project. There are many reasons, mostly related to integrating data. For more information, see [Data Adaptor - common scenarios](#).

Datasource - this is the datasource (or datasources) that this Data Adaptor plugin will connect to, to load data, save data, or both. It can be an SQL database, or an API that connects to a datasource; for example, Siebel.

Datasource library jars - the Datasource may have Java jar library files that is required for the Data Adaptor.

Datasource connection details - the Data Adaptor plugin will need to be able to connect to the datasource, so connection details must be known. This might be a specific server port; for example: `//server:portnumber/databasename`.

Steps

Analysis and design:

Before starting on the development of the plugin, refer to the [Data Adaptor overview](#) topic for an understanding of how the Data Adaptor is used by Web Determinations, and the various ways in which **load()** and **save()** are called.

Once you have familiarized yourself with the Data Adaptor plugin:

1. Determine what rulebase (or rulebases) the Data Adaptor will work for; you can make a Data Adaptor that services all the rulebases and its locales in the Web Determinations server, or you can create the Data Adaptor to only work for a specific Rulebase, or even only for one of its locales.
2. From the Data Adaptor business case, establish:
 - i. What data in the rulebase can or needs to be loaded from the datasource.
 - ii. What data in the rulebase needs to be saved into the datasource.

- iii. The datasource and location of the data to be loaded - or where the data is to be saved.
 - iv. How the save action is to be done during the Web Determinations interview; saved manually saved by the user, or saved automatically? Is the user allowed to specify the Case ID to use for saving, or is it automatically generated?
 - v. How the load action is to be done; will the user be directed to the Web Determinations interview with data pre-loaded, or is the user allowed to specify which data to load?
3. Determine how the Data Adaptor will connect and interact with the datasource, including transaction integrity processes, fallback on error, and so on.
 4. Design the authentication to Data Adaptor methods (**load()**, **save()**, **listCases()**) if needed.
 5. Design how the Data Adaptor will access the data and map it to rulebase attributes for **load()**, and vice versa for **save()**.
 - The complexity of how the necessary data is accessed and mapped to the rulebase, depends on various factors such as complexity of the datasource/rulebase, flexibility and maintainability requirements, and the datasource type/schema.

Development

1. Set up the **Java IDE** so that it has the **Web Determinations library jars** and **Datasource library jars**. This makes it easier to refer to objects for Web Determinations or for the datasource.
2. Start the Data Adaptor plugin class from the following pseudo code - [Data Adaptor - pseudo code](#).
3. Develop the Data Adaptor plugin based on the *Analysis and design* section.

Install and test

For steps on installing and testing the Data Adaptor, see [Create a plugin](#).

Use the default Data Adaptor

Data Adaptors allow Web Determinations interview data to be saved to a datasource, and also to load data into a Web Determinations interview before starting the interview session; more information about Data Adaptors can be found in [Data Adaptor plugin](#).

The default Data Adaptor for Web Determinations allow the user to load and save interview data into an XDS file. When saving the interview data, the user can provide the save name. When loading interview data, a list of the saved interview data for the current rulebase is displayed, listed using the save name.

XDS files are XML files that are used by Oracle Policy Automation technologies to store rulebase model and instance data. The XDS format is a common way for many Oracle Policy Automation applications to export/import rulebase data between each other. It is possible to use the default Data Adaptor to save interview data into an XDS file, and use the XDS file with another Oracle Policy Automation technology that use XDS files as input.

To use the default Data Adaptor make sure no other custom Data Adaptors are available. Custom Data Adaptors take precedence over the default Data Adaptor, so the default Data Adaptor will not be registered in the interview session if another custom Data Adaptor plugin exists and can be registered to the current Web Determinations interview.

Setup

There are no extra setup steps required to setup the default Data Adaptor. But it is expected that there are no customizations that affect:

- The default Data Adaptor being registered as the Data Adaptor to service a Web Determinations interview.
- The **Save (Save As)** and **Load** buttons from being accessible to the user in the Web Determinations interview; for example, customizations to Web Determinations templates or configuration that remove the Save and Load buttons.

The **Save** and **Load** buttons sit at the top right area of the Web Determinations interview screen with the default layout. See the screenshot below:

ORACLE Web Determinations

Data Review

Note: The **Save** button is only displayed when the current Web Determinations Interview session has been saved, or was loaded from a saved Case. See screenshot below.

ORACLE Web Determinations

Summary | Data Review

Using the default Data Adaptor

Saving the Web Determinations interview

Firstly you need to configure the directory to which the data will be saved. This is done by changing the **xds.file.path** property in the **application.properties** file. When doing this, you should note the following:

- The path can either be absolute or relative to the *<webroot>*.
- Ensure that Web Determinations has read/write permissions to the directory.
- On a Java application server that allows the deployment of unexploded war files (for example, WebLogic) the directory must be absolute.

The user can save the interview at any time during the interview; for example, during a Goal investigation, or at the *Summary* screen. The user is returned to the correct screen if saving during a Goal investigation, thus his/her investigation progress is not lost when performing a save.

If the current interview session has not been saved:

1. Click the **Save As** button (top-right area for default Web Determinations layout); the *Save As* screen is displayed.
2. Provide a *Case ID* as the save name; the *Case ID* must be unique against all other *Case IDs* for the current rulebase.
3. Click **OK** to confirm the *Case ID* provided, and to save the data to a *Case*; a *save successful* screen is displayed if the screen succeeds and data of the current interview is now saved into a *Case*.
4. The *save successful* screen will also have a link that resumes the interview and a **Continue** button; click on either the link or **Continue** button to return to the same screen where you initiated the interview save action (for example, if you were in between the investigation of a Goal, you will be returned to the correct screen so that progress is not lost).

If the current interview session has been saved, or the session was loaded from a saved *Case*:

1. Click the **Save** button (top-right area for default Web Determinations layout); a *save successful* screen is displayed if the screen succeeds and data of the current interview is now saved into a *Case*.
2. The *save successful* screen will also have a link that resumes the interview and a **Continue** button. The link and the **Continue** button returns the user to the same screen where he/she initiated the interview save action; click on either the link or **Continue** button to return to the same screen where you initiated the interview save action (for example, if you were in between the investigation of a Goal, you will be returned to the correct screen so that progress is not lost).

Loading data into a new Web Determinations interview

The user can load the interview at any time.

Note: Loading interview data means that the current interview session will be lost. Loading interview data always requires a fresh interview session.

1. Click the **Load** button (top-right area for default Web Determinations layout); the *Load* screen is displayed. It has a list of saved interview data listed using the *Case ID*. Each *Case ID* is a link; only saved *Cases* for the current rulebase are listed.
2. Click on one of the *Case IDs* to load it; the *Case ID* is loaded, and you are taken to the *Summary* screen of the current rulebase.

Accessing the saved XDS files

The saved XDS files are available in `<web-determinations>/data/<rulebase name>`; for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\data\MyRulebase\Case1.xds`

Document Generator Plugin overview

Sometimes just running through an interview isn't enough; you need a way to provide users with some or all of:

- a record of the interview.
- the answers they provided.
- the conclusion(s) reached.

Oracle Policy Modeling allows rulebase authors to create documents from interviews performed in either Web Determinations or the Determinations Server. These documents are accessible either by links inserted into interactive screens, or programmatically by way of the Determinations Server API.

Providing such a document is the responsibility of the Document Generator plugin, which since Oracle Policy Automation (OPA) v.10.3.0, is based on Oracle Business Intelligence Publisher (BI Publisher). The version of BI Publisher used was upgraded in OPA 10.4.5 from v.11.1.1.3 to v.11.1.1.7.

With this support, users can view and download transcripts about the interview session in a variety of different formats. Oracle Policy Automation ships with support for HTML, RTF, PDF and Excel documents out of the box, though this functionality can be extended through the use of a Custom plugin.

The Document Generator plugin is an Engine Session plugin to the Interview Engine that allows rule authors to generate those documents in a given type, and also define the document layout and which session data is displayed.

Only one Document Generator plugin is used by the Engine at any one point and while by default, Web Determinations ships with a BI Publisher Document Generator plugin, this can be replaced by a Custom plugin. For more information about using the default plugin, see the topic, [Use the Default Document Generator](#).

When providing a Custom Document Generator plugin it is still possible to call the BI Publisher plugin, therefore providing a way to easily extend the Document Generation ability of Web Determinations without losing existing functionality.

Go to:

[Common scenarios](#)

[Document Generator and the Web Determination architecture](#)

[Developing a Document Generator for a specific project/implementation](#)

[Miscellaneous notes](#)

See also:

[Use the Default Document Generator](#)

[Document Generator - sample code](#)

[Legacy Document Generation](#)

Common Scenarios

The following are brief descriptions of common implementation scenarios that use the Document Generator:

Pre Populated Claim Forms

Often Web Determinations is used in places where an official form already exists. One common scenario is self service applications in which potential clients can use Web Determinations to conduct a guided interview to assess their eligibility for a range of benefits

and services.

When the Web Determinations interview is complete, the *Summary* screen includes a download link for the form that has been pre-populated with information provided during the interview and which the user is able to print, sign and send off in order to have their claim processed. This is especially useful when the user must provide additional evidence with their application, that it is not possible to validate in an online interview.

Advice Letters

In many situations where there is interaction with a customer, it can be beneficial to provide them with a summary of either the changes they have made or the advice they have received in the form of a letter.

Take the example of a call center where a person has called to advise of a change in circumstances. The call center agent uses a Web Determinations interview to collect the change in circumstance information and determine what effect that has on the client's current situation. At the conclusion of the interview, the agent can then use the document generation process to generate a letter detailing the change of circumstance information and its effect, which can then be mailed or e-mailed to the client.

Auditing

While the data adaptors give you the ability to save the current state of user entered information in an interview, it may not always be possible to record the full audit trail of why the decision was made, or to do so in a human readable format.

Where, for auditing purposes, it is required that a breakdown of the interview including decision reports be saved out, the document generation process can be used to provide a document that can be saved for later retrieval. This document would be fully customizable to a format best suiting the purposes of the audit.

Document Generator and the Web Determination architecture

This section details how the Document Generator fits into the Web Determinations architecture, and how to use it in the Web Determinations environment. It is important to note the following terms:

BI Publisher Document Generator - the Interview Engine Plugin providing default document generation.

BI Publisher FOP Engine - a set of Java Libraries used to compile templates and render documents.

Document Generation Server - a Java Servlet wrapping the BI Publisher FOP Engine, used by both the Java and .Net BI Publisher Document Generator plugins

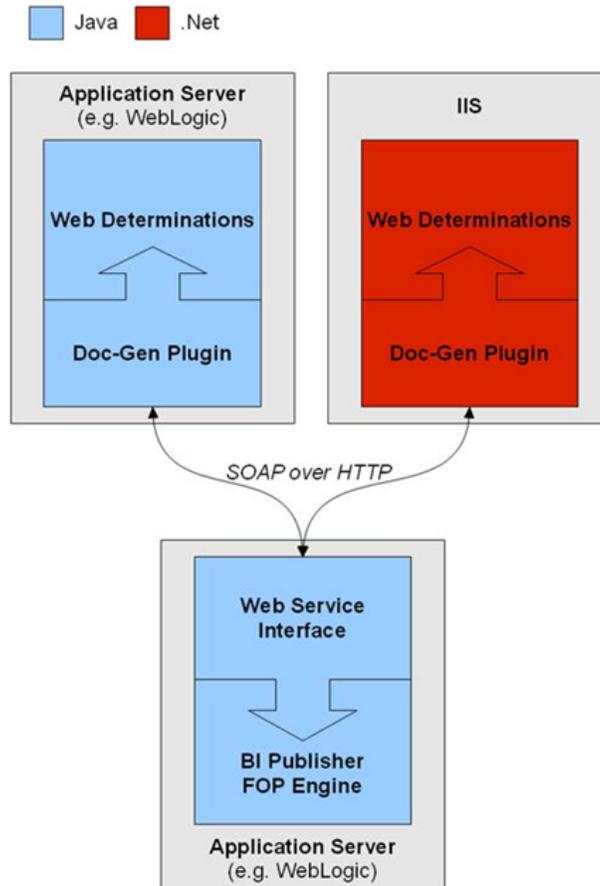
Document Template - an object containing the parameters required to generate a document, including the document's id, name and the location of the template it uses

RTF Template - the template produced in Word which is compiled by the BI Publisher FOP Engine and used to generate documents.

BI Publisher Document Generator plugin architecture

Only one Document Generator is ever "active" in the Engine at once. By default this is the **BI Publisher Document Generator**, and this is replaced when a Custom Document Generator Plugin is detected and loaded.

The plugin itself is responsible for generating XML from the session and maintaining a list of templates, but it only acts as a proxy for the document generation process. The actual generation of a given document from the session XML and template is done by the Document Generation server.



In this way it is possible to have one back-end FOP Engine being used by multiple Web Determination and Determination Server deployments, thus reducing infrastructure and licensing overheads.

Standard Document Generation

The two most common ways of calling the Document Generator are via either a link in a Web Determinations interview, defined by a rule author, or via the Generate Document method of the Determinations Server. These two methods both follow the same procedure.

Each time a new **InterviewSession** is instantiated it creates a new instance of the BI Publisher plugin. This instance is created based on the target URL of the Document Generation server and the locale of the session.

When a document is requested, a call is made to the Interview Session's **GenerateDocument** method, with the ID of the **DocumentTemplate** to be generated and a flag indicating whether or not only the session XML should be returned (**GenerateXml**).

If this flag is set then the plugin bypasses the template compilation check and document generation stage, instead returning the session XML straight back to the user in place of the document. This is useful for producing sample XML that can be imported into the BI Publisher Authoring plugin for Word.

Assuming the **DocumentTemplate** can be found, a check is made to see if the RTF template is available. This template must be compiled before it can be used by the BI Publisher FOP Engine and this compilation process occurs automatically when a template is used for the first time. Compiled versions of the templates are cached in the **BIPublisherDocumentGenerator** plugin, and used

for subsequent requests until a change to the deployed rulebase is detected. At this point, the template is flagged to be reloaded and recompiled on the next request.

Assuming a successful call to the Document Generation server, the document is returned as a **TypedInputStream**, for either display to the end user or transmission by the Determinations Server.

Document Generator plugin and other Web Determination extensions

Since the plugin is accessible via the **InterviewSession** it is also possible to trigger document generation from custom plugins and event handlers. This is especially useful in cases where you wish to save or forward the generated document automatically.

When calling the **generateDocument** method of an **InterviewSession** the only required parameter is the public id of the document you wish to generate. This is defined by the rule authors when the document is added to the Screens file. A list of **DocumentTemplates** is also available through the **InterviewRulebase** object itself by calling the **getDocumentRegistry** method with the appropriate locale.

The following is a piece of sample code to generate a document with the id of "final_report" from an **InterviewSession** object stored in **iSession** (Note that the second argument indicates we do not wish for the session XML to be returned in place of the document):

```
TypedInputStream document = iSession.generateDocument("final_report", false);
```

Document Generator class methods

A Document Generator plugin implements the **DocumentGeneratorPlugin** interface, which in turn extends the **InterviewSessionPlugin** interface. When implemented, the **DocumentGeneratorPlugin** interface requires the following methods:

Method Signature	Description
TypedInputStream generateDocument(InterviewSession session, DocumentTemplate parameters, Boolean generateXml)	<ul style="list-style-type: none">• The InterviewSession contains information about the current Web Determinations interview such as the rulebase and instance data.• The DocumentTemplate contains information about which template to use, the name of the document, which attributes should be displayed, and any attributes that should have their Decision Reports generated with the document.• The Boolean generateXml tells the Document Generation plugin whether it should just return the session XML instead of the fully rendered document.• An object of TypedInputStream is returned, which is commonly returned by Web Determinations to the user as a downloadable binary file
String getInstance()	<ul style="list-style-type: none">• Called when the plugin is first registered in the engine.

Developing a Document Generator for a specific project/implementation

This section explains the various approaches on designing and developing a Document Generator plugin for a specific project/implementation.

Factors to consider when designing a Document Generator plugin

When designing a Document Generator plugin, consider the following:

- What document type (and its corresponding mime type) will the Document Generator plugin provide?
- What support files (if any) does it need to use to generate the document?
- Will the Document Generator need third-party libraries to generate the document, or to read support files such as Excel or PDF?
- Will the generated document be returned to the user, or will it be handled by another Web Determinations extension?

Essentially, the Document Generator plugin needs to have the following functionality:

- Be the only Document Generator registered for a Web Determinations interview.
- Read the Web Determinations interview session data to extract the instance data needed to be written out to the document.
- Use the instance data together with layout/styling/formatting data (either coded in the plugin itself or via support file/s) to generate the document and return it as a **TypedInputStream**.
- Call the **BIPublisherDocumentGenerator** if necessary.

Programming the method generateDocument()

It is important to understand the three input arguments for the **DocumentGenerator** method **generateDocument()**.

1. **InterviewSession**

This object is useful for the instance data provided by the user during the Web Determinations interview. The **InterviewSession** object can also provide the rulebase used, rulebase model data, locale, and so on. More information about the InterviewSession can be found in [Web Determinations extensions - technical details#About the InterviewSession](#).

2. **DocumentTemplate**

These objects have the following members:

- Strings Id, Name, Template, Type
 - These strings define the basic identifying information for the **DocumentTemplate** and detail where the template to be used is stored.
- Booleans includeUserSet, includeGoal, includeIntermediate
 - These booleans define which subset of attributes should be included in the generated document.
- DecisionReportTarget[] includedDecisionReports
 - A list of attributes or relationships that should be taken note of by the Document Generator. Each Document item in the screens file (when in Oracle Policy Modeling Screen authoring) has a list of attributes for 'Decision Reporting'. How a Document Generator uses this list of attributes is up to the implementor of that Document Generator - it can be simply ignored, or decision reports for each of the attributes generated in the document, or maybe even just a subset of the attributes in the list.
- Map customProps
 - A map containing any custom properties set for the document when it was created in Oracle Policy Modeling.

Using the **BIPublisherDocumentGenerator**

The **BIPublisherDocumentGenerator** class contains a static method **generateXml** which can be used by Custom Document Generators to translate a given session into xml format based on a **DocumentTemplate** object. This can be useful when design-

ing a custom document generator as it bypasses the need to worry about session formatting and allows the document generator to focus on transforming the session xml into the appropriate output format.

Miscellaneous Notes

- As of Oracle Policy Automation V10.3.0 there is only ever one Document Generator plugin registered during a Web Determinations interview.
- A Document Generator can be accessed by Web Determinations extensions via the **InterviewSession**.
- The template filepath in **DocumentTemplate** is a relative path to the document's template inside the rulebase zip file.
- For more information on using BI Publisher, see:
 - the Template Builder for Word Help file (available under \Program Files\Oracle\BI Publisher\BI Publisher Desktop\Template Builder for Word) and/or
 - the BI Publisher Users Guide (available under \Program Files\Oracle\BI Publisher\BI Publisher Desktop\Template Builder for Word\doc).

Create a Document Generator

The following describes the steps specific to creating a Document Generator plugin. For steps on creating and installing a plugin, see [Create a plugin](#). Although this topic focuses on Java development, the same steps can be followed, for the most part, for .Net.

For more information about Document Generator plugins, see [Document Generator plugin](#).

A custom Document Generator plugin is most commonly needed to:

- Integrate with a third party document generation provider.
- Provide document formats that are not supported in the default Document Generator (HTML, PDF, RTF and XLS are all currently supported).
- Access a data source to obtain other data not present in the interview, to be added into the document.
- Provide a different process to generating HTML and PDF format (for example, under .Net where the Document-Generation-Server cannot be hosted).

Assets and information needed

These assets are required to develop a Document Generator plugin (Libraries can be found either in the /lib/ or /bin/ directories of your Web Determinations deployment).

- **Interview Engine Library** - *determinations-interview-engine.jar* / *Determinations.Interview.Engine.dll* (and associated dependencies).
- **Determinations Utilities Library** (Java Only) - *determinations-utilities.jar* (and associated dependencies).
- **Masquerade Library** (.NET Only) - *Oracle.Determinations.Masquerade.dll*.

The following assets are recommended when developing a Document Generator Plugin.

- **Web Determinations Library** - *web-determinations.jar* / *Web.Determinations.Platform.dll*.
- **Determinations Engine Library** - *determinations-engine.jar* / *Oracle.Determinations.Engine.dll*.
- **IDE** - this will make it much easier when referring to objects in the library files.
- **Document Generator plugin business case** - this is the reason/need why the custom Commentary plugin is being developed.

Steps

Analysis and design:

Before starting on the development of the plugin, read the [Document Generator plugin](#) topic.

Once familiarized with the Document Generator plugin, do the following:

1. Determine what rulebase (or rulebases) the Document Generator will be registered in. You can make a custom Document Generator plugin that services all the rulebases and its locales in a Web Determinations installation, or you can create the Document Generator plugin to only work for a specific Rulebase, or even only for one of its locales.
2. Determine the document type (file format) that the Document Generator will provide.
3. The **Document Generator plugin business case** should be able to address the following:
 - i. Will the plugin use a template to generate the document?
 - ii. Does the plugin require data other than what is available in the interview session?

- iii. Does the plugin need to access datasource/s to retrieve data?
- iv. Will the documents generated need to have binary content?

4. Design how the Document Generator plugin:

- i. will retrieve data that is not available in the interview session (if non-interview data is part of the document).
- ii. will access the datasource (if datasource is involved).
- iii. will construct and generate a document with non-text content.

Development

1. Setup the **IDE** so that it has the **Web Determinations libraries** and **data source libraries**. This makes it easier to refer to objects for the Web Determinations or the data source.
2. Create a document generator plugin class by implementing the **DocumentGeneratorPlugin** interface (See [Document Generator - sample code](#) for examples).
3. Develop the Document Generator plugin based on the *Analysis and design* section above.

Install and test

For steps on installing the Document Generator plugin and testing, refer to the topic, [Create a plugin](#).

Use the default Document Generator

Document Generators allow users to download and save a document with data from a Web Determinations interview. The document is usually a transcript of the interview for record keeping, but can be any document that requires data from the interview to be embedded in it, such as a pre-filled form, letter of advice or even a specially formatted web page.

The default Document Generator allows users to save and print a transcript of their Web Determinations interview in the following formats:

- HTML
- PDF
- XLS
- RTF
- XML (for debugging)

This topic describes the process necessary to create and debug a basic Interview containing a Document Generation link. Following all the steps in this tutorial will require the following:

- Oracle Policy Modeling (you are expected to have installed Oracle Policy Modeling and have previous experience authoring rulebases).
- Java application server (this tutorial uses Tomcat).
- 1 hour (this tutorial should take no longer but can take considerably less).

Go to:

[Set up the Document Generation server](#)

[Author a basic rulebase](#)

[Add document generation capabilities](#)

[Summary](#)

Set up the Document Generation server

The first and most important step to take before using the default Document Generation plugin, is to deploy the Document Generation server. This server acts as a wrapper for the BI Publisher Document Generation Engine and without it you will only be able to produce XML files from your document links.

If you already have a working Java application server then please deploy the *document-generation-server.war* file to this server, making a note of the resultant end-point.

If you don't have a working Java Application Server, or don't understand what one is, then the following steps will guide you through installing one.

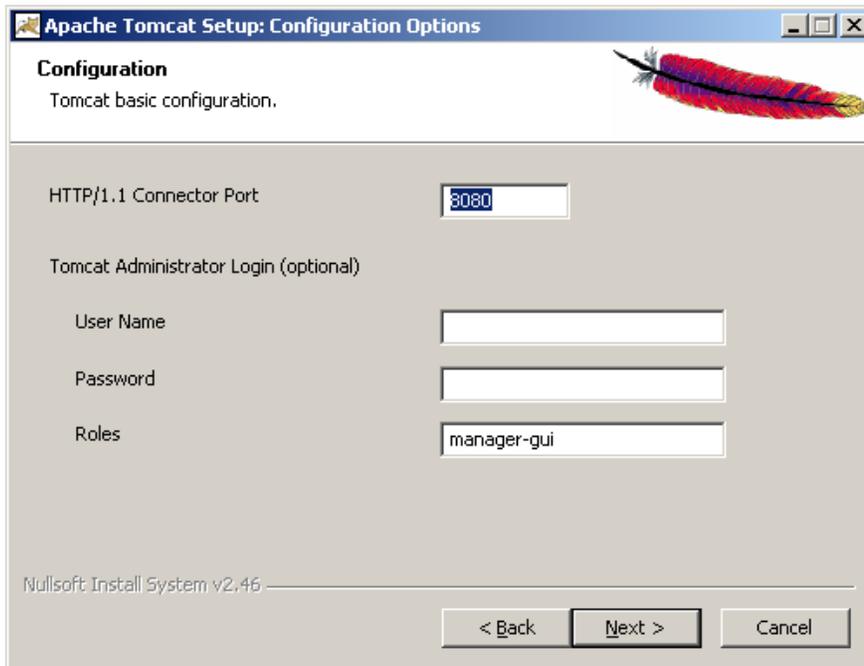
Install Tomcat:

- Use a browser of your choice and proceed to <http://tomcat.apache.org/>.
- On the left you should see a section of links under the *Download* header; click on the *Tomcat 6.0* link – this is the version of Tomcat that we will be using for this tutorial.

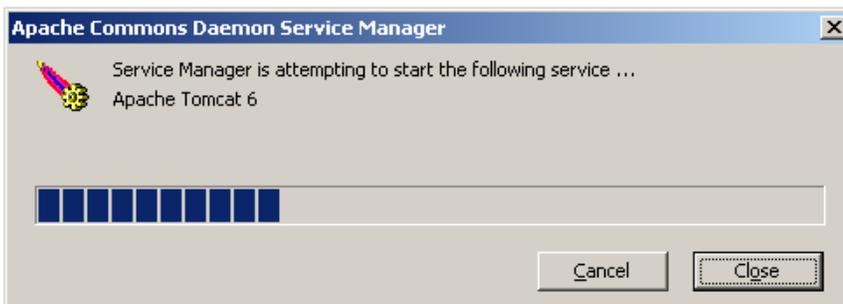
- Scroll down and find the *32-bit/64-bit Windows Service Installer* link – download this file to an easily accessible location.



- Launch the installer you have just downloaded; you will be asked to accept a license agreement, which you should read, then asked which components you wish to install. The default of a “Normal” install is fine!
- On the next page, make a note of the HTTP/1.1 Connector Port that you choose; this is important as you will need it later.



- Proceed through the rest of the installer, making a note of the directory you choose to install Tomcat to, and once it has completed ensure you Run Apache Tomcat. You will likely see a dialog box pop up similar to the one below which indicates that Tomcat is starting up



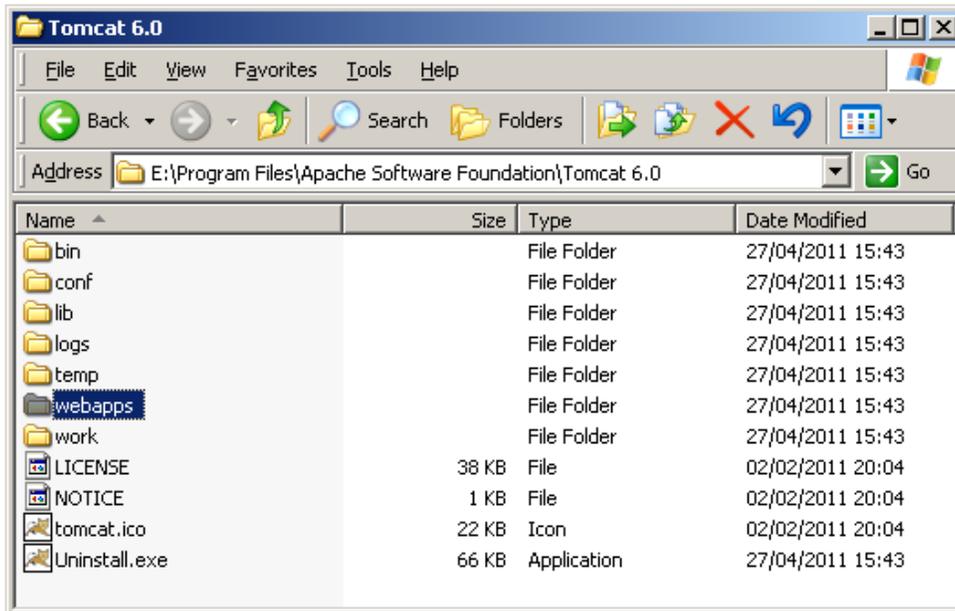
- You should also see a new Tray Icon which, if Tomcat starts up successfully, will display a "Play" symbol as shown below. If Tomcat fails to start and a "Stop" symbol is shown inside the icon, you will need to get in touch with your support team to help you determine why this is happening.



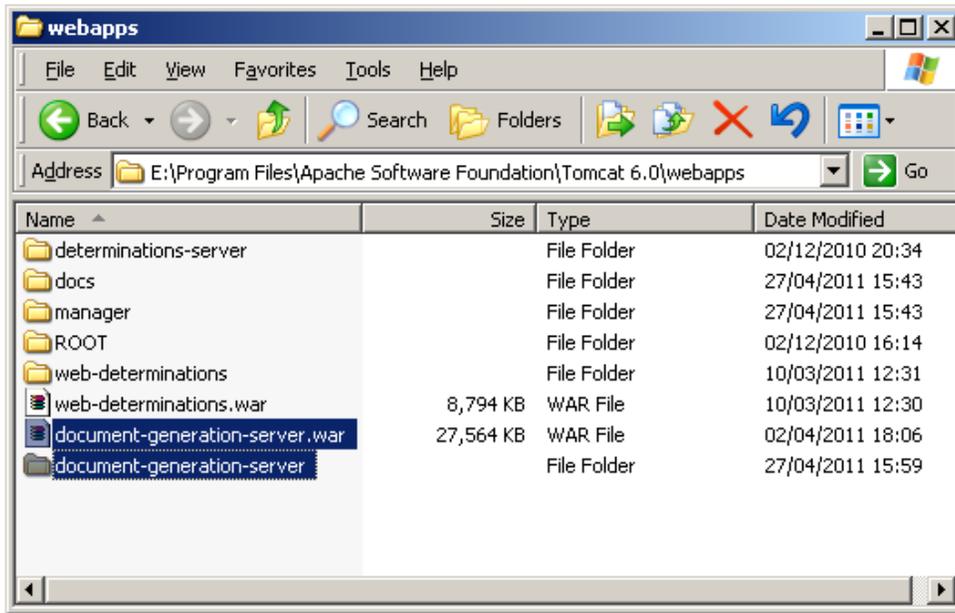
Deploy the Document Generation server:

Now that Tomcat has been installed and is running, it's time to deploy the Document Generation Server as follows:

1. Locate the Document Generation server; by default it is located in **Program Files\Oracle\Policy Modeling\templates\document-generation-server** as a file called *document-generation-server.war*.
2. Once you have located this file, deploying it is just a case of copying it into the right directory. Navigate to the directory you installed Tomcat to and there you should see a sub directory called *webapps*.



3. Copy the *document-generation-server.war* file into this directory and wait for a few seconds. The first indication that it has been deployed successfully is a new folder that should appear called *document-generation-server*.



- Remember the HTTP/1.1 Connector Port you made a note of earlier? You'll need that now. Go to <http://localhost:8080/document-generation-server> (substituting 8080 for the Port you've got written down if it is different) where you should see the following, indicating that the deployment has been successful.



Author a basic rulebase

We are now going to author a very quick rulebase that we can use to test out the new Document Generation capabilities found since 10.3.0.

If you already have a working rulebase that you wish to use, you can skip this section and go directly to [Add document generation capabilities](#).

Instead of walking step by step through the process for authoring the rulebase, you will just be given the information required to replicate it on your system.

Rules:

The person is eligible for a fake benefit if

- The person is of appropriate age and
- The person's earnings are under the threshold

The person is of appropriate age if

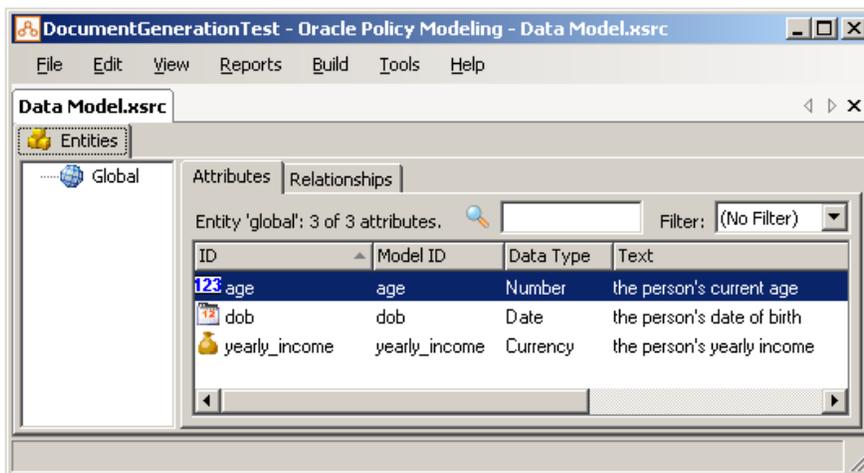
- The person's current age > 59

The person's current age = YearDifference(the person's date of birth, the current date)

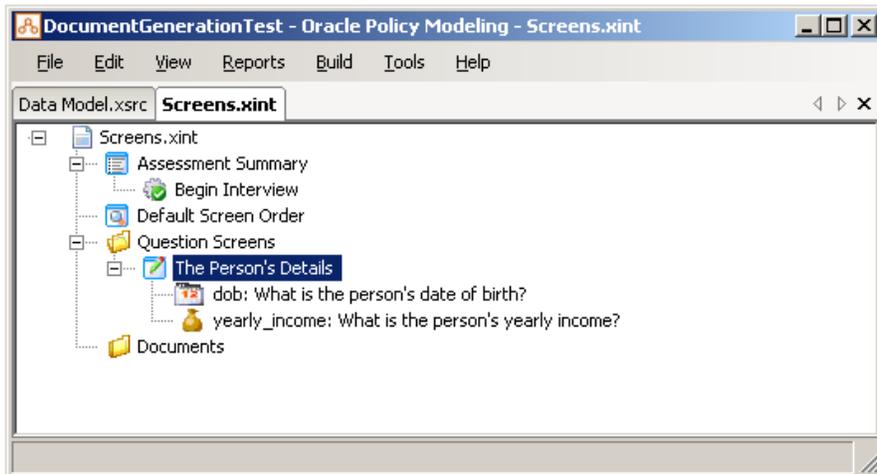
The person's earnings are under the threshold if

- The person's yearly income <= 25,000

Data model:



Screens:



Add document generation capabilities

In order to add document generation to our rulebase, we need to do three things:

1. Create a Document Definition in the Screens file.
2. Access the automatically generated Document Template, that is used by BI Publisher to define the format and contents of the generated document.
3. Create a Document link, generally placed on the *Summary* screen post-interview, to allow access to our generated document.

These things are done in order below for a simple confirmation letter example.

Add a Document definition:

Right click inside the Screens file and you will see the option to add a **New Document**. You can place this document anywhere inside the *Screens* file, though it's recommended that you place it within a specially created *Documents* folder, to aid organization.

Clicking the **New Document** option presents you with the following dialog box, the fields for which are explained below:

Field	Description
Document Name	The name by which you will refer to this document – every effort should be made to ensure it is unique unless specifically required otherwise. Good examples would be: “Advice Letter w/Decision Report”, “Claim Form (Ref:ABC1)”.
Public Name	Similar to an Attribute, Entity or Relationship’s public name, this must be unique and if not specified here, will be auto-generated. Good examples would be: “adv_letter_decision_report”, “claim_form_ABC1”.
Template	This is the template file (in rtf format) associated with the document. All templates should be placed in the <code>include\templates\<locale></code> folder underneath the main rulebase folder. Adding a new document automatically creates a blank document template.
Document Type	Since templates can be used to generate any support document, it is in this drop down that you decide what the format of the final document you generate should be. If you wish to output XML then see the section on adding Document links to screens and specifically the “Generate Xml” flag.

Field	Description
Decision Reports	<p>Adding entries into the Decision Reports list not only ensures that attribute is present, but that a full decision report is also available for it.</p> <p>Take care not to add unnecessary entries here, as they may slow down the document generation process.</p>
Export Schema	<p>This button allows you to export an XSD representation of the Document Definition. This is important as you will generally need to import this XSD file into the BI Publisher Word plugin in order to use the helper functions and dialogs provided.</p> <p>Remember that this XSD file will be different if you make changes to either your rulebase data model or Included Decision Reports.</p>

If you are following our rulebase example, you will need to enter the following into your new document.

The screenshot shows a dialog box titled "Example Document - Document". It contains the following fields and controls:

- Document Name:** Example Document
- Public Name:** example
- Template:** Example Template.rtf (with an "Edit" button)
- Document Type:** PDF (dropdown menu)
- Decision Reports:** A list box containing "the person is eligible for benefits" with "Add" and "Delete" buttons.
- Buttons:** "Export Schema", "OK", and "Cancel" are located at the bottom of the dialog.

Access the Document template and export the schema:

Templates for the document generation process are automatically created when adding a document and are then enriched using Microsoft Word and the BI Publisher Word plugin. It is beyond the scope of this tutorial to attempt to teach you how to author templates for BI Publisher, but a few important points should be noted.

- The XSD file exported using the **Export Schema** button from the *Document* dialog box we looked at in the last section can be imported into the BI Publisher Word plugin using the **Data -> Load XML Schema...** menu option, but this can be very slow!
- In the next section we will look at saving XML from the Document Generation process, and importing this into the tool instead using the **Data -> Load Sample XML Data...** option. Not only is this faster but it also makes it much easier to preview the output document from within Word.

If you are following our example rulebase, you will need to do the following:

1. Export the Document's Schema to an easily accessible location; you will need to import this into the BI Publisher Word plugin if you don't want to be authoring blind!
2. Click on **Edit** to begin authoring the template.

For our example rulebase, we can use the following template contents:



Fake Company Inc.

Dear Sir/Madam,

With regards to the person who filled out an online interview with us recently;

`for-each eligible text end eligible`

This is because you told us your Date of Birth was:

`for-each dob value end dob`

And your yearly income was:

`for-each yearly_income value end yearly_income`

Thank you!

A handwritten signature in black ink, consisting of several overlapping loops and a long horizontal stroke extending to the right.

Smith Smithson
President, CEO, Sole Employee

Add a Document link:

The last step is to give the end user a way to access the document. We do this by adding a Document link to the *Summary* screen of our rulebase. This link will have a number of options that will already be familiar, and two which are not.

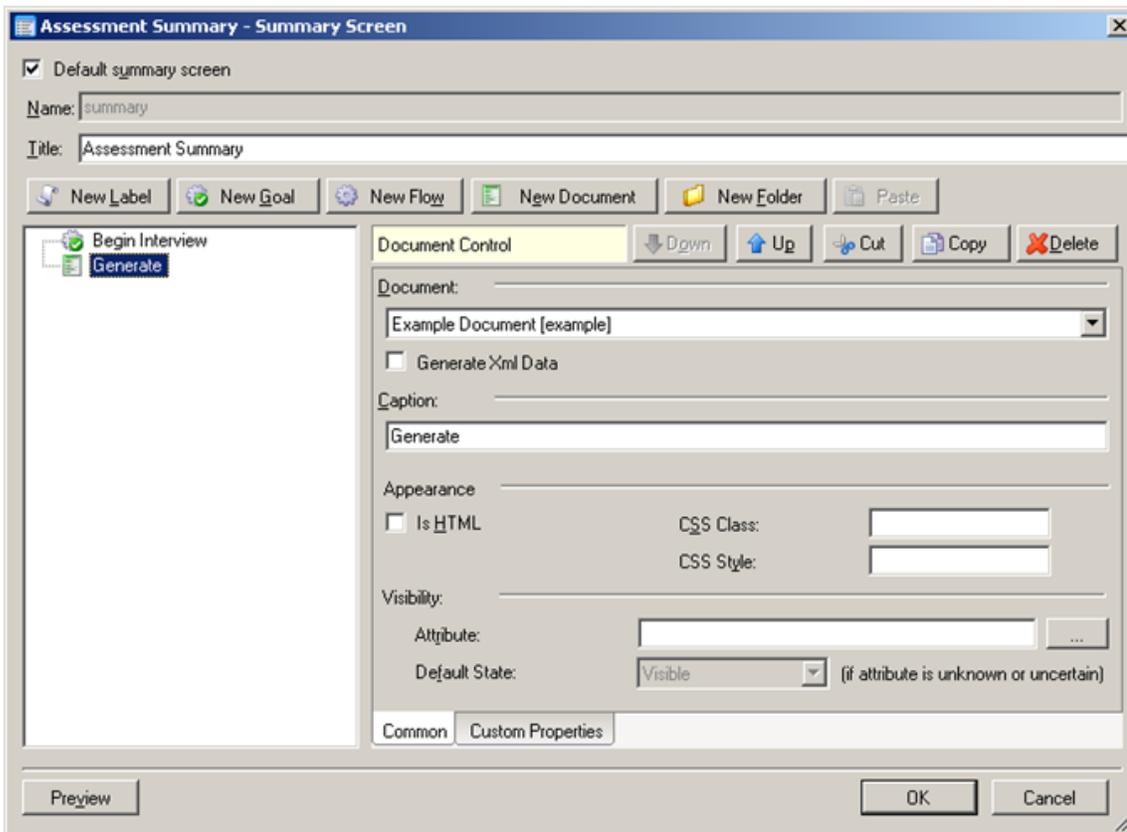
Document

This drop down box allows you to pick the document that you wish to generate when this link is clicked with the list in the format of "Document Name [Public Id]"

Generate Xml Data

When checked, this causes the link to bypass the BI Publisher Document Generation Server, instead returning a raw XML representation of the Session. This is useful not just for debugging purposes, but for importing into the BI Publisher Word plugin.

In our example rulebase, the last step is to add a Document link to our summary screen, as shown below:



Known issue

When following the document link from the *Summary* screen, if the BI Publisher template's Conditional Region settings have not been correctly defined (for example, an incorrect data type has been used), then the user may encounter a generic error. In such a case, it will be necessary to visit the BI Publisher template and make the appropriate adjustments, referring to the BI Publisher user documentation for assistance.

Summary

There is a lot more to the Document Generation capabilities of Oracle Policy Modeling than could be covered in a short tutorial, but the basic steps covered by this document are fundamental to any use of the default document generation process.

For more information, consult the following topics:

[Document Generator plugin overview](#)

[Create a Document Generator](#)

[Document Generator – sample code](#)

[Legacy Document Generation](#)

List Provider plugin overview

Rule authors can control the values that a user can provide for a variable attribute by using a List control instead of text field input. List controls are drop-down lists, list-boxes, and radio button lists. The simplest way to define the values of a list is via Screen Authoring in Oracle Policy Modeling, where the rule author inputs each selectable option for the list control. While simple, it is not reusable, flexible, or dynamic.

The List Provider plugin is an Engine Session plugin to the Web Determinations Engine which provides an alternate way of populating List controls via Screen Authoring.

A List Provider plugin allows a list to be loaded with values from any datasource. The default List Provider plugin in Web Determinations web application uses xml files to drive lists.

More information on using the default List Provider can be found in [Use the default List Provider](#)

Go to:

[List Provider and the Web Determination architecture](#)

[Developing a List Provider plugin for your specific project/datasource](#)

See also:

[Create a List Provider](#)

[Use the default List Provider](#)

[List Provider - pseudo code](#)

[List Provider -sample code](#)

List Provider and the Web Determination architecture

The following information details how the List Provider plugin fits into the Web Determinations architecture, and how to use it in the Web Determinations environment.

How Web Determinations use the List Provider plugin

Web Determinations uses the List Provider plugin every time a List **InterviewInputControl** object is instantiated in Web Determinations.

- It retrieves the List Provider registered to the current Web Determinations interview.
- Calls the List Provider to retrieve a List object that contains **ListOption** objects.
- Each **ListOption** object represents an option item in the List control.
- If the List Provider returns a null List object, Web Determinations uses the list values set by the rule author in Oracle Policy Modeling.

List Provider class methods

A List Provider plugin implements the **ListProviderPlugin** interface. The **ListProviderPlugin** interface requires the following when implemented:

Method signature	Description
List getListOptions(InputInterviewControl controlInstance, InterviewSession ses- sion)	<ul style="list-style-type: none"> • The InputInterviewControl contains information about the List control. • From the InputInterviewControl the Attribute object can be retrieved (which contains information about the variable attribute linked to the List). • The InterviewSession contains information about the current Web Determinations interview, such as the current rulebase, instance data known so far, and so on. • The List object returned contains ListOptions, and each ListOptions contains the information for each option item in the List control. • If the returned List object is null - Web Determinations uses the list of values defined by the rule author for the List control in Oracle Policy Modeling Screen Authoring.

Developing a List Provider plugin for your specific project/datasource

This following information explains the various approaches to designing and developing a List Provider plugin for a specific project, rulebase, or datasource.

Factors to consider when designing a List Provider plugin

When designing a List Provider plugin, keep the following in mind:

- What datasource/s will it be using?
- Which rulebase/s will the plugin service?
- How will the List Provider retrieve and map data to its List control?
- Maintainability and flexibility - does the List Provider plugin need architecture to be able to easily change the mapping of data to List controls?

Essentially, the List Provider plugin needs to have the following functionality:

- Inspect the current List control to determine what dataset is needed.
- Retrieve the dataset from the datasource.
- Construct the List to be returned from the retrieved dataset.

The List Provider plugin can also use multiple datasources to populate various List controls.

Simple implementations of the List Provider plugin may use simple logic to retrieve and map datasets for each List control it needs to populate. As the complexity increases (multiple datasources, multiple List controls, supporting several rulebases), providing configuration files becomes a necessity to be able to handle changes from the datasource, rulebase, or the List controls.

About the data passed into the LP method 'getListOptions'

It is important to understand the two input arguments for the **ListProvider** method **getListOptions**.

InputInterviewControl

The **InputInterviewControl** object is useful for determining what attribute the List control is attached to, which can be used in the Plugin logic to retrieve and setup the dataset for the List control.

The **InputInterviewControl** object passed in is the object representation of the List control. It contains valuable data about the List control, such as:

- The **Attribute** object for the List control, which holds information about the **Variable** attribute attached to the List such as public name, and variable type.
- The default value for the List as set by the rule author in Oracle Policy Modeling.
- Is the field Mandatory.
- Is the field Read Only.

InterviewSession

The **InterviewSession** object is useful for determining the current state of the Web Determinations interview such as what instance data has been provided by the user, as well as Web Determinations interview metadata such as the locale and rulebase.

How to build the List object to be returned in getListOptions()

Building the List object is very simple. Here are the general steps to build the List object:

1. Inspect the List control to determine what data to retrieve.
2. Retrieve the data for the current List control.
3. For each option item in the retrieved data,
 - a. Create a **ListOption** object and populate its displayText and value members.
 - b. Add the **ListOption** to the List object.

ListOption object

The **ListOption** object has the following key members:

- displayText - the text displayed in the rendered List control; for example, New South Wales.
- value - the actual value assigned to the **Variable** attribute; for example, NSW.

Create a List Provider

The following describes the steps specific to creating a List Provider plugin. For steps on creating and installing a plugin, see the [Create a plugin](#) topic. Although it focuses on Java development, the same steps can be followed for .Net (with slight implementation difference in certain areas of course).

For more information about List Provider plugins, see the topic, [List Provider plugin](#).

Assets and information needed

The following are assets that are needed to develop the List Provider plugin, or information that will be referred to in the *Steps* section below:

Web Determinations library jars

- *web-determinations.jar* - this can be found in the Web Determinations folder 'lib' folder; for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\lib`.
- *determinations-interview-engine.jar* - this can also be found in the Web Determinations folder 'lib' folder; for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\lib`.

Java IDE - this is the Java development application you are using. While this is optional, it will make it much easier when referring to objects in the *web-determinations.jar* and *determinations-interview-engine.jar*.

Datasource - this is the datasource (or datasources) that this List Provider will connect to, to retrieve datasets for List control values.

Datasource library jars - the datasource may have Java jar library files that is required for the List Provider.

Datasource connection details - the List Provider plugin will need to be able to connect to the datasource, so connection details must be known. This might be a specific port in a server; for example, `//server:portnumber/databasename`.

Steps

Analyze and design:

Before starting on the development of the plugin, read the [List Provider plugin](#) topic to understand the List Provider plugin.

Once familiarized with the List Provider plugin, do the following:

1. Determine what rulebase (or rulebases) the List Provider will service.
2. Determine which attributes in the rulebase/s will need the List Provider, and whether the List Provider relies on other requirements; for example, the current locale, a certain goal attribute has to be known, and so on.
3. Map out how the List Provider will connect and interact with the datasource, including transaction integrity processes, fall-back on error, and so on.
4. Design how the List Provider will retrieve datasets from the datasource; for example a map might be created where a List Control attribute determines the table/columns to use for data retrieval (see [List Provider - sample code](#)).

Develop

1. Set up the **Java IDE** so that it has the **Web Determinations library jars** and **Datasource library jars**. This makes it easier to refer to objects for the Web Determinations or the datasource.
2. Start the List Provider plugin class from the following pseudo code - [List Provider - pseudo code](#).
3. Develop the List Provider plugin based on the *Analyze and design* section above.

Install and test

For steps on installing the List Provider and testing, see the topic, [Create a plugin](#).

Use the default List Provider

The rulebase list provider is intended as an out-of-the-box alternative to the static selectable list options for input controls given through the Oracle Policy Modeling product. It provides the rulebase engineer with the ability to package list files (in a strict XML format organized by Locale) along with the rulebase archive. For each input control that is to take advantage of these deployed list files, the name of the list file (with or without the '.xml' extension) can be provided on the Input Control Editor screen in Oracle Policy Modeling. When an input control with a backing list XML file is to be displayed in Oracle Web Determinations to the user, the XML file is read. If there are any problems with this file (doesn't exist, wrong XML structure, etc.), the static list options are used (if they exist for the input control).

The rulebase list provider expects the list XML files to exist in the ***lists/[session locale]*** folder in the rulebase archive. The Oracle Policy Modeling Build process also packages the rulebase in the **output** folder automatically and having the Lists folder in the **[Rulebase]/Development/include** folder means it's contents will be picked up as well. In this way, multiple localized versions of the same list can be deployed, with the particular one chosen being dependant on the current Oracle Web Determinations session's Locale. The XML files themselves are expected to be in the following format:

```
<list>
  <option text="[list option text]" value="[list option value]" />
  ...
</list>
```

Using an XML file to populate a list-type Attribute Input Control in Oracle Policy Modeling

1. Locate your rulebase project *Development* folder; the default installation is `C:\projects[Rulebase Name]\Development`.
2. Open the *include* folder (create one if it doesn't exist).
3. In the include folder, create a new folder called *lists*.
4. In the *lists* folder, create a new folder for each locale that this rulebase supports in Web Determinations, using the locale code; for example, English US is 'en-US', English UK is 'en-GB'.
 - a. In most cases, a rulebase running in Web Determinations will support one locale - the rulebase's project language.
 - b. The locale code can be derived by using the locale name (for example, the Project Language) and this table - [Localization - Locale Codes](#).
5. For each locale folder (for example, en-GB for English International)
 - a. Create an xml file. The name of the xml file should be related to the list it will be used for; for example, *JobIndustry.xml*.
 - b. In the xml file, add XML nodes as per stated above. For example, to create list options for a Job Industry list input, with Finance and IT as the list options:

```
<list>
  <option text="Finance" value="Finance" />
  <option text="IT" value="IT" />
</list>
```

- c. Input the above into *JobIndustry.xml*, and save.

6. In Oracle Policy Modeling, open the rulebase project open the Screen with the list Attribute Input Control that will use the XML file, and open that Attribute Input (in this example *JobIndustry.xml*).
7. Scroll down to *Values* section in the main (right-side) pane and find the *Specify list name* label; the *Specify list name* has a radio button to its left, and a text field on the right.
8. Tick the radio button for *Specify list name* as it defaults to 'Specify selection items'.
9. In the text field next to *Specify list name*, enter the XML filename (*JobIndustry.xml* for this example).
10. Save the changes to the screen and project.
11. To check if it has worked, run *Build and Run* command, and check that the list Input has options that you specified in the XML file.

Custom Screen and Custom Control Provider plugins

The following information is intended as a guide for plugin authors wishing to develop custom screen and control providers to extend the natively implemented screen and control objects with extra functionality, new ways of rendering, extra validation, and so on.

The first thing to note is that the actual plugin objects are the custom screen and control *providers*, not the custom screens and controls themselves. A Web Determinations session (represented by the **SessionContext** object) may have at most **one** of each provider. If multiple implementations of one or both of these providers are found, one will be non-deterministically (from the perspective of a plugin developer) chosen. Every time a control or screen is required, Web Determinations will query the provider registered (if one exists) to retrieve a custom screen or control for the current **InterviewScreen** or **ControlInstance** object in the current session context. It is up to the provider implementation to provide concrete instances of **CustomScreen** and **CustomControl** objects (or concrete implementations of any subinterfaces, such as **CustomInputControl** as appropriate) if the provider determines that it can. Otherwise the provider is required to return a *null* object.

The **CustomScreen** and **CustomControl** implementations served by the providers are required to be packaged up and deployed in a plugin archive in the same way that all other plugins are deployed. These dependencies don't necessarily have to be deployed into the very same archive as the providers that reference them, but they must be located inside one of the plugin archives loaded.

Custom screen and control providers are **PlatformSessionPlugin** implementations, meaning that they belong on a particular platform session (**SessionContext**).

Custom screen provider

To provide a custom screen provider implementation, it is necessary to implement the **CustomScreenProvider** interface. The implementation of this interface consists of:

- Implementing the standard **getInstance(args : RegisterArgs) : Plugin** method.
The specific instance of **RegisterArgs** passed through is **PlatformSessionRegisterArgs**, which encapsulates the current **SessionContext**. In this way, plugin authors can determine whether or not they would like their custom screen provider implementation to be registered on a particular session based on such data as the session's rulebase. It is in this way that multiple custom screen providers may be deployed without contesting with each other for registration against a particular session (for example, have one per rulebase).
- Implementing the **getScreen(context : SessionContext, iScreen : InterviewScreen) : CustomScreen** method.
This is the method queried each time a **Screen** object is required. If the provider can provide a custom screen instance for the given session context and Web Determinations Engine screen model object then it must return it. Otherwise, this method must return null.

Custom screens

Custom screens are defined by implementing the **CustomScreen** interface. The implementation of this interface consists of:

- Implementing the **processAndRespond(context : SessionContext, uri : URI) : Response** method.
The custom screen implementation is responsible for doing everything for itself that is usually taken care of by the **Action** framework. There is no restriction on what is actually implemented in this method, as long as a valid **Response** object is returned. It is this response object that is then passed back to the servlet and displayed to the user.
- Implementing the **submit(screenId : String, params : Map, context : SessionContext) : boolean** method.
This method is responsible for submitting the screen to the **InterviewSession** and returning a boolean representing

whether or not the submission was successful. The screen ID is the ID of the interview screen submitted and the parameter map is the one submitted with the POST.

More information

For more information on creating a custom screen, go to:

- [Create a Custom Screen](#)
- [Create a Custom Screen example](#)

Custom control provider

To provide a custom control provider implementation, it is necessary to implement the **CustomControlProvider** interface. The implementation of this interface consists of:

- Implementing the standard **getInstance(args : RegisterArgs) : Plugin** method.
The specific instance of **RegisterArgs** passed through is **PlatformSessionRegisterArgs**, which encapsulates the current **SessionContext**. In this way, plugin authors can determine whether or not they would like their custom control provider implementation to be registered on a particular session based on such data as the session's rulebase. It is in this way that multiple custom control providers may be deployed without contesting with each other for registration against a particular session (for example, have one per rulebase).
- Implementing the **getControl(iCtrl : ControlInstance, parent : ControlContainer, screen : NativeScreen) : CustomControl** method.
This is the method queried each time a **Control** object is required by the screen controller when populating the controls on a screen. If the provider can provide a custom control instance for the given Web Determinations engine control instance that supports being placed inside the given control container on the given Web Determinations platform native (non-custom) screen then this instance is to be returned. Otherwise return *null*. It is important to remember that this method is only called automatically if the screen on which the control is needed is a native screen, **not** a custom screen. If you would like to place a custom control on a custom screen, it is your responsibility to instantiate that control yourself from the custom screen's **processAndRespond** method, or wherever it is that you are constructing your custom screen model (if at all).

Custom controls

Custom controls are defined by implementing the **CustomControl** interface. This is an empty marker interface that extends from the base control interface, which defines a variety of methods that controls have to implement. There is also a **CustomInputControl** interface to be implemented if your custom control is intended to collect user input data that maps to at least one rulebase attribute. Ensure that your implementation matches the following requirements:

- The class name you have chosen for your custom control implementation does not conflict with any of the native controls.
- You have provided a template to render your custom control named *[custom control class name].vm*.
- You have implemented the **getControlType() : String** method to return the class name of your control.

More Information

For information on creating a custom control and for a walkthrough example, go to:

[Create a Custom Control](#)

[Custom Control - BenefitCode walkthrough example](#)

Create a Custom Screen

The following information describes the steps on how to create a custom screen, installing it onto a Java webserver, and running the custom screen. While this covers Java, similar steps can be followed for .Net.

For generic background on how to construct a plugin, see [Create a plugin](#).

For a description of custom screens and their purpose, see [Custom Screen and Custom Control Provider plugins](#).

For how to implement a custom screen plugin for Web Determinations, see [Create a Custom Screen example](#).

Assets and information needed

The following are information that will be referred to in the steps below:

Web Determinations plugin folder - this is where the Web Determinations webserver will pick up the plugins. The user needs to place the plugin packages (.jar for Java).

- J2SE(Java) - default is in the **WEB-INF\classes\plugins** path in the Web Determinations web application in the Tomcat webapps folder, for example, **C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\interactive\WEB-INF\classes\plugins**.
- IIS (.Net) - default is in **C:\inetpub\web-determinations\plugins**.

Web Determinations library jars

- *web-determinations.jar* - this can be found in the Web Determinations folder 'lib' folder; for example, **C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\interactive\WEB-INF\lib**.
- *determinations-interview-engine.jar* - this can also be found in the Web Determinations folder 'lib' folder; for example, **C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\interactive\WEB-INF\lib**.

Steps

To develop a custom screen:

1. Planning -- which default screen will your custom screen be replacing? How will it know? What data will your screen be handling? The custom screen provider can use any available information for each screen, to determine whether it will insert a custom screen. A common technique is to key on the screen name set by the screen author in Oracle Policy Modeling.
2. Start with a standard screen with standard controls allowing you to manually enter the data that your custom screen will eventually provide. You will need to verify that there is an attribute, entity, or relationship in the rulebase for each piece of data that your screen is meant to collect.
3. If you are using a Java IDE - set it up with a Java project and load the Web Determinations library jars as Java libraries. The Web Determinations library jars have Javadoc, which will make it much easier when using Web Determinations Java objects in the jars.
4. Create a java class for the custom screen provider in the new Java project, and implement the **CustomScreenProvider** interface and its methods in the new Class. Each time a new user begins a session, the **CustomScreenProvider** will be examining an **InterviewSession** to determine whether its custom controls are relevant to this session. More information about using rulebase model and instance data in the **InterviewSession** object can be found in [Understanding the InterviewSession](#).
5. Create a java class for the custom screen itself, and implement the **CustomScreen** interface and methods.

To install the Custom Screen:

1. Package the custom screen, custom screen provider, and any other required classes in a .jar file.
2. Deploy the jar file into the Web Determinations plugin folder, together with any other libraries that were required (except the Web Determination library jars - they are already in the 'lib' folder).
3. Restart the J2SE server to enable the new plugin.

Testing the plugin:

1. The first step to check is if it was recognized and loaded by the Tomcat server. The Web Determinations web application prints out data about plugins that were loaded into the runtime successfully. Any errors when loading a plugin is also printed out. Check the webserver's output file (for example, *stdout.txt* for Tomcat)
2. The next step is to check if the plugin is registered. The **getInstance** method will be called each time a Web Determinations interview is started (that is, rulebase and locale have been selected). If you are having trouble getting the plugin registered, ensure that your plugin **getInstance** method simply returns a new **CustomScreenProvider** object.
3. Finally, if the plugin is loaded and registering correctly, test its functionality.

Create a Custom Screen example

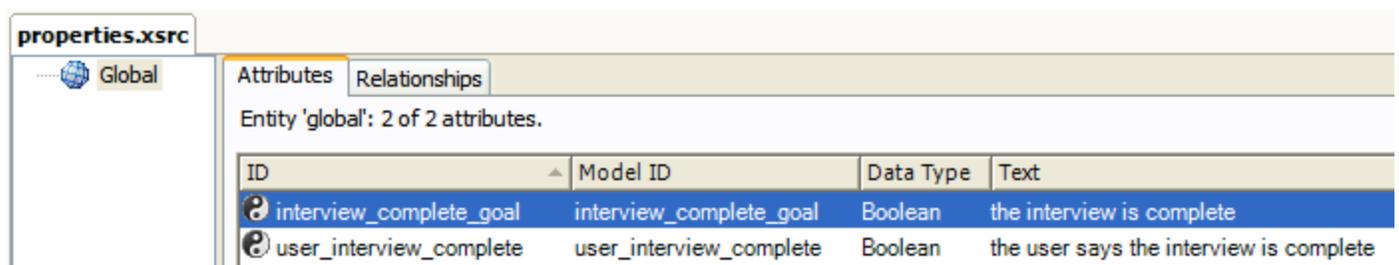
This example demonstrates how to implement a custom screen plugin for Web Determinations. For more general information on constructing and debugging custom screen plugins, see [Create a Custom Screen](#).

In our example, we will have a custom screen plugin replace a Web Determinations screen by calling out an external application.

Step 1. Build the rulebase

The first step is to create a simple rulebase with a single base-level attribute and a single question screen for that attribute.

1. In Oracle Policy Modeling, choose **File | New Project...** and name the project "TestCustomScreen".
2. Add a new properties file.
3. Add the following global attributes to the properties file:

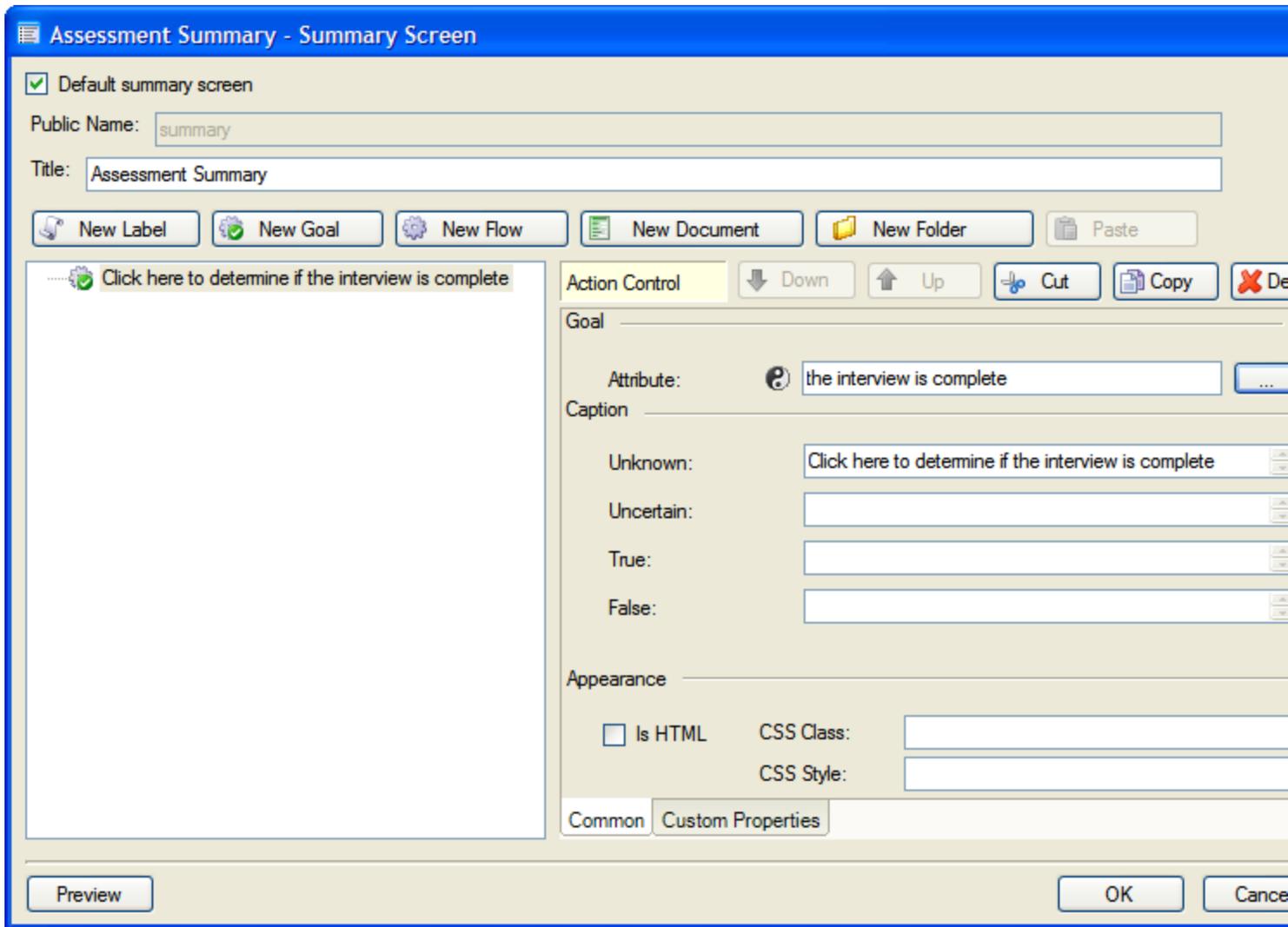


ID	Model ID	Data Type	Text
interview_complete_goal	interview_complete_goal	Boolean	the interview is complete
user_interview_complete	user_interview_complete	Boolean	the user says the interview is complete

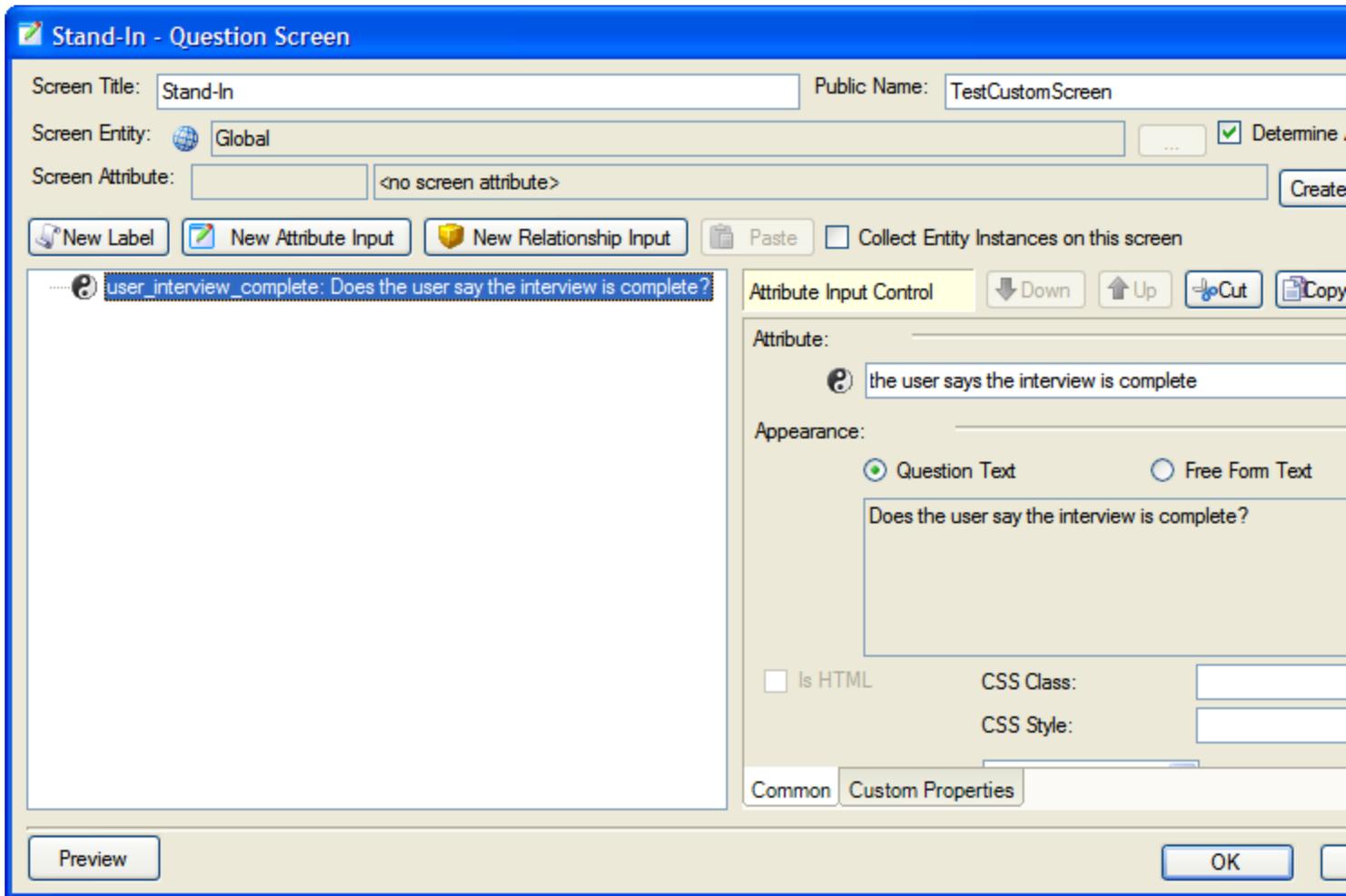
4. Add a new Word document and write the following rule:

the interview is complete if
the user says the interview is complete

5. Compile the rule and close the document, returning to Oracle Policy Modeling.
6. Add a new screens file.
7. Open the assessment summary screen, and add the "the interview is complete" attribute as the goal:



8. Also in the screens file, add a new question screen. Edit the screen to add the base-level attribute "the user says the interview is complete" as an input control:



Now we want to test that the rulebase is working correctly.

1. In Oracle Policy Modeling, choose **Build | Build and Debug | With Screens**.
2. A Web Determinations interview should open, in which you can determine if "the interview is complete" by whether the user says the interview is complete:

Data Decision Oracle Web Determinations Temporal Options... Import Export

http://localhost:9000/web-determinations9000/endsession/TestCustomScreen/en-US?user=guest&cancelURI=/scre Go

Screen: [Assessment Summary \(summary\)](#)

ORACLE Web Determinations

Data Review Save Save As Load Restart Close

Rulebase: TestCustomScreen Locale: en-US User ID: guest

Assessment Summary

- Click here to determine if the interview is complete

Data Decision Oracle Web Determinations Temporal Options... Import Export

http://localhost:9000/web-determinations9000/investigate/TestCustomScreen/en-US/Attribute~interview_complete_ Go

Investigation: [Show in decision view](#)

Screen: [Stand-In \(s2@Interviews Screens xint\)](#)

ORACLE Web Determinations

Summary Data Review Save Save As Load Restart Close

Rulebase: TestCustomScreen Locale: en-US User ID: guest

Stand-In

Does the user say the interview is complete? * Yes No

Submit



You should now have a functioning rulebase. If your rulebase is not working correctly so far, you should fix any problems before proceeding to the customization steps.

Step 2. Build a custom screen plugin

The next task is to build the plugin. Our plugin has two classes:

- *ExampleScreenProvider.java*
the class that Web Determinations will look for each time a screen is to be displayed. It determines if the screen to be displayed will be replaced by a custom screen. In our example, the Stand-In screen will be replaced by the custom screen, *ExampleScreen*, if the screen ID is "TestCustomScreen" - which is the public ID of the question screen we added.
- *ExampleScreen.java*
the custom screen itself. This custom screen redirects the user to an external web application.

Note that the Java source code is provided but the .NET source code should be very similar.

To build the custom screen plugin:

1. Copy the java files found in `<runtime_zip_dir>\examples\web-determinations\custom-screen\src` and compile them to a JAR file.
2. Tell your java compiler how to find the needed Oracle Policy Automation dependencies, as detailed in [Create a plugin](#). Call the resulting JAR file, *ExampleCustomScreen.jar*.
3. Copy the *ExampleCustomScreen.jar* file you have created to your plugins directory, where Oracle Web Determinations can find it. If your rulebase is at `<...>\TestCustomScreen`, the correct directory for your plugin is `<...>\TestCustomScreen\Release\web-determinations\WEB-INF\classes\plugins`. If the Release directory does not exist, Build and Debug your rulebase once to create it.
4. Build and Debug with Screens again and your plugin should now be loading; but it will display an error page once you click on the goal. This error is shown because the external web application where the user is supposedly to be redirected does not yet exist.

Step 3. Create the external web application

The last step is to create a simple servlet application where the user will be redirected by the custom screen. The servlet code can be found in `<runtime_zip_dir>examples\web-determinations\custom-screen\ExampleCustomScreenWebApp`. The servlet has two classes:

- *ExternalProcessingSystem.java*
the servlet that will be displayed. This screen contains two buttons: one will cause the user to return to Web-Determinations, and the other button keeps the user on the current screen while simulating passing of data to Web-Determinations.
- *web.xml*
the servlet configuration.

To create the external web application:

1. Create and test the servlet.
2. Package the application as a WAR file then deploy it to the same server where Web-Determinations is running.
3. Restart the server; your custom screen should now be working.
4. Start the interview. Notice that when you click on the goal, you are now redirected to the external web application.

Example Custom Screen

This is an example custom screen, under the control of a separate web application. If this were a real web application, it would do some work. It might get some information from the request URL, or there might be some separate back-channel communication between the external web service and the custom screen plugin in the Web Determinations application.

If this custom screen were implemented entirely as a plug-in in the web determinations application, it would have access to all of the application's session data with no extra work.

The number of times you have submitted data from this page to the Web Determinations application is 3.

Custom Screen Source Code

Examples containing the source code can be found at `<runtime_zip_dir>\examples\web-determinations\custom-screen\src`.

Create a Custom Control

The following information describes the steps for creating a custom control, installing it onto a Java webserver, and running the custom control. While this describes a Java implementation, similar steps can be followed for .Net.

For a generic background on how to construct a plugin, see [Create a plugin](#).

Assets and information needed

The following are information that will be referred to in the steps below:

Web Determinations plugin folder - this is where the Web Determinations webserver will pick up the plugins. The user needs to place the plugin packages (.jar for Java).

- J2SE(Java) - default is in the `WEB-INF\classes\plugins` path in the interactive web application in the Tomcat webapps folder, for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\interactive\WEB-INF\classes\plugins`.
- IIS (.Net) - default is in `C:\inetpub\web-determinations\plugins`.

Web Determinations templates folder - this is where Web Determinations will look for Velocity templates from which it renders the HTML code served to an end-user. Custom controls generally require a matching template to be added to this folder.

- J2SE (Java) - default is in the `WEB-INF/classes/templates` folder, for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\interactive\WEB-INF\classes\templates`.
- IIS (.Net) - default is in `C:\inetpub\web-determinations\templates`.

Web Determinations library jars

- `web-determinations.jar` - this can be found in the Web Determinations folder 'lib' folder; for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\interactive\WEB-INF\lib`.
- `determinations-interview-engine.jar` - this can also be found in the Web Determinations folder 'lib' folder; for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\interactive\WEB-INF\lib`.

Steps

To develop a custom control:

1. Determine the purpose of your control. If it is attached to a rulebase attribute (even if it is a read-only control with no input), you want an **InputCustomControl**. If it is a display which gets information from other sources, it is a plain **CustomControl**.
2. Planning -- which default controls will your custom control be replacing? How will it know? The custom control provider can use any available information for each control on each screen, to determine whether it will insert a custom control. Common techniques are to examine the control label set by the screen author in Oracle Policy Modeling (for custom controls which are meant to be used only in a single location) or custom properties set on the control (for more generic controls that could be used more than once.).
3. If you are using a Java IDE - set it up with a Java project and load the Web Determinations library jars as Java libraries. The Web Determinations library jars have Javadoc, which will make it much easier when using Web Determinations Java objects in the jars.
4. Create a java class for the custom control provider in the new Java project, and implement the **CustomControlProvider** interface and its methods in the new Class. Each time a new user begins a session, the

CustomControlProvider will be examining an **InterviewSession** to determine whether its custom controls are relevant to this session. More information about using rulebase model and instance data in the **InterviewSession** object can be found in [Understanding the InterviewSession](#).

5. Create a java class for the custom control itself, and implement the **CustomControl** or **CustomInputControl** interface and methods.
6. Create a velocity template from which the control's HTML code will be rendered.

To install the Event Handler:

1. Package the custom control, custom control provider, and any other required classes in a .jar file.
2. Deploy the jar file into the Web Determinations plugin folder, together with any other libraries that were required (except the Web Determination library jars - they are already in the 'lib' folder).
3. Restart the J2SE server to enable the new plugin.

Test the plugin:

1. The first step to check is if it was recognized and loaded by the Tomcat server. The Web Determinations web application prints out data about Plugins that were loaded into the runtime successfully. Any errors when loading a Plugin is also printed out. Check the webserver's output file (for example, *stdout.txt* for Tomcat).
2. The next step is to check if the Plugin is registered. The **getInstance** method will be called each time a Web Determinations interview is started (that is, rulebase and locale have been selected). If you are having trouble getting the plugin registered, ensure that your plugin **getInstance** method simply returns a new **CustomControlProvider** object.
3. Finally, if the plugin is loaded and registering correctly, test its functionality.

Walkthrough Examples

The [Benefit Code example](#) walks through constructing a rulebase and a custom control which allows the user to enter a single attribute using a two-part control, demonstrating how the velocity template and the custom control plugin communicate to render arbitrary HTML using data from the application.

The [Calendar Date Control example](#) walks through using an example rulebase, a custom property, a calendar written in javascript, a custom input control plugin and velocity templates to demonstrate how the date entry could be customized to use a calendar.

This [Filtered Dropdown selection list example](#) demonstrates how to create a filtered dropdown list (**FilterListCode**) which, when a user starts typing, only displays the relevant options.

Commentary plugin overview

Some screens and attributes during a Web Determinations interview are complex or ambiguous and need further explanation. Commentary functionality in Web Determinations allows the rulebase author to provide extra information about screens and attributes, which can be accessed by the user during the interview when more information or explanation is required. The commentary functionality also allows an external webpage to be displayed as commentary.

When commentary is available for a screen or attribute, the attribute/screen label is rendered as a link that the user can click on to view the commentary information. By default, commentary is displayed in a frame to the right of the interview screen. Optionally, it is possible to configure Web Determinations to display commentary information in a new window (for more about the configuration, see [Configuration files](#)). Attributes and screens do not have to have commentary, and it is possible for a Web Determinations interview to have commentary available for some screens/attributes only.

Commentary plugins are Engine Session plugins that manage commentary in a Web Determinations interview. The main functionality is to determine whether commentary for a screen/attribute is available, and to be able to provide the commentary content to those screens/attributes when requested. Commentary plugins have the flexibility of being able to retrieve content from data-sources accessible in a Java/.NET class, or returning a URL to be displayed in the commentary frame/window. The ability to return a URL allows Commentary plugins that can make use of a client's existing information web pages.

Web Determinations ships with a default Commentary plugin that uses commentary HTML files generated from Oracle Policy Modeling by the rulebase author. For more information about the default Commentary, see the topic, [Use the default Commentary](#).

Like many other plugins, only one Commentary plugin can be registered per Web Determinations interview.

Go to:

[Common scenarios](#)

[Commentary plugin and the Web Determination architecture](#)

[Developing a Commentary plugin for a specific project/implementation](#)

See also:

[Use the default commentary](#)

[Pseudo code](#)

[Sample code \(DerbyCommentary\)](#)

[Sample code \(RedirectCommentary\)](#)

[Configuration files](#)

Common scenarios

The following are common scenarios that require custom Commentary plugins:

Ability to store and use commentary content from a datasource

There may already be commentary content stored in an existing datasource (for example, a file or database). Alternatively, there may be a desire to manage the commentary information via a Content Management System or other enterprise application. A custom Commentary plugin allows a Web Determinations interview to display commentary information from any datasource (or several datasources) by retrieving data from those datasources, formatting the data into a HTML page, and displaying the page.

Ability to display other web pages as commentary

There may already be existing web pages that serve as commentary to some attributes/screens of a Web Determinations Interview. These web pages can be the client's internal web pages, or even an external page owned by another party. A custom Commentary plugin allows a Web Determinations interview to provide a URL as commentary for a screen/attribute, and the web page of the URL is displayed as the commentary.

Commentary plugin and the Web Determinations architecture

The following details how the Commentary plugin fits into the Web Determinations architecture, and how to use it in the Web Determinations environment:

How Web Determinations uses Commentary plugins by default

When a Web Determinations interview begins, Web Determinations checks for custom Commentary plugins to register. If there is a custom Commentary plugin available, Web Determinations registers that plugin for the interview session. If there are no custom plugins, Web Determinations registers the default Commentary plugin.

During the Web Determinations interview, whenever a screen and control is to be rendered and displayed, Web Determinations uses the Commentary plugin to check whether:

- Commentary is enabled for the current Web Determinations interview (based on interview session information).
- The current screen to be displayed has commentary available.
- The attribute of each control in the current screen has commentary.
- The attribute of the control has commentary available, when rendering a control.

The above allows Web Determinations to render screen and attribute labels as links. Screens/attributes that have available commentary can be clicked by the user to view commentary information, while screens/attributes without commentary are not able to be clicked.

When the user clicks on a screen/attribute label to see commentary information, Web Determinations calls the Commentary to provide commentary information. Web Determinations first checks the Commentary plugin to see if it will be returning a URL redirect or HTML content for the target (the screen/attribute that requires commentary). The Commentary plugin can read the target information provided and the current interview session data to determine whether to return a URL or HTML content.

If the Commentary plugin returns a URL for the target, Web Determinations gets the redirect URL from the Commentary plugin, and displays the web page of the URL as the commentary information.

If the Commentary plugin returns HTML content for the target, Web Determinations gets the HTML content from the Commentary plugin, and displays the HTML.

As mentioned in the overview, when the user clicks on a screen/attribute for commentary, Web Determinations will display the URL/HTML content as a frame to the right of the interview screen by default. But it is possible to configure the Web Determinations to display commentary information in a new window.

Commentary plugin and other Web Determination extensions

Other ways in which Commentary plugins can be used in Web Determinations is through other Web Determinations extensions. The Commentary plugin is accessible via the **InterviewSession** object, so only Web Determinations extensions with access to this object or its parent object **SessionContext** may use Commentary plugins.

- Because Commentary plugins return HTML (as `InputStream`) or a URL for display, only Custom Screens can be used to call Commentary plugins and be able to return commentary (return its `InputStream` as a binary `Response` or URL as a `RedirectResponse`).

- Another way of using a Commentary plugin is by using an Event Handler to handle the HTML content/URL returned by the Commentary plugin; for example, to be saved to a datasource

Error handling

As with other Web Determinations extensions, raising an exception is the common way to handle errors. Note that **RuntimeExceptions** must be raised in Java, not the generic exception.

Commentary class methods

A Commentary plugin implements the **CommentaryProviderPlugin** interface which in turn extends the **InterviewSessionPlugin** interface.

The **CommentaryProviderPlugin** interface requires the following when implemented:

Method Signature	Description
boolean isCommentaryEnabled(InterviewSession session)	<ul style="list-style-type: none"> • The InterviewSession contains information about the current Web Determinations Interview such as the locale, rulebase, instance data. • return true if the current interview session has one or more commentary during the interview, or false if there are no commentary based on the InterviewSession; for example, this plugin may have been registered based on the rulebase, but the locale used does not have commentary available.
boolean hasCommentary(InterviewSession session, String target)	<ul style="list-style-type: none"> • The InterviewSession contains information about the current Web Determinations interview such as the locale, rulebase, instance data. • The target is a string that identifies a screen/attribute. Example of 'target' values are: <ul style="list-style-type: none"> • 'attribute/child_name' for an attribute called child_name, or • 'screen/#personschildren^global' for a screen that collects instances for personschildren relationship, and the relationship's source is the entity global • Return true if there is commentary available for the target provided, false if there is no commentary available
boolean isCommentaryRedirect(InterviewSession session, String target)	<ul style="list-style-type: none"> • The InterviewSession contains information about the current Web Determinations interview such as the locale, rulebase, instance data. • The target is a string that identifies a screen/attribute. Example of 'target' values for default Web Determinations interviews: <ul style="list-style-type: none"> • 'attribute/child_name' for an attribute called child_name, or • 'screen/#personschildren^global' for a screen that collects instances for personschildren relationship, and the relationship's source is the entity global • Return true if the commentary to be returned for this target is a URL, false if the Commentary plugin will generate and return the HTML content to be displayed
String getCommentaryURL(InterviewSession session, String target)	<ul style="list-style-type: none"> • The InterviewSession contains information about the current Web Determinations interview such as the locale, rulebase, instance data. • The target is a string that identifies a screen/attribute. Example of 'target' values for default

Method Signature	Description
<pre> sion, String target) </pre>	<p>Web Determinations interviews:</p> <ul style="list-style-type: none"> • 'attribute/child_name' for an attribute called child_name, or • 'screen/#personchildren^global' for a screen that collects instances for personchildren relationship, and the relationship's source is the entity global • Returns the URL that Web Determinations should use in the commentary frame/new window.
<pre> TypedInputStream getCommentaryContent (InterviewSession ses- sion, String target) </pre>	<ul style="list-style-type: none"> • The InterviewSession contains information about the current Web Determinations interview such as the locale, rulebase, instance data. • The target is a string that identifies a screen/attribute. Example of 'target' values for default Web Determinations Interviews: <ul style="list-style-type: none"> • 'attribute/child_name' for an attribute called child_name, or • 'screen/#personchildren^global' for a screen that collects instances for personchildren relationship, and the relationship's source is the entity global • Returns the HTML content that Web Determinations should use in the commentary frame/new window.

Developing a Commentary plugin for a specific project/implementation

The following explains the various approaches on designing and developing a Commentary plugin for a specific project/implementation:

About the data and objects passed into the Commentary plugin methods

There are 2 main input arguments provided to the Commentary plugin methods: the **InterviewSession**, and a string that represents the identifier for the target screen/attribute.

The **InterviewSession** has information about the current Web Determinations interview such as the locale selected, the rulebase being used, rulebase data model, and also data provided by the user to the interview so far. Information about using/accessing the **InterviewSession** object can be found in [Understanding the InterviewSession](#)

The 'target' string identifier identifies the screen or attribute that the Web Determinations needs to know more about from a commentary perspective. Web Determinations can ask the following questions about a target screen/attribute to the Commentary plugin:

- is there commentary available for this target? (**hasCommentary** method)
- is the commentary for this target a URL or HTML content (**isCommentaryRedirect** method)
 - if URL - what is the URL for the commentary of this target (**getCommentaryURL** method)
 - if HTML content - what is the HTML content of this target (**getCommentaryContent** method)

The 'target' string is of a specific format. By default, it includes the type (attribute or screen), and the ID of the attribute/screen; for example:

- 'attribute/child_name' for an attribute called 'child_name'
- 'screen/#personchildren^global' for a screen that collects instances for personchildren relationship, and the relationship's source is the entity global

Guidelines for creating a custom Commentary plugin that returns URLs for commentary

To create a custom Commentary plugin that exclusively returns URLs for commentary:

- The **isCommentaryRedirect()** method should return false
- The **getCommentaryURL()** method should read the 'target' (and if needed, the **InterviewSession**) to determine the URL to return. From the read, it should be able to locate the URL, and return
- For simple redirect Commentary plugins, the target-URL mapping can be easily stored in a **Map** object. This approach is very quick and easy.
- For complex redirect Commentary plugins, the target-URL mapping can be retrieved from a properties file, or from an SQL database.

Guidelines for creating a custom Commentary plugin that returns HTML content for commentary

To create a custom Commentary plugin that exclusively returns HTML content for commentary:

- The **isCommentaryRedirect()** method should return true
- The **getCommentaryContent()** method should read the 'target' (and if needed, the **InterviewSession**) to be able to construct the commentary HTML content. The 'target' and possibly other data in the **InterviewSession** (for example, locale) can be used by this method to retrieve commentary data from a datasource; for example, filename, or the ID of a row in a database table.
- For simple 'HTML' Commentary plugins, the HTML content can be stored as is in the datasource; for example database content is already HTML content.
- For complex 'HTML' Commentary plugins, the Commentary plugin can retrieve commentary data, and use HTML template to insert the commentary data. This approach separates content and formatting for flexibility and maintainability.

Creating a custom Commentary plugin that returns HTML content or URL depending on the target

It is possible to create a Commentary plugin that can return HTML content or URL depending on the target (and/or **InterviewSession**). This requires the **isCommentaryRedirect()** method to be able to decide based on target and **InterviewSession** whether to return URL or HTML; this has the flow on effect that the **hasCommentary()** may also need to have separate checking mechanisms for targets that will return URL and targets that will return HTML

Use the Default Commentary

Below are steps for setting up and using the default Commentary plugin in Web Determinations:

1. The commentary HTML files is generated in Oracle Policy Modeling, and the generated text inside the HTML file/s modified by the rulebase author.
2. The rulebase package together with the commentary is deployed into Web Determinations (by default the commentary is packaged with other rulebase output files into the package).
3. Web Determinations must not have another custom Commentary plugin since the custom Commentary plugin will be registered into the interview session instead of the default Commentary plugin.

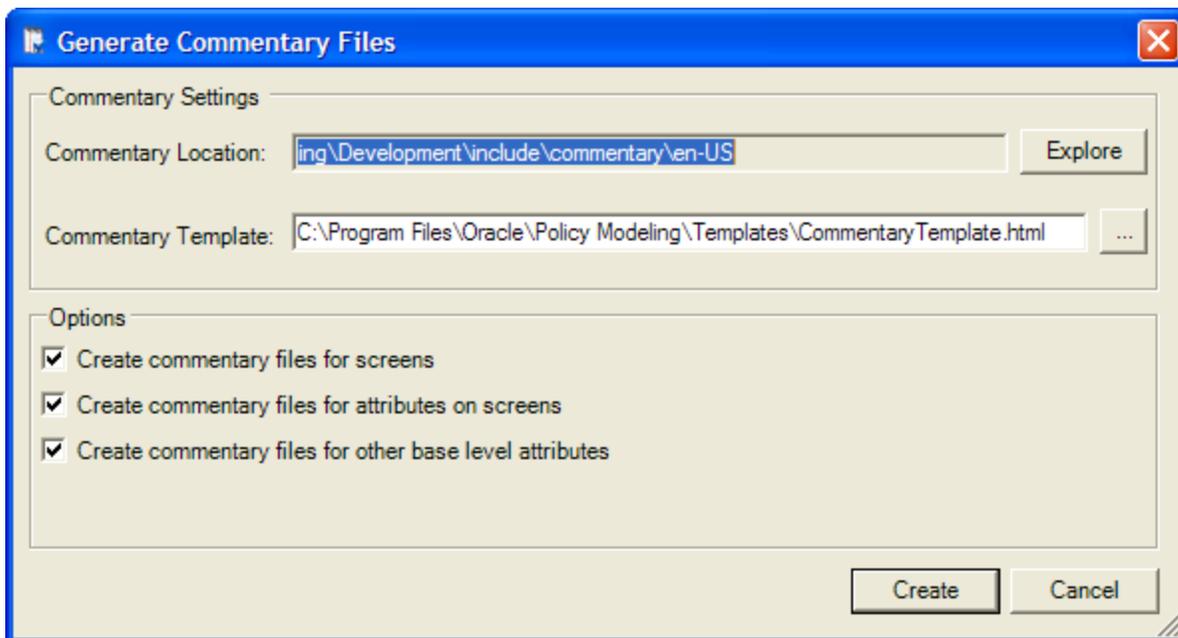
Generating the commentary HTML files

When the commentary HTML files are generated, it will be based on a template file. You can modify the default commentary template, or create your own template before generating the commentary files.

The default commentary template resides in *<OPM Installation>\Templates\CommentaryTemplate.html*; for example, *C:\Program Files\Oracle\Oracle Policy Modeler\Templates\CommentaryTemplate.html*

To generate commentary files in Oracle Policy Modeling, do the following:

1. Open the rulebase.
2. Click on the **Build** button in the menu.
3. In the *Build* submenu, click on **Generate commentary files**.
4. The *Generate Commentary Files* popup appears:



Commentary Location (readonly)

This is where the HTML commentary files will be placed; the default location is:

Rulebase Folder>\Development\include\commentary\<locale>\

for example, C:\rulebase_projects\CommentaryRulebase\Development\include\commentary\en-US\

Commentary Template

This is the location of the commentary template to use. By default it uses the default commentary template, but you can set it to a custom template that you have prepared.

Options:

Create commentary files for screens - generate commentary for screens.

Create commentary files for attributes on screens - generate commentary for attributes that will be displayed/asked in screens.

Create commentary files for other base level attributes - generate commentary for **all** attributes, regardless of whether they will appear on screen or not.

5. Click on **Create** to generate the files.

After generating the commentary HTML files, you can go to the commentary location and further modify the content of each commentary file. **Do not** change the filenames of the commentary files, only the HTML content inside. To further modify the HTML commentary files, you will need to understand how the files are identified and structured.

The following is the general folder structure inside the commentary folder:

- <locale 1>
 - attribute
 - screen
- <locale 2>
 - attribute
 - screen
- etc

HTML files inside the attribute folder are commentary files for attributes. The naming of the HTML files is straightforward; it is exactly the name of the attribute.

HTML files inside the screen folder are commentary files for screens. The naming of the HTML files for authored screens is straightforward; it uses the name of the authored screen.

You have the option of creating commentary for automatic screens, but the 'generated' names for those screens are a little more complex; for example, #personschildren^global for an 'instance collection' page, collecting the target entity 'child', for the relationship 'personschildren' with the entity global as the source.

Packaging the commentary together with the rulebase

You do not have to do anything for this step, as it happens automatically when you build the project after generating the commentary.

You must ensure that you have generated the commentary (which creates a commentary folder in the 'include' folder) before building the rulebase.

Once you have generated the commentary, building the rulebase will package all the contents of both the output and include directories into a zip file, hence the commentary folder is automatically packaged.

Ensuring the default Commentary plugin is registered

To use the default Commentary plugin, it must be the Commentary plugin registered when running the interview session.

Other custom Commentary plugin takes priority over the default Commentary plugin, so ensure there are no custom Commentary plugin that can be registered into the same interview session.

Using the Commentary during the Web Determinations interview

During the Web Determinations interview, Web Determinations will turn screen and control labels as links when there is commentary available for them. Screen and control labels will remain as text if there are no commentary available.

The user can click on the screen/control label to see the commentary available.

When the user clicks on the link, by default Web Determinations opens the commentary HTML content in a frame to the right of the interview pane.

Web Determinations can be configured to open commentary HTML in a new window; see [Configuration files](#) for information on how this is done.

Commentary plugins can also provide URL instead of HTML content. Web Determinations will display the URL webpage in the frame/new window; for more information, see [Commentary plugin](#).

Create a Commentary Plugin

The following describes the steps specific to creating a Commentary plugin. For steps on creating and installing plugins in general, refer to [Create a Plugin](#). Although this topic focuses on Java development, the same steps can be followed, for the most part, for .Net.

For more information about Commentary plugins, refer to the topic, [Commentary Plugin](#)

A common need for a custom Commentary plugin is to drive commentary data from a datasource, and custom logic to process commentary data before being displayed. A custom Commentary plugin can also use the Commentary HTML files, but perform additional processing to the HTML contents before display.

Assets and information needed

The following assets are required to develop a Commentary plugin; additionally they may be referred to in the *Steps* section below:

Web Determinations library jars

- **web-determinations.jar** - this can be found in the Web Determinations folder 'lib' folder, `<web-determinations>\WEB-INF\lib`; for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\lib`
- **determinations-interview-engine.jar***-* this can also be found in the Web Determinations folder 'lib' folder

Java IDE - this is the Java development application you are using. While this is optional, it will make it much easier when referring to objects in the *web-determinations.jar* and *determinations-interview-engine.jar* files.

Commentary plugin business case - this is the reason/need why the custom Commentary plugin is being developed.

Steps

Analysis and design:

Before starting on the development of the plugin, it is recommended that you read the [Commentary plugin](#) topic.

Once familiarized with the Commentary plugin, do the following:

1. Determine what rulebase (or rulebases) the Commentary will be registered in. You can make a custom Commentary plugin that services all the rulebases and its locales in the Web Determinations installation, or you can create the Commentary plugin to work only for a specific rulebase, or even only for one of its locales.
2. Determine if the Commentary plugin will use URL redirect, or return HTML content.
3. Ensure that the **Commentary plugin business case** addresses the following:
 - i. Will the custom Commentary plugin be driven via a datasource? (this works for both URL redirect and HTML content).
 - ii. Will the custom Commentary plugin be using the standard HTML commentary files generated in Oracle Policy Modeling?
 - iii. Does the custom Commentary plugin need to access other resources/datasources? such as templates or configuration.

4. Design how the Commentary plugin:
 - i. Will determine if commentary is enabled for a Web Determinations interview.
 - ii. Checks if commentary data is available for a target commentary item (that is, target screen or control/attribute).
 - iii. Will access datasources and resources, including transaction integrity processes, fallback on error, and so on.

Development

1. Setup the **Java IDE** so that it has the **Web Determinations library jars** and **Datasource library jars**. This makes it easier to refer to objects for the Web Determinations or the datasource
2. Start the Commentary plugin class from the following pseudo code - [Commentary - pseudo code](#)
3. Develop the Commentary plugin based on the *Analysis and design* section above.

Install and test

For steps on installing and testing the Commentary plugin, see [Create a plugin](#).

Custom Service plugin overview

The Determinations Server Custom Service plugin allows you to leverage the Oracle Policy Automation technology to create a custom SOAP/HTTP based web service. It is a high level plugin that allows custom web services to be added to the Determinations Server. It provides access to the following Determinations Server APIs:

- Interview
- Assess
- Determinations Engine.

At a high level, this plugin establishes which service a request corresponds to, then passes the processing to that request and writes out the response. Each component requiring a separate WSDL is considered a different service, and each service must have its own WSDL.

The most likely reasons for using a custom service are:

- Where there is a need to provide a customized service definition (WSDL).
- Because the new service needs to modify the default process flow of an existing service and/or actually orchestrate multiple processes to such a degree that it may be easier to implement a custom service.

Custom Service and the Determinations Server architecture

The following information details how the Custom Service plugin fits into the Determinations Server architecture, and how to use it in the Determinations Server environment.

Service plugin loading and registration.

Plugins are loaded using much the same mechanism as the one for Web Determinations (see [Plugin loading, invocation and discovery](#))

Each plugin is tied to a particular rulebase and so will be polled once for each rulebase.

If hot-swapping mode is turned on:

- When adding a new rulebase, the custom service plugin is given an opportunity to register a service against that rulebase.
- When a rulebase is deleted, any service registered against it is automatically unloaded
- Updating an existing rulebase is the equivalent of a delete and load.

Service plugin endpoints

As soon as an instance of a plugin is returned it will be asked for its endpoint. This endpoint is a URI and signifies that any request to that endpoint will be handled by this service.

- It is the responsibility of the implementer to make sure the endpoint for their service is unique.
- If multiple services use the same endpoint, the first service will be registered and all subsequent ones will be disregarded (remember you can't control the order in which plugins are loaded).
- The following endpoints are reserved for the native services:
 - `"/server/soap/*"` - Server Service(s)
 - `"/assess/soap/*"` - The Assess Service(s)
 - `"/interview/soap/*"` - The Interview Services(s)

- Service endpoints are case sensitive so /My/Custom/Service is not the same as /my/custom/service.
- **Note For .NET Plugins:**
Because ASP .NET is effectively integrated into IIS 6.0 by means of an ISAPI filter, each request must contain one of the recognized .NET extensions (for example, .aspx, .asmx) in order for it to be forwarded to the application properly. Note however that as this is automatically handled by the servlet layer, endpoints should be specified **without** any extension.

Thread safety

Since only one instance of a service plugin is registered per endpoint each service must be thread safe. Furthermore, any state that needs to be maintained by the service is entirely the responsibility of that service.

Getting the service WSDL

The service WSDL can be retrieved by appending '?wsdl' to the service endpoint. For example: if the service is registered at /my/custom/service then a call to <http://<server>/my/custom/service?wsdl> will return whatever wsdl is provided by the services' getWsdI() method.

Error handling

As the service is responsible for everything that happens at the SOAP/XML layer, it is the responsibility of the service to handle any exceptions it wishes to report in a meaningful way; for example, via a SOAP Fault.

Any unhandled exception that makes its way up to the servlet layer will be logged and cause the Determinations Server to return an HTTP 500 Internal Service Error.

Create a Custom Service

The following information describes the steps specific to creating a Custom Service plugin. For steps on creating and installing a plugin, see [Create a plugin](#). Although it focuses on Java development, the same steps can be followed for .Net (with slight implementation differences).

Assets and information needed

The following provide information that will assist in understanding what is happening in the steps below:

Determinations Server plugin folder - this is where the Determinations Server will pick up the plugins. The user needs to place the plugin packages (.jar for Java).

1. Tomcat (Java) - default is in the `WEB-INF\classes\plugins` path in the Web Determinations web application in the Tomcat webapps folder,
 - for example: `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\determinations-server\WEB-INF\classes\plugins`
2. IIS (.Net) - default is in `C:\inetpub\determinations-server\plugins`

Determinations Server library jars

- **determinations-server.jar**- this can be found in the Web Determinations folder 'lib' folder, for example: `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\determinations-server\WEB-INF\lib`
- **determinations-interview-engine.jar***-* this can also be found in the Web Determinations folder 'lib' folder, for example: `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\determinations-server\WEB-INF\lib`
- **determinations-engine.jar***-* this can also be found in the Web Determinations folder 'lib' folder, for example: `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\determinations-server\WEB-INF\lib`

Java IDE - this is the Java development application you are using. While this is optional, it will make it much easier when referring to objects in the `web-determinations.jar` and `determinations-interview-engine.jar` files.

Steps

To develop a plugin, do the following:

1. Determine what the plugin is extending (the plugin type), and the Java interface the Java plugin needs to extend. See [Plugins - general technical information](#).
2. If you are using a Java IDE - set it up with a Java project and load the Determinations Server library jars as Java libraries. The Determinations Server library jars have Javadoc, which will make it much easier when using Determinations Server Java Objects in the jars.
3. Create a Java Class for the plugin in the new Java project, and implement the required interface and its methods in the new Class.
 1. More information can be found on the specific pages of each plugin type.
 2. Technical details about plugin authoring can be found in [Plugins - general technical information](#).

To install the plugin, do the following:

1. Package the Plugin Class and any other Java Class developed with it in a .jar file.
2. Deploy the jar file into the Determinations Server plugin folder, together with any other libraries that were required (except the Determinations Server library jars - they are already in the 'lib' folder as mentioned previously).
3. Restart the Tomcat server to enable the plugin.

To test the plugin, do the following:

1. The first step to check is if it was recognized and loaded by the Tomcat server. The Web Determinations web application prints out data about plugins that were loaded successfully. Any errors when loading a plugin are also printed out. Check the webserver's output file (for example: *stdout.txt* for Tomcat).
2. The next step is to check if the plugin is registered. Depending on the plugin type - sometimes it might be loaded when the main web application root is accessed, or others when a Web Determinations interview is started (that is, rulebase and locale have been selected). If you encounter problems getting the plugin registered, ensure that your plugin **getInstance** method simply returns the plugin object, for example: "return new **MyDataAdaptor()**".
3. Finally, if the plugin is loaded and registering correctly, test its functionality.

See also:

[Determinations Server Custom Service example](#)

Events and Event Handlers

The following is intended as a reference for implementing plugin event handlers to provide custom logic in response to events being fired at defined points throughout the execution cycle in both the Web Determinations Interview Engine and Web Determinations platform.

Go to:

[Events](#)

[Platform events](#)

[Implementing Event Handlers](#)

[Do's and don'ts](#)

[Examples](#)

Events

This section describes the specific engine and platform events - providing details about the objects contained by the events (accessible and modifiable by associated event handler implementations) and the mapping between the events and the event handler interfaces to be implemented by plugin classes in order to be registered as valid event handlers for the respective events.

Interview Engine events

These events are tied to an instance of the **InterviewEngine**. They allow subscribers to be notified when a rulebase has been added, removed or otherwise modified. This is particularly useful when using a [Rulebase Resolver plugin](#) that is capable of dynamically reloading rulebases (also known as, *hotswapping*).

Event	Encapsulated objects	Description	Event Handler Interface
OnRulebaseAddedEvent	InterviewRulebase object that has just been added.	Fired immediately after the RulebaseService has notified the InterviewEngine that a rulebase has been added.	OnRulebaseAddedEventHandler
OnRulebaseRemovedEvent	InterviewRulebase object that has just been removed.	Fired immediately after the RulebaseService has notified the InterviewEngine that a rulebase has been removed.	OnRulebaseRemovedEventHandler
OnRulebaseUpdatedEvent	InterviewRulebase object that has just been updated.	Fired immediately after the RulebaseService has notified the InterviewEngine that a rulebase has been updated.	OnRulebaseUpdatedEventHandler

Interview Session events

These events are tied to an instance of the **InterviewSession**:

Event	Encapsulated objects	Description	Event Handler Interface
OnSessionCreatedEvent	<ul style="list-style-type: none"> The InterviewSession object that has just been created. 	<p>Fired immediately after an interactive session is created, with the InteractiveEngine object as the sender. This is the chance to inject data into the session or perform any other custom initialization logic required.</p>	OnSessionCreatedEventHandler
BeforeSessionDestroyedEvent	<ul style="list-style-type: none"> The InterviewSession object about to be destroyed. 	<p>Fired just before a Web Determinations session is to be destroyed, with the InterviewSession as the sender. This is the last chance to extract the session data out of the session and persist it somewhere, clean up any resources created/opened during the session's lifetime, and so on.</p>	BeforeSessionDestroyedEventHandler
OnValidateControlEvent	<ul style="list-style-type: none"> The InputControlInstance that has just been validated. The entity instance on which the control's attribute is located. The data model encapsulating the entity instances that currently exist on the current Web Determinations session. An object encapsulating the current status of the data submission transaction into the Web Determinations session. This object contains the errors, warnings and events raised dur- 	<p>Fired immediately after an input control is validated by the core validation logic, with the Web Determinations Engine ControlInstance as the sender. Custom validation may be applied here and errors and/or warnings attached to the transaction result object.</p>	OnValidateControlEventHandler

	<p>ing the submission process.</p>		
OnValidateScreenEvent	<ul style="list-style-type: none"> • The InterviewScreen that has just been validated. • The data model encapsulating the entity instances that currently exist on the current Web Determinations session. • An object encapsulating the current status of the data submission transaction into the Web Determinations session. This object contains the errors, warnings and events raised during the submission process. 	<p>Fired immediately after an Interview screen is validated by the core validation logic, with the Web Determinations Engine InterviewScreen object as the sender. Custom validation may be applied here and errors and/or warnings attached to the transaction result object.</p>	OnValidateScreenEventHandler
BeforeSubmitDataEvent	<ul style="list-style-type: none"> • The data model encapsulating the data to be submitted into the Web Determinations session. • An object encapsulating the current status of the data submission transaction into the Web Determinations session. This object contains the errors, warnings and events raised during the submission process. 	<p>Fired immediately before data (encapsulated in an InterviewUserData object instance) is to be submitted into an Web Determinations session, with the InterviewSession object as the sender. Event handler implementations that handle this event have the ability to cancel the submission by appending an error to the transaction result object.</p>	BeforeSubmitDataEventHandler

OnCommitEvent	<ul style="list-style-type: none"> • The data model encapsulating the data that has just been committed into the Web Determinations session. • An object encapsulating the current status of the data submission transaction into the Web Determinations session. This object contains the errors, warnings and events raised during the submission process. 	<p>Fired immediately after data submitted into an Web Determinations session is committed to the rulebase session, with the InterviewSession object as the sender. If this event fires, it means that the data submitted has been attached to the underlying rule session and is safe to work with, persist, export, and so on. Some sort of auto-save functionality could be implemented as a handler of this event.</p>	OnCommitEventHandler
OnSubmitRollbackEvent	<ul style="list-style-type: none"> • The data model encapsulating the data submitted during the submission operation. • An object encapsulating the current status of the data submission transaction into the Web Determinations session. This object contains the errors, warnings and events raised during the submission process. 	<p>Fired immediately after a failed submit data operation was rolled back, with the InterviewSession object as the sender. If this event fires, it means that the data submitted has not been committed to the underlying rule session</p>	OnSubmitRollbackEventHandler
OnSubmitDataEvent	<ul style="list-style-type: none"> • The data model encapsulating the data submitted during the submission operation. • An object encapsulating the current 	<p>Fired immediately after the submit operation, regardless of whether the operation was successful and the data submitted was committed or the operation failed and the operation was rolled back.</p>	OnSubmitDataEventHandler

	<p>status of the data submission transaction into the Web Determinations session. This object contains the errors, warnings and events raised during the submission process.</p>	<p>Note that this event will not fire if the transaction result contained errors immediately after the call to handleEvent on all the BeforeSubmitDataEventHandler implementations - the submission operation is aborted in this case and the BeforeSubmitDataEvent is the only submission event fired.</p>	
OnGenerateDocumentEvent	<ul style="list-style-type: none"> • The document type of the document being generated • An object encapsulating the document-generation parameters, and attributes for Decision Reports of this specific document-generation action. 	<p>Fired when the user submits a document generation action. The event handler receives the document type and other various document-generation parameters in the event object. The event handler also has access to the InterviewSession object from the sender input argument. The event handler has the option of generating a document of its own and providing it to the event object (as a 'replacement document'). The Web Determinations Server will use the replacement document and bypasses normal Document Generation steps.</p>	OnGenerateDocumentEventHandler

Interview Portlet events

These events allow developers to customize the behavior of the Interview Portlet:

Event	Description	Event Handler Interface
OnPortletRequestEvent	<ul style="list-style-type: none"> • This event fires when the interview portlet receives either a doView, processAction or serveResource request and provides access to the PortletRequest object. It is fired immediately prior to request being processed by Web Determinations. • The PortletRequest object can be useful, for example, for adding additional parameters to the portlet session. 	OnPortletRequestEventHandler

OnPortletWriteResponseEvent	<ul style="list-style-type: none"> This event provides access to the PortletResponse and Web Determinations Response objects. It is fired after the Portlet / Web Determinations has processed the request but prior to any response being written to the response stream. This event allows the developer to modify the Web Determinations Response object, for example, to add a custom heading to a page programmatically. 	OnPortletWriteResponseEventHandler
OnDecodeParamEvent	<ul style="list-style-type: none"> This event provides access to the parameters contained in PortletRequest.getParameterMap(), just before these parameters are URLUTF8-decoded and passed to Web Determinations. This event allows the developer to modify the parameter name and value by using the setParamName and setParamValue methods. <p>Note: You can find a detailed Javadoc API of the portlet event in the Oracle Policy Automation installation's help\api directory.</p>	OnDecodeParamEventHandler
OnAfterProcessActionEvent	<ul style="list-style-type: none"> This event is fired after the Portlet / Web Determinations has processed the request in the action phase but prior to the portlet entering the render phase. The event provides access to the portlet's ActionRequest and ActionResponse objects, as well as the Web Determinations Session Context and the Response about to be rendered. Third parties can use the ActionResponse to, for example, send events to external portlets. 	OnAfterProcessActionEventHandler

Web Determinations servlet events

These events are tied to an instance of the Web Determinations **WebDeterminationsServlet**. They are designed to provide access to the native **HTTP Request**, **Response**, and **Session** objects. Unlike other events the encapsulated objects differ with regards to the .NET and Java implementations.

Event	Encapsulated objects	Description	Event Handler Interface
-------	----------------------	-------------	-------------------------

OnRequestEvent	<p>Java</p> <ul style="list-style-type: none"> • The HttpServletRequest being processed. • The SessionContext relating to the current request • A string indicating the type of request currently being processed. This will either be either "GET" or "POST". 	Fired on receipt of either a HTTP Get or Post request, immediately prior the request being processed by Web Determinations.	OnRequestEventHandler
	<p>.NET</p> <ul style="list-style-type: none"> • The HttpRequest being processed. - The HttpSessionState associated with this request. • The SessionContext relating to the current request • A string indicating the type of request currently being processed. This will either be either "GET" or "POST". 		
OnWriteResponseEvent	<p>Java</p> <ul style="list-style-type: none"> • The HttpServletResponse that the response will be written to • The Response to be written • The WebDeterminationsServletContext associated with this event 	Fired after Web Determinations has processed the request but prior to any response being written to the response stream.	OnWriteResponseEventHandler
	<p>.NET</p> <ul style="list-style-type: none"> • The HttpResponse that the response will be written to • The Response to be written • The WebDeterminationsServletContext associated with this event 		

Web Determinations session events

These events are tied to an instance of the Web Determinations **SessionContext**:

Event	Encapsulated objects	Description	Event Handler Interface
OnInvestigationStartedEvent	<ul style="list-style-type: none"> • The SessionContext of 	Fired immediately after an invest-	OnInvestigationStartedEventHandler

	<p>the current user session.</p> <ul style="list-style-type: none"> • The GoalInstance object encapsulating the goal to be investigated 	<p>igation on a particular goal attribute is started.</p>	
OnInvestigationEndedEvent	<ul style="list-style-type: none"> • The SessionContext of the current user session. • The GoalInstance object encapsulating the goal that has just been investigated. 	<p>Fired immediately after an investigation on a particular goal attribute is concluded (that is, the goal becomes known).</p>	OnInvestigationEndedEventHandler
OnGetScreenEvent	<ul style="list-style-type: none"> • The Platform Screen object of the screen to be rendered for display response 	<p>Fired after the Platform Screen is generated from the InterviewScreen, and before it is rendered to HTML. The Screen object can be a native or custom screen. The sender object is the SessionContext object for the current session.</p>	OnGetScreenEventHandler
OnRenderScreenEvent	<ul style="list-style-type: none"> • The string HTML of the rendered Platform Screen object for display response 	<p>Fired after the HTML is generated from the PlatformScreen object, and before the HTML is set back as a Response. The sender object is the SessionContext object for the current session.</p>	OnRenderScreenEventHandler

<p>OnSaveEvent</p>	<ul style="list-style-type: none"> • The Ses-sionContext of the current user session. 	<p>Fired when the user clicks on the Save button to save the current interview data, and before the DataAdaptor 'save' is called. The handler can provide a Case ID for the save - poor man's data adaptor, and a quick way to auto-generate CaseID for save (avoiding the 'Get CaseID from user' screen').</p> <p>The handler can also raise an error and halt the save process.</p>	<p>OnSaveEventHandler</p>
<p>OnRequireSessionEvent</p>	<ul style="list-style-type: none"> • The Ses-sionContext of the current user session • The request URI object of the current Request 	<p>Fired if the current Interview request requires an InterviewSession to have already been started, and the current InterviewSession is null. The handler can provide the Inter-viewSession to be used by the Interview request. The handler can use the URI object, which contains various details about the current request such as the rulebase, locale, request action, and so on.</p> <p>Note: This event will not fire on a StartSession request since having</p>	<p>OnRequireSessionEventHandler</p>

		an existing active session is not a prerequisite for starting a new session.	
OnApplyTemplatesEvent	<ul style="list-style-type: none"> • The ScreenTemplate to be used to render the screen. • The StyleTemplate to be used to style the screen. • The TemplateContext to which the templates are to be applied 	The event that fires before the Velocity template is applied to the screen. This method can be used to push additional parameters to the velocity template or modify the template used to render the screen.	OnApplyTemplatesHandler
OnInterviewSessionCreatedEvent	<ul style="list-style-type: none"> • The SessionContext of the current user session • The request URI object of the current Request • The newly created InterviewSession 	Fired immediately after a new InterviewSession is created in Web Determinations. It provides the opportunity for handlers of this event to perform additional post session configuration such as load reference data. Additional parameters that are passed in on the query string to the StartSession Action can be accessed via the provide URI object. The sender object is the SessionContext object	OnInterviewSessionCreatedEventHandler

Implementing Event Handlers

Along with the requirements placed upon plugin implementations, event handler implementations must conform to the following:

- The **handleEvent(event : Event, sender : Object) : void** method must be implemented. When an event of the type (s) that the event handler implementation is a handler of is fired, the event handler implementation's **handleEvent** method is called, with the fired event and the object in which this event was fired as the parameters.
Note: One event handler implementation may implement multiple event handler interfaces, marking itself as an event handler for multiple event classes. In this case, it is the responsibility of the event handler implementation to check the runtime type of the **event** parameter and casting it as necessary before determining what to do with it.

Do's and don'ts

- It is possible to register multiple handlers for the same instance of an event, however, no assumptions can be made about the order in which those events will be called.
- It is not permissible for event handlers alter the state of the sender object.
- Event handlers are not permitted to alter the state of the system, unless the event itself provides provision for doing so; for example, the **OnValidateControl** event provides the ability to perform additional validation and mark a control as being valid or invalid when it might not have otherwise been the case.

Examples

Follow the links below to view the event handler examples contained in the *Oracle Policy Automation Developer's Guide*:

Interview Engine

[Custom Control Validator](#)

[Custom Screen Validator](#)

Web Determinations

[Plugins - Data Adaptor - sample code \(Autosave with Derby\)](#)

[Example: Use the OnInterviewSessionCreatedEvent to pre-seed data into a newly created session](#)

Interview Portlet

[Example: Encode the Interview Portlet's response](#)

[Example: Extract the username in the Interview Portlet](#)

[Example: Send external events to other portlets](#)

Create an Event Handler

The following steps describe how to create an Event Handler, install it onto a Java webserver, run the Event Handler, and debugging tips. While this relates specifically to Java, similar steps can be followed for .Net.

Note that because an Event Handler is technically also a plugin, many of the steps are very similar to creating Plugins; the only real difference is that there are many different Event Handlers, called at different times during a Web Determinations interview. For more information, see [Create a plugin](#).

Assets and Information needed

The following information will be referenced in the steps below:

Web Determinations plugin folder - this is where the plugin packages (.jar for Java) are stored and from where Web Determinations webserver will pick them up. The user needs to place the:

1. **Tomcat (Java)** - default is in the `WEB-INF\classes\plugins` path in the Web Determinations web application in the Tomcat webapps folder; for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\classes\plugins`.
2. **IIS (.Net)** - default is in `C:\inetpub\web-determinations\plugins`.

Web Determinations library jars

- *web-determinations.jar* - this can be found in the Web Determinations folder 'lib' folder; for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\lib`.
- *determinations-interview-engine.jar**-*this can also be found in the Web Determinations folder 'lib' folder; for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\web-determinations\WEB-INF\lib`.

Java IDE - this is the Java development application you are using. While this is optional, it will make it much easier when referring to objects in the *web-determinations.jar* and *determinations-interview-engine.jar*.

Steps

Develop an Event Handler:

1. Determine which Event the Event Handler will catch; information about the different Events and Event Handlers in Web Determinations can be found in the topic, [Events and Event Handlers](#).
2. If you are using a Java IDE - set it up with a Java project and load the Web Determinations library jars as Java libraries. The Web Determinations library jars have Javadoc, which will make it much easier when using Web Determinations Java objects in the jars.
3. Create a Java Class for the Event Handler in the new Java project, and implement the required Event Handler interface and its methods in the new Class.
 - a. More information can be found for a specific Event in [Events and Event Handlers](#).
 - b. More information about using rulebase model and instance data in the **InterviewSession** object can be found in [Understand the InterviewSession](#).

Install the Event Handler:

1. Package the Event Handler class and any other Java class developed with it in a .jar file.
2. Deploy the jar file into the Web Determinations plugin folder, together with any other libraries that were required (except the Web Determination library jars - they are already in the 'lib' folder as mentioned previously).
3. Restart the Tomcat server to enable the Event Handler.

Test the plugin:

1. Check that it was recognized and loaded by the Tomcat server. The Web Determinations web application prints out data about plugins (Event Handlers) that were loaded into the runtime successfully. Any errors when loading a plugin are also printed out. Check the webserver's output file (for example, *stdout.txt* for Tomcat).
2. The next step is to check if the plugin is registered. Depending on the plugin type - sometimes it might be loaded when the main web application root is accessed, or others when a Web Determinations interview is started (that is, rulebase and locale has been selected) If you are having troubles getting the plugin registered, ensure that your plugin **getInstance** method simply returns the plugin object.
3. Finally, if the plugin is loaded and registering correctly, test its functionality.

Batch Processor

The Batch Processor allows a large number of cases to be processed in batch. It is available in both Java and .NET implementations to enable support for platform specific custom functions. It uses a variety of techniques to maximize throughput for both csv and database connections.

Batch Processor inputs

The batch processor can take comma separated files as well as database tables via JDBC/ADO.NET. Inputs can be easily configured to include multiple entities and relationships.

Batch Processor outputs

The batch processor can output the results of a batch run to csv or database (see [Batch Processor output](#)). It can also be used to output coverage, test scripts, or what-if analysis for Oracle Policy Modeling.

Topics in "Batch Processor"

- [Run the Batch Processor](#)
- [CSV input for the Batch Processor](#)
- [Database input for the Batch Processor](#)
- [Batch Processor output](#)
- [Configure the Batch Processor](#)
- [Use log messages in the Batch Processor](#)
- [Find out what SQL queries are being used by the Batch Processor](#)
- [System properties and the Batch Processor](#)
- [Batch failure and recovery](#)
- [How do I identify my Batch Processor input as database tables?](#)
- [How do I identify my Batch Processor input as csv files?](#)
- [How do I map Batch Processor data to a global boolean format?](#)
- [How do I map Batch Processor input data to a rulebase entity?](#)
- [How do I map Batch Processor input data to a rulebase relationship?](#)
- [How do I map Batch Processor input data to a rulebase attribute?](#)
- [How do I migrate my Data Source Connector project to a Batch Processor project?](#)
- [Example: Create test cases in the Batch Processor](#)
- [Example: Run the Batch Processor \(InsuranceFraudScore\)](#)
- [Example: Using the Batch Processor with a database](#)
- [Known issues for the Batch Processor](#)

Run the Batch Processor

The batch processor is a standalone application that is run from the command line using the parameters listed below:

```
java -jar determinations-batch.jar <command line parameters>
```

```
Determinations.Batch.exe <command line parameters>
```

Batch Processor parameters can be set from the command line, or by a configuration file that can be specified using the **--config** parameter, or will be automatically picked up if it is named *config.xml* in the Batch Processor's working directory.

For details on command line parameters see: [Command line configuration](#)

For details on providing parameters via an configuration file, see [XML file configuration](#).

For an example of running the Batch Processor, see [Example: Run the Batch Processor \(InsuranceFraudScore\)](#).

CSV input for the Batch Processor

The Batch Processor is designed to be easy to configure and use. If comma separated files are used as input, then little or no configuration is needed to get the Batch Processor to correctly read them in and output the correct results. To achieve this, a set of *zero-configuration* conventions are used for identifying entities, attributes and relationships.

Zero-configuration conventions for csv input

CSV file name used as Entity public name.

When reading in csv files, unless otherwise specified, the file name (without the .csv extension) is used to match an entity public name, in the rulebase.

For example, if you have a csv file with the name *person.csv*, and your rulebase has an entity with the public name *person*, the Batch Processor will automatically load data from this file into the *person* entities when processing.

Column headings used as attribute public names

The first line in a csv file should contain the column headings for the data in subsequent lines. These column headings will be matched to attribute public names.

For example, in the csv text below, the headings "name" "age" and "date_of_birth" will be matched to attributes with the corresponding names.

```
#,name,age,date_of_birth
1,John Citizen,30,1982-02-13
```

Column with name "#" as identifier

By default, a column with the heading "#" is assumed to be a unique identifier for that entity. Only the identifier for an entity (row) needs to be unique.

Example:

In the example data below, the entity Bank has a unique identifier row ("#").

```
Bank.csv
#, bank_name
1, First National Bank
2, Second National Bank
3, Third National Bank
```

Column headings used for relationships

One-to-many relationships can be represented in a csv file. If a column heading is matched to the public name of a relationship, the relationship will be loaded.

Example

In the example data below, the "to-one" side of the "one-to-many" relationship can be represented, by putting the the id of the bank in a column called "customers_bank". This will be correctly read in by the Batch Processor if the customer to bank side of the

relationship has the public name "customers_bank"

Bank.csv

#, bank_name

1, First National Bank

2, Second National Bank

Customer.csv

#, customer, balance, customers_bank

1, John Citizen, 3000, 1

2, Joan Citizen, 1000, 2

CSV file name used as many-to-many relationship

A many-to-many relationship will be identified if the name of the csv file matches the public name of the many-to-many relationship. The csv file should contain two fields which are the source and target entities of the relationship.

Example

If a rulebase has a many-to-many relationship "the customer's products" with the public name "customers_products", the many-to-many relationship will be read in as:

- Customer 1 targets: Products A, B, C
- Customer 2 targets: Products C, D
- Customer 3 targets: Products A, B

Customer.csv

#, customer

1, John Citizen

2, Joan Citizen

3, Fred Bloggs

Product.csv

#, product

1, Product A

2, Product B

3, Product C

4, Product D

customers_products.csv

Customer, Product

1, 1

- 1, 2
- 1, 3
- 2, 3
- 2, 4
- 3, 1
- 3, 2

Value formats for csv inputs

Attribute values specified in a csv file can be mapped on any attribute in the corresponding entity. Any column in a csv file that cannot be mapped is ignored (a warning is written to the log files).

The expected formats for various attribute values are:

Value type	Format description	Blank Value
Number / Currency	<p>The Batch Processor treats currency values as identical to simple numeric values when reading input and writing output. For numeric and currency values, it will apply the following rules:</p> <p>For character fields:</p> <ul style="list-style-type: none"> • Must use the '.' character as the decimal separator. • Thousands separators are not supported and must not be used. • Currency symbols are not supported and must not be used. • For scientific notation, the '+' character is not supported in the exponent. 	Blank and NULL values are considered UNCERTAIN
Text	Text values will be read and written verbatim.	Blank text values are treated as empty strings.
Boolean	<p>For reading and writing boolean values, the Batch Processor will apply the following rules:</p> <p>Reading input values:</p> <ul style="list-style-type: none"> • For character fields: <ul style="list-style-type: none"> ◦ String values are case insensitive, so "YES" is the same as "yes" and "Yes". ◦ White space characters will be ignored, so "YES" is the same as "YES". ◦ "true", "yes" and "1" will be read as TRUE. ◦ "false", "no" and "0" will be read as FALSE. <p>Writing output values:</p> <ul style="list-style-type: none"> • The values will be "true" and "false". 	Blank and NULL values are considered UNCERTAIN

Value type	Format description	Blank Value
Date	<p>For reading and writing date values, the Batch Processor will apply the following rules:</p> <p>For character fields:</p> <ul style="list-style-type: none"> • Input values must be provided in the format "yyyy-MM-dd". • Output values will be written in the format "yyyy-MM-dd". 	Blank and NULL values are considered UNCERTAIN
Date-Time	<p>For reading and writing date-time values, the Batch Processor will apply the following rules:</p> <p>For character fields:</p> <ul style="list-style-type: none"> • Input values must be provided in the format "yyyy-MM-dd HH:mm:ss", in 24-hour time. • Output values will be written in the format "yyyy-MM-dd HH:mm:ss", in 24-hour time. 	Blank and NULL values are considered UNCERTAIN
Time of Day	<p>For reading and writing time values, the Batch Processor will apply the following rules:</p> <p>For character fields:</p> <ul style="list-style-type: none"> • Input values must be provided in the format "HH:mm:ss", in 24-hour time. • Output values will be written in the format "HH:mm:ss", in 24-hour time. 	Blank and NULL values are considered UNCERTAIN

See also:

[Value formats for csv output](#)

Database input for the Batch Processor

The Batch Processor supports the reading of input data from database tables and the writing of the results to database. However, the following limitations should be noted:

- Java Batch Processor only - the .NET Batch Processor does not support database reading and writing.
- Output fields must be columns in the same table as input fields.

Supported Databases

Database connectivity is implemented using JDBC and so any database that provides JDBC drivers can be used with the Batch Processor. Oracle Policy Automation supports the following databases, for which the *OPA types* are described in the tables that follow:

- Oracle 10g, 11g
- SQL Server 2008

Oracle

OPA type	Compatible Oracle (11g) Database data types	Formatting Notes
boolean	number, any text type	For number and char data types: 0 = false, 1 = true. Custom format available, see "boolean-format" in XML file configuration .
date	date, Timestamp, any text type	When reading or writing date attributes, time and time zones are ignored. Oracle Policy Automation date attributes will have time removed when they are read in. For text fields, the format must be: "yyyy-mm-dd" (see Value formats for csv input).
datetime	date, Timestamp, any text type	Datetime attribute will be resolved to seconds (milliseconds or smaller will be truncated). For text fields, the format must be: "yyyy-mm-dd HH:mm:ss" (see Value formats for csv input).
text	any text field (char, text, nchar, nvarchar and so on)	
time of day	date, Timestamp, any text type	Time of day only supports a string value formatted as "HH:mm:ss" (see Value formats for csv input).
number and currency	any number (float, decimal, bigint, int, smallint and so on)	Number fields are rounded to accuracy of 15 decimal places.

SQL Server

OPA Type	Compatible SQL Server (2008 R2) data types	Formatting Notes
boolean	bit, number, time, any text	Bit data type supported.

OPA Type	Compatible SQL Server (2008 R2) data types	Formatting Notes
	type	For number and char data types: 0 = false 1 = true. Custom format available, see "boolean-format" in Data mapping in the XML configuration file .
date	date, datetime smalldatetime, datetime2, any text type	Any datetime field read in as an Oracle Policy Automation Date will have the time component removed. "yyyy-mm-dd" (see Value formats for csv input).
datetime	datetime, smalldatetime, datetime2, any text type	For all datetime data types, milliseconds will not be read in. Oracle Policy Automation does not support milliseconds in date or time values. "yyyy-mm-dd HH:mm:ss" (see Value formats for csv input). Note: the datetimeoffset data type is not supported in this release.
text	all text fields (char varchar, varchar2 and so on)	
time of day	time, any text types	Time of day only supports a string value formatted as "HH:mm:ss" (see Value formats for csv input). For time data type, milliseconds will not be read in. Oracle Policy Automation does not support milliseconds in date or time values
number and currency	any number, money, any text type	Number fields are rounded to accuracy of 15 decimal places.

See also:

[Value formats for database output](#)

Batch Processor output

Go to:

[Value formats for csv output](#)

[Value formats for database output](#)

[Coverage output](#)

[Test case output](#)

[Sessions output](#)

[Include inferred entities in Batch Processor output](#)

Value formats for csv output

When the Batch Processor writes out csv output it writes them in the following manner:

- Input values (those provided in the input csv) are written as they were provided (that is, input format is unchanged)
- Output values are written in a format depending on their value type; the formats are outlined in the following table:

Value Type	Output Format	Uncertain or Unknown
Number and Currency	Numeric values written as decimal numbers with a period as the decimal separator. The number is formatted up to 15 significant decimal places; for example: 3.123456789876543	Both written as blank
String	String values are written as is.	Both written as blank
Boolean	Boolean are always written as "true" or "false"	Both written as blank
Date	Date values are always written in the format "yyyy-MM-dd" where: <ul style="list-style-type: none">• yyyy is the four-digit year.• MM is the two-digit month, including leading zero for values below 10.• dd is the two-digit day, including leading zero for values below 10.	Both written as blank
DateTime	Date time are always written in the format "yyyy-MM-dd HH:mm:ss" where: <ul style="list-style-type: none">• yyyy is the four-digit year.• MM is the two-digit month.• dd is the two-digit day.• HH is the two-digit 24-hour hour value.• mm is the two-digit minute value.• ss is the two-digit seconds value.	Both written as blank

Value Type	Output Format	Uncertain or Unknown
Time	Time are always written in the format "HH:mm:ss" where: <ul style="list-style-type: none"> • HH is the two-digit 24-hour hour value. • mm is the two-digit minute value. • ss is the two-digit seconds value. 	Both written as blank

Value formats for database output

Batch Processor results are written to the database using JDBC. It is recommended that where possible, the output column in the database matches the Oracle Policy Automation attribute (date and datetime, numbers and so on).

When writing date or datetime attributes to the database, the resulting value will depend on the data type of the table column to which it is being written:

- If an Oracle Policy Automation date or datetime attribute is written to a text field, the resulting value will depend on the default formatting for dates and times for that database; the resulting value will be the date, formatted by the database as a string. The formatting of this date must be controlled by the formatting in the database; it is recommended that you consult your database documentation for further information.
- If an Oracle Policy Automation number or currency value is written to a text field, the resulting value will, as mentioned above, depend on database formatting.

For boolean values, a global boolean format is provided to control the value written to the database; see the topic [Specify the global boolean format](#).

Null values are always treated as uncertain; except for string types which will produce output of *empty string*.

Coverage output

When the output of the batch processor is set to coverage (**--coverage <coverage file>**) the result is a single file which can be imported into Oracle Policy Modeling as follows:

1. Select **Analyze coverage file** from the Reports menu.
2. Select the coverage file generated by the batch processor and click on the **Open** button; the coverage file will be imported into Oracle Policy Modeling.

For more information see the *Oracle Policy Modeling User's Guide*.

Test case output

When the output of the Batch Processor is set to test cases (**--exporttsc <filename>**), the result is a single file which can be imported into Oracle Policy Modeling as follows:

1. In the Project explorer, right-click and choose "Import existing file".
2. Select the test case file generated by the Batch Processor and click on the **Open** button; the test cases file will be imported into Oracle Policy Modeling.

For more information see the *Oracle Policy Modeling User's Guide*.

Sessions output

When the output of the Batch Processor is set to sessions (`--export <export dir>`) the result is a set of OPA sessions, generated as XML files. There will be one session file generated for each case.

A session XML file can be used in Oracle Policy Modeling debugger. To load an individual session file into an active debugger session in Oracle Policy Modeling as follows:

1. Start the debugger.
2. In the Data tab of the debugger click on the **Import** button.
3. Select the session file to import and click on **OK**; the session data will be imported into the active debugger session.

Include inferred entities in Batch Processor output

The Batch Processor supports the ability to include inferred entities as output when writing to DB tables or CSV data files. The output will include the containment relationships of the inferred entities, but will not include any other inferred relationship details. Inferred entity output is only supported for database output or test script output; other output types such as session data, do not include inferred entities.

Output mapping

To include instances of an inferred entity in the output, the CSV input directory must include a data file that the Batch Processor can match to the inferred entity. The data file mapping rules are the same as those for non-inferred entities:

- Using an XML configuration file, an inferred entity name can be explicitly mapped to a CSV data file by using a `<mapping>` element.
- If an explicit mapping is not provided, the inferred entity name is implicitly mapped to a CSV data file matching the public name of the inferred entity.

As with non-inferred entities, implicit mappings are supported when reading from CSV data files. When reading from a database, a configuration file with explicit mappings is required. If no explicit or implicit mapping can be found for an inferred entity, instances of the inferred entity will not be included in the output.

Output assumptions

When writing inferred entity output, the Batch Processor will apply the following assumptions:

Assumption	
The output destination is empty	<p>Before the Batch Processor is run with inferred entity output, the output destination for the inferred entity output should be empty</p> <ul style="list-style-type: none">• For CSV output, the input file is used to define the structure of the output file only. Any data rows in the input file will be ignored, and will not be included in the output file.• For DB output, the Batch Processor will attempt to insert the inferred entity instances as new rows in the database table.<ul style="list-style-type: none">◦ If the Batch Processor is generating the primary key values and the output table is not empty, key violations may occur. If a key violation occurs, the entire block of case output will be rolled back and the Batch Processor will stop.

Assumption	
	<ul style="list-style-type: none"> ◦ If the database is generating the primary key values and the output table is not empty, the new output rows will be inserted with the existing output rows from previous runs of the batch processor. The data written out does not include an easy way of separating the output from multiple Batch Processor runs.
All output attributes are identified	<p>Every attribute to be included in the output of an inferred entity must be identified as an output attribute. Attributes identified as input, or not identified in the implicit or explicit mapping will be excluded from the output.</p> <ul style="list-style-type: none"> • For implicit mappings, the column names, including the '#' identity column, must be marked as output using parenthesis characters; for example: (column_name) • For explicit mappings, the field configuration must include the output="true" attribute; for example: <code><attribute name="attr_1" field="field_1 output="true" /></code>

Generating Primary Key values

When writing inferred entity output, the Batch Processor is creating new rows in the table which require unique primary key values. If writing to a database table that includes an auto-generating primary key, setting the primary-key-auto attribute to true in the entity configuration instructs the Batch Processor to allow the database to provide the primary key values when new rows are inserted:

```
<mapping entity="benefit" table="benefit-mapped" primary-key="#" primary-key-auto="true"> <attribute ... /></mapping>
```

If the **primary-key-auto** attribute is set to false, is excluded, or the Batch Processor is writing output to CSV files, then the Batch Processor will attempt to generate the primary key values. If the generated key causes a key violation with existing data when writing new rows with generated keys, the entire block of cases will be rolled back and the Batch Processor will stop and report the error.

Examples

The following are simple examples of how instances of the inferred Benefit entity can be included in the output, using explicit configuration and implicit configuration. The examples are based on the *Inferred Benefits* example rulebase provided with Oracle Policy Automation and located at: [examples\rulebases\InferredBenefits.zip](#).

Explicit configuration

The CSV input directory contains a CSV file named *benefit-mapped.csv*. This file contains a row of column headings only and contains no data. The column headings are not marked as output, and do not match the public names of the inferred entity attributes:

	A	B	C	D	E
1	#	g-key	b-name	b-total	
2					
3					
4					
5					
6					

The XML configuration file contains a `<mapping>` element for the benefit entity:

```

...
  <mapping entity="benefit" table="benefit-mapped" primary-key="#" >
    <attribute name="benefit_name" field="b-name" output="true" />
    <attribute name="benefit_total_eligible" field="b-total" output="true" />
    <relationship name="all-benefits" source-entity="global" foreign-key="g-key"
  />
</mapping>
...

```

The explicit mapping instructs the Batch Processor to map the inferred benefit entity to the *benefit-mapped.csv* file with the identified output columns for the entity attributes.

Implicit Configuration

The CSV input directory contains a CSV file named *benefit.csv*, matching the public name of the benefit entity from the rulebase; no explicit mapping is provided. The file contains a row of column headings only and no data. The column headings are marked as output and match the public names of the inferred entity attributes:

	A	B	C	D	E
1	(#)	(global)	(benefit_name)	(benefit_total_eligible)	
2					
3					
4					
5					
6					
7					

In the absence of an explicit mapping provided by a configuration file, the Batch Processor will match the *benefit.csv_file* name to the public name of the **_benefit** entity. The identified output columns will be mapped to the entity attributes with matching public names.

Configure the Batch Processor

The Batch Processor is invoked in the following ways.

```
java -jar determinations-batch.jar <command line parameters>
```

```
Determinations.Batch.exe <command line parameters>
```

Command line configuration

The following is a list and description of each of the Batch Processor's command line parameters:

--rulebase <rulebase path>

Specifies the rulebase to be used for the batch processor.

--csv <folder>

Specifies the folder in which the csv data files are located. This parameter must be provided if the **--database** parameter is not used.

--delimiter <character>

Identifies the value delimiter to be used when reading and writing CSV files. Defaults to a single comma (,) character. This parameter will be ignored if the batch processor is not reading from or writing to CSV files.

As white space characters cannot be passed easily as command line parameters, special values of **\t** (tab) and **\s** (space) can be used to specify a tab or space character as the delimiter; for example: **--delimiter \t**.

--coverage <coverage file>

Outputs a coverage file that can be imported into Oracle Policy Modeling's *Analyze Coverage File* feature.

--database <db-connection-string>

Specifies the connection string of the database to be used as the source of input data. This parameter must be provided if the **--csv** parameter is not used; for example: **jdbc:oracle:thin:user/password@localhost:1521/example**.

--dboutput

Writes the results of the batch run back to the database. This parameter can only be used if the output came from the database (**--database** option).

--userid <db-userid>

Specifies the user id for a database connection. This parameter can only be used when the userid is not provided in the connection string (**--database** option).

--password <db-password>

Specifies the password for a database connection. This parameter can only be used when the password is not provided in the connection string (**--database** option).

--dbprovider <db-connection-string>

Specifies the provider Invariant name for a .NET database connection.

--driver <driver-name>

Specifies the name of the database driver to be used to connect to the database specified by the **--database** parameter; for example, `oracle.jdbc.OracleDriver`. This parameter will be ignored if the **--database** parameter is not included.

--driversrc <path>

Specifies the full path of the external resource containing the database driver identified by the **--driver** parameter; for example, jar file name. This parameter will be ignored if the **--database** parameter is not included.

--base <name>

Specifies the 'base' table that represents the cases. Multiple csv files represent a database, but one must be identified as the one corresponding to cases. This parameter is optional if there is only a single csv file, or there is a csv file named 'global'; otherwise it is mandatory.

--processors <number>

Specifies the number of processors to use for the batch processor; default value is the number of processors available.

--blocksize <number>

Specifies the number of cases included in each data block read or updated.

--output <folder>

Specifies the path of the file to write any input or output attributes in csv format. If included, the path for the output file must not be the same as the data folder specified by the **--csv** parameter.

--limit <number>

For database input only, this sets a limit to the number of cases processed by the batch processor. This can be useful if you are operating on a large data set, but don't necessarily want to process all the cases; for example you may be verifying that the configuration is correct.

--export <folder>

Exports cases as saved sessions into the specified folder. The **--limit** parameter is handy if you wish to limit the number of cases to be exported.

--exporttsc <filename>

Exports cases into a single .tsc test case file, suitable for adding to an Oracle Project Modeling project. The test file will

have the extension '.tsc' appended if it is missing; for example, `--exporttsc c:\temp\my_test.tsc`.

--config

Specifies the xml configuration file that is used for mapping (non-zero configuration).

XML file configuration

As well as specifying options on the command line (see [Command line configuration](#) above) you can also specify options in a Batch Processor configuration file.

When the Batch Processor starts, it looks for a file in the current working directory called *config.xml* and if this file is found, it will read in the configuration from this file.

Set options in the XML configuration file

All options that can be set on the command line can also be set in the `<options>` section of the configuration file; note that if an option is found on the command line **and** in the config file, then the command line overrides the configuration file setting.

The following options can be set:

Element Name	Description	Example
base	Name of the 'base' table that represents the cases; equivalent to <code>--base</code> on the command line.	<code><base>tablename</base></code>
processors	The number of slave processors to start; equivalent to <code>--processors</code> on the command line.	<code><processors>2</processors></code>
limit	Limits the number of rows to process; equivalent to <code>--limit</code> on the command line.	<code><limit>1000</limit></code>
rulebase	The rulebase to use; equivalent to <code>--rulebase</code> on the command line.	<code><rulebase>SocialServicesScreening.zip</rulebase></code>
csv	The csv directory to get input from; equivalent to <code>--csv</code> on the command line.	<code><csv>./data/csv</csv></code>
delimiter	The value delimiter to be used when reading from or writing to CSV files. Equivalent to <code>--delimiter</code> on the command line. Special values of <code>\t</code> (tab) and <code>\s</code> (space) can be used to specify a tab or space character as the delimiter respectively.	<code><delimiter>\t</delimiter></code>
blocksize	Specifies the number of cases included in each data block read or updated.	<code><blocksize>800</blocksize></code>
database	The definition for a database source; this element has sub-elements which are equivalent to	<code><database> <url>http://localhost/db:8001</url></code>

Element Name	Description	Example
	the --database , --driver , --driversrc , --userid , --password and --dbprovider options on the command line.	<pre><driver> </driver> <driversrc> </driversrc> <userid> </userid> <password> </password> </database></pre>
output	<p>The output location. The "type" attribute indicates the type of output (defaults to "csv"). Equivalent to the --export, --exporttsc, --db and --coverage options on the command line.</p> <ul style="list-style-type: none"> • If the type is "db" then output is written back to the database. No value is expected here. • If the type is "csv" the value is a directory where the csv files with outcomes will be written. • If the type is "coverage", "export" or "exportsc" the value represents the file the exported test case, session or coverage file. 	<pre><ouput type="csv"> ./data/out/csv </output></pre>

Data mapping in the XML configuration file

Mappings are used to map csv and database structures to Oracle Policy Automation data structures: boolean format, entities, relationships and attributes.

If the input data is csv files, much of the mapping from csv data to Oracle Policy Automation data may be done automatically (see [Zero-configuration conventions for CSV input](#)). Specifying data mappings can be used to enhance or change the default mappings of csv data.

If the input data is database tables, mapping information must be specified, as there are no zero configuration conventions for database input.

Specify the global boolean format

The global boolean format defines the format for boolean values read from and written to a csv or database data source.

This element must include the following attributes:

1. The xml attribute **true-value** defines the value for true when reading from, or writing to, the data source
2. The xml attribute **false-value** defines the value for false when reading from, or writing to, the data source.

Example global boolean mapping

```
<mappings>
  <boolean-format true-value="" false-value="" />
  <!-- entity mapping -->
  ...
```

```
        <!-- entity attributes and relationships -->
    </mapping>
</mappings>
```

Specify an entity mapping

Specifying an entity mapping is done as follows:

1. The xml attribute **entity** is used to specify the entity on the rulebase. In this case, it refers to an entity called *customer* in the rulebase.
2. The xml attribute **table** is used to define the source table. This states the source is either from a csv file called *customer.csv* or a database table called *customer*.
3. The optional xml attribute **output-table** can be used when writing to a database, to identify an alternate table into which the output attributes of the entity are to be inserted.

The output table must have a *primary key* field matching the configured **primary-key** attribute, and attributes matching all output attributes identified in the entity mapping.

The output table should be empty before the batch processor is run to prevent primary key collisions. This attribute will be ignored when writing to CSV output.

4. The xml attribute **primary-key** is used to define the primary key for the source. For a database source, this is where you specify the primary key of your table. Note that by default, the primary key for a csv source is the '#' column in the csv file.
5. The optional xml attribute **primary-key-type** can be used to specify whether the source table has a text or integer primary key. If the attribute is not provided, the batch processor will assume an integer key.
6. The optional xml attribute **primary-key-auto** can be used to specify whether primary key values for the underlying table are automatically generated by the data source. If the attribute has been set to "true" the Batch Processor will not include a primary key value when attempting to insert a row into the table. If the attribute has not been provided, the Batch Processor will use the default value of "false", and will attempt to generate the primary key values itself.

This attribute will be ignored when writing to CSV output, or if this is not an inferred entity; see [Batch Processor output](#) for more information.

Example Entity mapping

```
<mappings>
  <mapping entity="customer" table="customer" output-table="customer_out"
    primary-key="customer_id" primary-key-type="text" primary-key-auto="false">
    ...
    <!-- entity attributes and relationships -->
  </mapping>
</mappings>
```

Specify an attribute mapping

Attribute mappings are contained within an entity mapping. Each attribute element specifies the mapping for the rulebase attribute.

1. 'name' is used to define the name of the attribute on the rulebase.
2. 'field' is used to define the source field (for example, column in the csv file).
3. The optional 'output' is used to identify the field as an output field. If not included the value will default to "false" and the field will not be used as output.
4. If we are writing csv output, we can use the optional 'csv-output-field' to change the column name on the output.

IMPORTANT:

If a field was specified as an output field in the csv via parentheses '(' and ')', but is specified again in the configuration XML file without the output="true" flag, it will not be an output field. The information specified in the configuration file supersedes the CSV information if present in both places.

Example attribute mappings

```
<entity entity="customer" table="customer" primary-key="#">
  <attribute name="income" field="income" />
  <attribute name="result" field="result" output="true" />
  <attribute name="result" field="result" csv-output-field="newcolumnname" />
</entity>
```

Specify a relationship mapping

All relationships must have the two xml attributes:

1. 'name' - must match the relationship name of the source entity.
2. 'source-entity' - is the name for the source entity of the relationship.

Specific for one-to-one, one-to-many/many-to-one relationships.

'foreign-key' - is the column field name which is used as the foreign-key for the relationship. The foreign-key has to be specified on the many side of the one-to-many relationship.

Specific for many-to-many relationships

1. 'rel-source' - is the source of the many-to-many mapping. In the example below, it states that the source of the many-to-many mapping is coming from the csv file called plansproducts.
2. 'source-key' - is the foreign key reference to the primary key of source table.
3. 'target-key' - is the foreign key reference to the primary key of target table.

Note: Many-to-many relationships can be specified at either side.

Example relationship mappings

```
<mapping entity="customer" table="customer" primary-key="#">
  <relationship name="applicanttopincomeearner" source-entity="global" foreign-key-
y="applicanttopincomeearner" />
  <relationship name="customersfavoriteproductrev" source-entity="product" foreign-key-
y="customersfavoriteproduct" />
</mapping>
```

```
<mapping entity="product" table="product" primary-key="#">
  <!-- many-to-many -->
  <relationship name="plansproducts" source-entity="plan" rel-source="plansproducts" source-key="plan" target-key="product"/>
</mapping>
```

Structure of the XML configuration file

```
<configuration>
  <options>
  ...
  </options>
  <mappings>
  ...
  </mappings>
</configuration>
```

The structure of a Batch Processor configuration is quite simple. The root element of the configuration file is the `<configuration>` element.

Next there can be a single `<options>` element which contains all the Batch Processor configuration options (see [Set options in the XML configuration file](#)).

Next there can be a single `<mappings>` element which contains all of the Batch Processor data mappings.

Use log messages in the Batch Processor

The Batch Processor provides feedback through log messages. While basic progress details are logged to the console, the Batch Processor can be configured using **log4j** (or **log4net** for the .NET version), to provide more verbose details, or to log details to different destinations.

For more information, go to:

<http://logging.apache.org/log4j/1.2/index.html>

<http://logging.apache.org/log4net/>

Note: If you wish to write Batch Processor logs to a database, you will need to consult the documentation for the chosen *appender* regarding what you will need to do to escape special characters like single quotes (') and commas (,).

Basic Progress

Basic progress information is always logged to the console. This information provides progress indicators during processing, and a brief summary that includes the number of cases processed, number of cases ignored and total time taken, once processing has been completed.

Java - log4j configuration

There is a default *log4j.xml* file inside the *determinations-batch.jar* file. The logging level is set to "warn" and send output to the console. If a *log4j.xml* file is placed in the working directory of the Batch Processor, that file will be used instead of the default configuration.

.NET - log4net configuration

The .NET runtime comes with a log4net configuration file (*log4net.xml*). The debugging level is set to "warn" and send output to the console.

Note: You should change this file (or replace it with one of your own). You should also ensure that the *log4net.xml* is in the working directory when executing the batch process.

Named Loggers

To allow for specific categories of information to be easily separated from the rest of the logged information, the Batch Processor uses the following log categories:

- **MAIN** - Log category for the main progress information
- **CASE_ERROR** - Log category for reporting errors that affect individual cases.
- **CASE_WARNING** - Log category for reporting warnings that do not prevent an individual case from being processed, but may have adversely affected the case outcomes.

Log messages for these categories are filtered and logged according to the default log configuration described above. If a custom log configuration file is provided, the logging level and destination can be changed. The example below is of a custom **log4j** configuration for Java.

Case ID Field for Case Errors and Case Warnings

Messages logged to the **CASE_ERROR** and **CASE_WARNING** categories described above are attributed to individual cases. The identity of the case is included in the message itself, but to allow for simpler parsing the case identity can also be accessed from a custom log field named **CaseID**. This custom log field can be included in the layout of a configured appender with the syntax **%X{CaseID}**.

Example

The following is an example of a custom log4j configuration for Java; you will notice that the logging in this example has been configured to:

1. Set the logging threshold of the **MAIN** category to "info" and log messages to a file named "case-main.log"
2. Set the logging threshold of the **CASE_ERROR** category to "error" and log messages to a file named "case-error.log"
3. Set the logging threshold of the **CASE_WARN** category to "warn" and log messages to a file named "case-warn.log"

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration>
  <appender name="console" class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%p %t %c - %m%n" />
    </layout>
  </appender>

  <appender name="case-main-log" class="org.apache.log4j.FileAppender">
    <param name="File" value="case-main.log" />
    <param name="Threshold" value="info" />
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%t - %m%n" />
    </layout>
  </appender>

  <appender name="case-error-log" class="org.apache.log4j.FileAppender">
    <param name="File" value="case-error.log" />
    <param name="Threshold" value="error" />
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%t - %m%n" />
    </layout>
  </appender>

  <appender name="case-warn-log" class="org.apache.log4j.FileAppender">
    <param name="File" value="case-warn.log" />
    <param name="Threshold" value="warn" />
    <layout class="org.apache.log4j.PatternLayout">
      <param name="ConversionPattern" value="%t - %m%n" />
    </layout>
  </appender>
</log4j:configuration>
```

```
<category name="MAIN" additivity="false">
  <priority value="info" />
  <appender-ref ref="case-main-log" />
</category>

<category name="CASE_ERROR" additivity="false">
  <priority value="error" />
  <appender-ref ref="case-error-log" />
</category>

<category name="CASE_WARNING" additivity="false">
  <priority value="warn" />
  <appender-ref ref="case-warn-log" />
</category>

<root>
  <priority value="warn" />
  <appender-ref ref="console" />
</root>
</log4j:configuration>
```

Debugging the Batch Processor

In order to find out specific details about what the batch processor is doing, such as identifying the SQL queries being performed or verifying that the configuration is being applied as expected, the log configuration provides the ability to change the amount of information provided by the batch processor, and where the information is written.

In the topic [Use log messages in the Batch Processor](#), the sections [Java – log4j – Configuration](#) and [.NET – log4net – Configuration](#) both describe the default logging configuration, and where to place an alternate configuration file.

The [Example](#) section provides an alternative log configuration that includes examples on how to configure a file appender and set the logging level for a specific appender, or for the application as a whole.

For a more specific example, a new file appender could be declared to write debug level information to a file called *debug.log* as follows:

```
<appender name="debug-log" class="org.apache.log4j.FileAppender">
  <param name="File" value="debug.log" />
  <param name="Threshold" value="debug" />
  <layout class="org.apache.log4j.PatternLayout">
    <param name="ConversionPattern" value="%t - %m%n" />
  </layout>
</appender>
```

To configure the application to generate DEBUG level information and to write it to the file appender defined above, the root logger would be configured as follows:

```
<root>
  <priority value="DEBUG" />
  <appender-ref ref="debug-log" />
</root>
```

For more detailed information, it is recommended that you visit the following web sites:

<http://logging.apache.org/log4j/1.2/index.html>

<http://logging.apache.org/log4net/>

System properties and the Batch Processor

Note: This information is for the Java version of the Batch Processor only.

Because the Batch Processor spawns *slave* processes while running, it is important that the system properties can be effectively passed from the main *master* process to the sub-process.

This topic describes which system properties are automatically passed to a slave sub-process and the techniques for passing other system properties to be used only in the sub-process.

Passing system properties to worker sub-processes

There is a single simple pattern for passing a system property to a slave process; any system property specified with a prefix of 'dbslave.' will be passed to each slave sub process.

User properties that are normally set by '-D<property name>=<property value>', can be set by passing **--bplslave <property name>=<property value>**.

Example: Passing a specific timezone configuration to a sub-process

```
java -Duser.timezone=UTC -jar determinations-batch.jar --bplslave -Duser.timezone=UTC ...
```

In this case, the initial Java VM is using the system property **user.timezone=UTC**, the **--bplslave** argument will pass the system property "-Duser.timezone=UTC" to all worker sub-processes.

Passing VM arguments to worker sub-processes

X properties (memory etc) can be passed to worker processes using the same method.

Example: Passing -Xms and -Xmx memory properties to each slave

```
java -jar determinations-batch.jar --bplslave -Xmx128m --bplslave -Xms128m ...
```

In this case the arguments **-Xms256m -Xmx256m** are passed to the worker sub-processes when it starts up.

Batch failure and recovery

The Batch Processor uses parallel processors to progress through cases quickly and efficiently. However, when using multiple parallel processors, it is possible for an error that prevents some, but not all, of the processors from continuing. It is important in this scenario that the Batch Processor handles the error gracefully and that it can be restarted to complete the processing for any cases that were missed.

Handling errors

There are two types of errors that can occur when the batch processor is running; Fatal and Non-Fatal. The Batch Processor manages occurrences of these errors separately.

Fatal errors

Fatal errors are unexpected errors that apply to a processor as a whole; for example, if the database the processor is reading from or writing to becomes unexpectedly unavailable. Fatal errors may affect one or more of the parallel processors, but do not necessarily affect all parallel processors.

When a fatal error is encountered, the error details are logged according to the log configuration, and the affected processor stopped. If using multiple parallel processors, any unaffected processors will continue working to ensure that as many cases as possible are processed. The final summary message provided by the Batch Processor will indicate that an error occurred during processing, and identify the total number of cases that were successfully processed.

Non-Fatal errors

Non-Fatal errors are predictable errors that apply to a single case only; for example, data validation errors (such as trying to read the value 'abc' into a numeric attribute) and errors returned by a specific rule in a rulebase.

When a non-fatal error is encountered, the error details are logged according to the log configuration, the affected case ignored, and the processor continues on to the next case. The final summary message provided by the Batch Processor will identify the total number of cases that were successfully processed and the total number of cases that were ignored due to non-fatal errors.

Recovery after failure

If the Batch Processor has encountered a fatal error, it is likely there will be cases that were not processed. Once the cause of the fatal error has been identified and resolved, the Batch Processor can be run again to reprocess all cases, including those missed previously due to the fatal error.

Best practice - identifying processed and unprocessed cases

To assist in recovery from a failure, a means of easily identifying processed and unprocessed cases should be implemented within the data sources and rulebases to be used. The recommended approach is to use a top-level attribute to specifically record if a case has been processed or not.

When the output of the Batch Processor is database out, you should always treat the database tables that will be updated as the final point of truth as to which cases were processed regardless of log messages that the Batch Processor has produced.

- In the rulebase, include a top-level attribute against the Global entity of the rulebase that can identify if a case has been processed.

- Define the attribute with clear values for differentiating processed and unprocessed cases. Some examples are:
 - A boolean attribute that will be set to TRUE only when a case is successfully processed.
 - A date-time, date or time value that will record the last time a case was successfully processed
- Define a rule within the rulebase that will set the value of the attribute appropriately, regardless of any other outcomes determined for the case.
- In the data source, include an output column that is mapped to the top-level attribute.
 - The initial value of the column for each case must indicate that the case has not been processed:
 - For a boolean attribute, a value representing FALSE or NULL
 - For a date-time, date or time attribute, a value representing NULL or a time in the past.

Using a top-level attribute, the data source can be examined after a fatal error to easily identify the cases that have been successfully processed and the cases that have not.

Best practice - presenting unprocessed cases with database views

When connecting to a database, it is possible that the batch processor is responsible for reading and updating data for a very large number of cases. To make recovery from failure more efficient, the data source can be designed to ensure only unprocessed cases are presented to the Batch Processor. The recommended approach is to combine a method of clearly identifying cases that need to be processed (such as the recommended approach described above) with database views.

- In the database, implement database views that only select data for cases that need to be processed.
 - Using the boolean or date-time value options described earlier, the view definition should only select data for cases that have not been processed.
- Configure the batch processor to read from and write to the database views.

Using this approach, the Batch Processor will only read and update data for cases that need to be processed. When restarting the Batch Processor after a fatal error, it will only process those cases that were not processed in the previous run.

How do I identify my Batch Processor input as database tables?

You can identify your Batch Processor input as database tables through command line options or in the configuration file.

In the command line options, use the **--database <db-connection-string>**, where db connection string is the JDBC URL or ADO.NET connection string for the database.

You may also have to specify additional database connection parameters; see [Configure the Batch Processor](#) for more information.

Once you have identified the database connection to use, you will have to define a mapping in the Batch Processor configuration to indicate which tables and columns to use; see [Data mapping in the XML configuration file](#) for more information on mapping database tables and columns to Oracle Policy Automation data.

How do I identify my Batch Processor input as csv files?

If no input is specified, the Batch Processor will assume the input in csv and look for a directory named "csv". You can explicitly specify csv data and the directory to get that data from by using the `--csv <directory>` command line option, or using the `<csv>-directory</cv>` in the configuration file.

For more information, see [Configure the Batch Processor](#).

An example of the command line follows:

```
java -jar determinations-batch.jar --rulebase SimpleBenefits.zip --csv "./data/csv_in" --output "./data/csv_out"
```

How do I map Batch Processor data to a global boolean format?

Normally, boolean values are written as the "true" or "false". However, if you have a specific format that you would prefer these booleans to be written in, for example, "Y" for true or "N" for false, you can do so by configuring it with the alternative boolean formatter; the configured value will then apply a value of *True* or *False* to all boolean attributes within the configuration file when reading from, or writing to, a data source.

- When writing output values, the configured values will be applied strictly as follows:
 - The configured true-value will be used for all TRUE output values
 - The configured false-value will be used for all FALSE output values
- When reading input values, the configured values will augment the default parsing as follows:
 - The input value is considered TRUE if it matches the configured true-value
 - The input value is considered FALSE if it matches the configured false-value
 - The default parsing rules will be applied if the input value does not match either of the configured values.

For more information, see [Data mapping in the XML configuration file](#).

How do I map Batch Processor input data to a rulebase entity?

When using csv data, by default, the name of a csv file will be mapped to an entity with the same name if one exists; for example, if there is a csv data file called *child.csv* and the rulebase has an entity with the public name *child* then rows in the *child.csv* will be treated as child entity instance. You can explicitly map an csv file or database table to a rulebase entity by specifying a mapping.

Use the xml element `<entity>` in the configuration file, to specify a mapping with the following attributes:

- Entity attribute defines the rulebase entity public name.
- Table attribute defines the csv or database table.
- Primary-key attribute defines the unique primary key for each entity. This is needed if any relationships will be mapped for this entity.

There are also two optional attributes which are as follows:

- Output-table attribute is used to identify an alternate table into which the output attributes of the entity are to be inserted, when writing to a database.
- Primary-key-type attribute specifies whether the source table has a text or integer primary key. If not provided, an integer key is assumed.

For more information, see [Data mapping in the XML configuration file](#). Also see [Zero-configuration conventions for csv input](#) for more information on default mapping of csv data.

How do I map Batch Processor input data to a rulebase relationship?

When using csv data, if a column in a table exists which has the public name of relationship for that entity, then that column will be mapped as a relationship. If a csv file matches the public name of a many-to-many relationship then, the data in that table will be treated as a many to many relationship.

See the description of column headings used for relationships in [Zero-configuration conventions for csv input](#) for a detailed explanation of how one-to-many relationships are identified in csv input.

You can use data mapping to explicitly identify a column or table to be used as a relationship.

For a one-to-many relationship, the relationship mapping should be placed in the entity which has the foreign key, this is usually the entity on the "to-many" side of the relationship.

For a many-to-many relationship, the relationship mapping can be placed in either (or both) entities.

Use the xml element <relationship> to specify a mapping for a one-to-many relationship, with the following attributes:

- name attribute - defines relationship public name.
- source-entity attribute - defines the source entity for the relationship.
- foreign-key attribute - defines the column in which the foreign-key for the source entity is stored. This field must be a integer field which references a primary key in the source entity table (or csv file).

A many-to-many relationship is specified in a separate table and should contain rows which are source and target keys for the many-to-many relationship. This type of relationship is also specified by using the <relationship> element with the following attributes:

- name attribute - defines relationship public name.
- source-entity attribute - defines the source entity for the relationship.
- rel-source attribute - specifies the table where the relationship rows are specified.
- source-key attribute - specified the column with the foreign key reference to the source entity.
- target-key attribute - specifies the column with the foreign key reference to the target entity.

For more information, see [Data mapping in the XML configuration file](#). See also [Zero-configuration conventions for csv input](#) for more information on the default mapping of csv data.

How do I map Batch Processor input data to a rulebase attribute?

When using csv data, if a column in a table exists which has the public name of attribute for the entity, then that column will be mapped as an attribute.

You can explicitly map a column from a csv file or database table to a rulebase attribute by specifying a mapping.

Use the xml element `<attribute>` in the configuration file, to specify a mapping with the following attributes:

- name attribute defines the rulebase attribute public name
- field attribute defines the csv or database column in which the data exists.
- output attribute (optional) is used to identify the field as an output field. If not included the value will default to "false" and the field will not be used as output.

For more information, see [Data mapping in the XML configuration file](#). See also [Zero-configuration conventions for csv input](#) for more information on default mapping of csv data.

How do I migrate my Data Source Connector project to a Batch Processor project?

The Batch Processor replaces the Data Source Connector, providing the same (and more) functionality. If you have an existing project or application using the data source connector you can migrate from the Data Source Connector to the Batch Processor by following the steps outlined below.

Upgrade the rulebase

The Batch Processor only supports rulebases compiled in the current version of Oracle Policy Modeling. You will have to update the rulebase you are currently using with the Data Source Connector by opening it in Oracle Policy Modeling.

1. Open the rulebase Project in Oracle Policy Modeling
2. Follow the upgrade project steps
3. Build the rulebase

In most cases, upgrading a rulebase is a straightforward procedure. For more information see the topic *Upgrade a project* in the *Oracle Policy Modeling User's Guide*.

Convert Data Source Connector file descriptors to Batch Processor configuration

The Data Source Connector requires a descriptor for each csv file to be processed, the file descriptor contains information about the columns and the data contained in each column. The Batch Processor does not require file descriptors.

You can migrate the information from a Data Source Connector file descriptor to Batch Processor configuration in the following way.

Header information must exist in the original csv file. The first row of any csv file should contain the column names.

Example Data Source Connector file descriptor for *parent.csv*

parent.xml

```
<table name="parent" xmlns="http://oracle.com/determinations/connector/ data-source/table">
  <columns>
    <column header="id" ordinality="1"/>
    <column header="name" ordinality="2"/>
    <column header="wage" ordinality="3"/>
  </columns>
</table>
```

Example data in *parent.csv*:

parent.csv

```
1,John Robinson,350
2,Lois Griffin,400
```

For the Batch Processor, the parent.xml file descriptor is no longer required, but the column headers should be added to parent.csv

Example data in parent.csv for Batch Processor:

```
parent.csv
id,name,wage
1,John Robinson,350
2,Lois Griffin,400
```

If the column headings match the public names of attributes in the parent entity, then no further configuration is needed. Otherwise, attributes can be explicitly configured in the Batch Processor configuration file.

If you did need to explicitly map the parent entity you would add the mapping to the Batch Processor configuration file.

Example mapping for *person.csv* (mapped to rulebase global entity)

```
<mappings>
  <mapping entity="global" table="person" primary-key="id">
    <attribute name="name" field="name" />
    <attribute name="wage" field="wage" />
  </mapping>
  ...
</mappings>
```

For more information on Batch Processor mappings, see [Example Entity mapping, Specify an attribute mapping](#).

Convert Data Source Connector configuration to Batch Processor configuration

The Batch Processor relies on configuration very similar to the Data Source Connector. Configuration for the Batch Processor can be done in a configuration file or passed in as command line arguments. If you are converting a Data Source Connector project, it will be easier to place your configuration in an xml file as this is very similar to the Data Source Connector configuration.

DSC Configuration	Description	BP Con-figuration	Description
<threads>	The number of threads to process the data; example : <threads>3</threads>	In <options> element: <processors>	The number of processes to process the data. Similar to threads, but these are distinct java or .NET processes; example: <processors>3</processors>
<run-limit>	The number of records to process; example: <limit>1000</limit>	In <options> element: <limit>	For csv file the Batch Processor will always process all records. For database only input a limit can be set; example: <limit>1000</limit>
<time-out>	The maximum time the batch processor will run	No corresponding setting.	

DSC Configuration	Description	BP Con-figuration	Description
	for before quitting.		
<data-sources>	Defines all the csv files to be used and the rulebase to process	In <options> element: <csv> and <rulebase>	All csv files in directory specified in the csv element will be processed; example input: <pre><csv>data</csv></pre> example rulebase: <pre><rulebase>./rulebase/SimpleBenefits.zip</rulebase></pre>
data-mappings	Defines attributes, entities and relationships	<mappings>	See Data mapping in the XML configuration file
output	Defines the output	In <options> element: <output>	The Batch Processor has many more available output options, the default output is csv, which can be explicitly defined. Data is always overwritten; example: <pre><output type="csv">../output</output></pre>

Ensure that output attributes are defined in the Batch Processor configuration

As part of converting the data mappings in step 3 you should have the output attributes from the Data Source Connector as output in the Batch Processor.

Because outputs are the most important aspect of a batch process, it is worth checking that the outputs defined in the Batch Processor are correct. In Batch Processor, output attributes can be defined in two different ways.

Defined in the csv file

In this case, the output attribute is defined in an empty column of the csv file, with parentheses around name. The parentheses indicate that this column is an output column.

The name of the column should match the public name of the inferred attribute that will be the output.

Example *person.csv*

```
id,name,wage,(person_is_eligible)
1,John Robinson,350,
2,Lois Griffin,400,
```

In the example above the entity that the person rows correspond to is expected to have an inferred attribute with a public name "person_is_eligible" when the Batch Processor executes, the *person.csv* file will be written to the output with this column filled in with the known results.

Note that the number of commas in the csv file must match the number of columns even if they are empty.

Defined in the configuration mapping

Output attributes can be defined in the Batch Processor mapping in a very similar way to how they are defined in the Data Source Connector.

Define the attribute in the corresponding mapping element for the entity and add the attribute `output="true"`, the field attribute represents the column that the attribute will be written to.

```
<attribute name=" person_is_eligible " field="person_is_eligible" output="true"/>
```

Differences between Data Source Connector output and Batch Processor output

When outputting csv files the Data Source Connector, will write out the resulting attributes to the specified location. However, there are some differences.

The Data Source Connector writes the original csv file and adds the outputs to the out csv whereas the Batch Processor does the following:

Example:

person.csv in input

```
id,name,wage,(person_is_eligible)
1,John Robinson,350,
2,Lois Griffin,400,
```

will look like the following *person.csv* in output

```
id,name,wage,(person_is_eligible)
1,John Robinson,350,true
2,Lois Griffin,400,false
```

In the example above, you can see that all the input fields present in the *person.csv* remain, in their original format. Any output parameters are inserted into the csv file. Unknown and Uncertain results are always outputted as a blank string.

Known issues for the Batch Processor

The following are issues that currently affect the Batch Processor:

Java must be in the path for Java Batch Processor

When the Java Batch Processor starts multiple worker processes, the sub-processes are invoked using "java". In order for execution to be successful, a suitable Java run-time must be available from the operating system path.

Inferred relationships and instances not supported as outputs

Inferred relationships and instances are not supported as outputs in this version of the Batch Processor. While they can be used in a rulebase, any inferred instances will not appear in the output. Also, inferred relationships cannot be specified as outputs; only attributes of existing instances may be specified as outputs.

Batch Processor updates database in fixed sized chunks

The number of cases the Batch Processor can write to the database (if database output is specified) within a single transaction is controlled by the --blocksize configuration option. If these database chunks contain many entity instances, you may need to increase the memory allocation of the batch processor process to avoid *out of memory* errors.

If there is an error updating the database, the whole update chunk will be rolled back.

Language support

Topics in "Language support"

- [Implement support for non-English language](#)
- [Provide language support for messages](#)
- [Localize an existing rulebase](#)

Other reference material:

- [Oracle Policy Automation Rapid Language Support \(RLS\) User Guide](#) (located in the **opm** directory at: [C:\Program Files\Oracle\Policy Modeling\help\opm\OPA RLS User Guide.pdf](#))

Implement support for non-English language

Determinations Engine versus Web Determinations

When discussing localization in Web Determinations, it is important to distinguish between what is the concern of the Interview Engine and that of any clients to that engine such as Web Determinations. Basically, Web Determinations is responsible for localizing everything that is not authored as a part of the rulebase. Further, all Web Determinations and Determinations Engine sessions are created in a specific locale and it is assumed that anything coming out of the Determinations Engine session will be localized according to that locale.

The following are a few examples of what the Determinations Engine is responsible for:

- Attribute text, sentence forms, decision reports etc.
- Controls and screens authored within Oracle Policy Automation.
- The display format of attribute values.
- Warning and Error daemon text.

The following are a few examples of what the Web Determinations is responsible for:

- Accepted attribute input format(s).
- Any text on screens or screens themselves that are not authored inside Oracle Policy Automation; for example, copyright statements, button text, menu text.
- Error and warning messages, excluding the text of error and warning messages that are authored in Oracle Policy Automation; for example, daemons.
- Templates and style sheets used for rendering screens.

File encoding

It should be noted that the .stxt and .exs files must be saved as UTF-8 otherwise invalid characters will result upon deployment. Failure to do this will cause errors when the properties located in non-UTF-8 encoded files are loaded and retrieved.

Locale codes

Generally, the two letter ISO 639-1 code followed by the two letter ISO 3166-1 country code separated by a hyphen is used. The only exception to this is where the two letter ISO 639-1 is not adequate to distinguish the language type, in which case the three character ISO 639-2 codes are used. An example of the latter is where it is necessary to distinguish between Simplified and Traditional Chinese.

Localizations for the following languages are provided:

Language	Locale code
Arabic (Modern)	ar
Chinese (Simplified)	zh-CN
Chinese (Traditional)	zh-HK
Czech	cs

Language	Locale code
Danish	da
Dutch	nl
English (American)	en
English (Great Britain)	en-GB
Finnish	fi
French (France)	fr
German	de
Hebrew	he
Italian	it
Japanese	ja
Korean	ko
Polish	pl
Portuguese (Brazilian)	pt-BR
Portuguese (European)	pt-PT
Russian	ru
Spanish (Modern)	es
Swedish	sv
Thai	th
Turkish	tr

For further information on how Oracle Policy Automation provides language support, refer to the *Oracle Policy Automation Rapid Language Support (RLS) User Guide* that can be found in the **opm** directory at: <C:\Program Files\Oracle\Policy Modeling\help\opm\OPA RLS User Guide.pdf>

Provide language support for messages

Save for a few exceptions, it is the responsibility of Web Determinations to handle the localization of error and warning messages. Only error and warning messages are localized (that is, all objects that implement either `com.oracle.determinations.interview.engine.data.error.Error` or `com.oracle.determinations.interview.engine.data.error.Warning`). Uncaught exceptions and other non-recoverable errors are handled differently.

In Web Determinations, localization is performed through the message service which is responsible for loading the correct message bundle for the specified locale and returning the message for the specific error or warning to be localized. The message bundle itself is a properties file called **messages.<locale>.properties**. The location of this file is specified by the 'messages.path' property in the `application.properties` file.

The message service also utilizes the [Velocity Templating Engine](#) to allow substitution of current property values in the error message. For example, if you wanted to display the attribute ID and the actual value that was entered in an invalid value error, you could do so by setting your error property as:

```
AttributeValueError = The value ${message.value} is not valid for attribute ${message.attributeId}
```

See also:

[messages.<locale>.properties file](#)

[Velocity Templates Developer Guide](#)

Localize an existing rulebase

A rulebase authored in Oracle Policy Modeling can be localized so that when running the rulebase in Oracle Web Determinations, the user has the choice of running the interview in two or more locales. If the rulebase has commentary and has been localized to another locale, the author has the choice of:

- localizing the commentary or not; that is, it is possible to localize a rulebase to a new locale.
- deferring localizing the commentary for the new locale.

Note that to localize a rulebase for small rulebase engagements (presales, proof of concept), it is ideal to have the following:

- some knowledge of xgen files - the files generated from the rulebase after Build process; that is, the .stxt and .exs files in the output directory.
- foundation rule knowledge such as various states of the Boolean and non-Boolean attributes, entities, screens.

For production rulebases, a strong knowledge of the above mentioned areas is essential.

Example scenario (SocialServicesScreening rulebase):

There is a requirement for a rulebase authored in Oracle Policy Modeling with the default English (en-US) locale, to also be available in another language (for example, Danish: da-DK). To enable this:

- The user can perform localization of the English rulebase so that it can support da-DK locale.
- The user **does not** have to create (and keep in sync) a duplicate project in Oracle Policy Modeling as da-DK locale, only the original English Oracle Policy Modeling project needs to be maintained.
- When selecting the rulebase in Oracle Web Determinations, the user is given the option to choose what locale to run the interview as illustrated here:

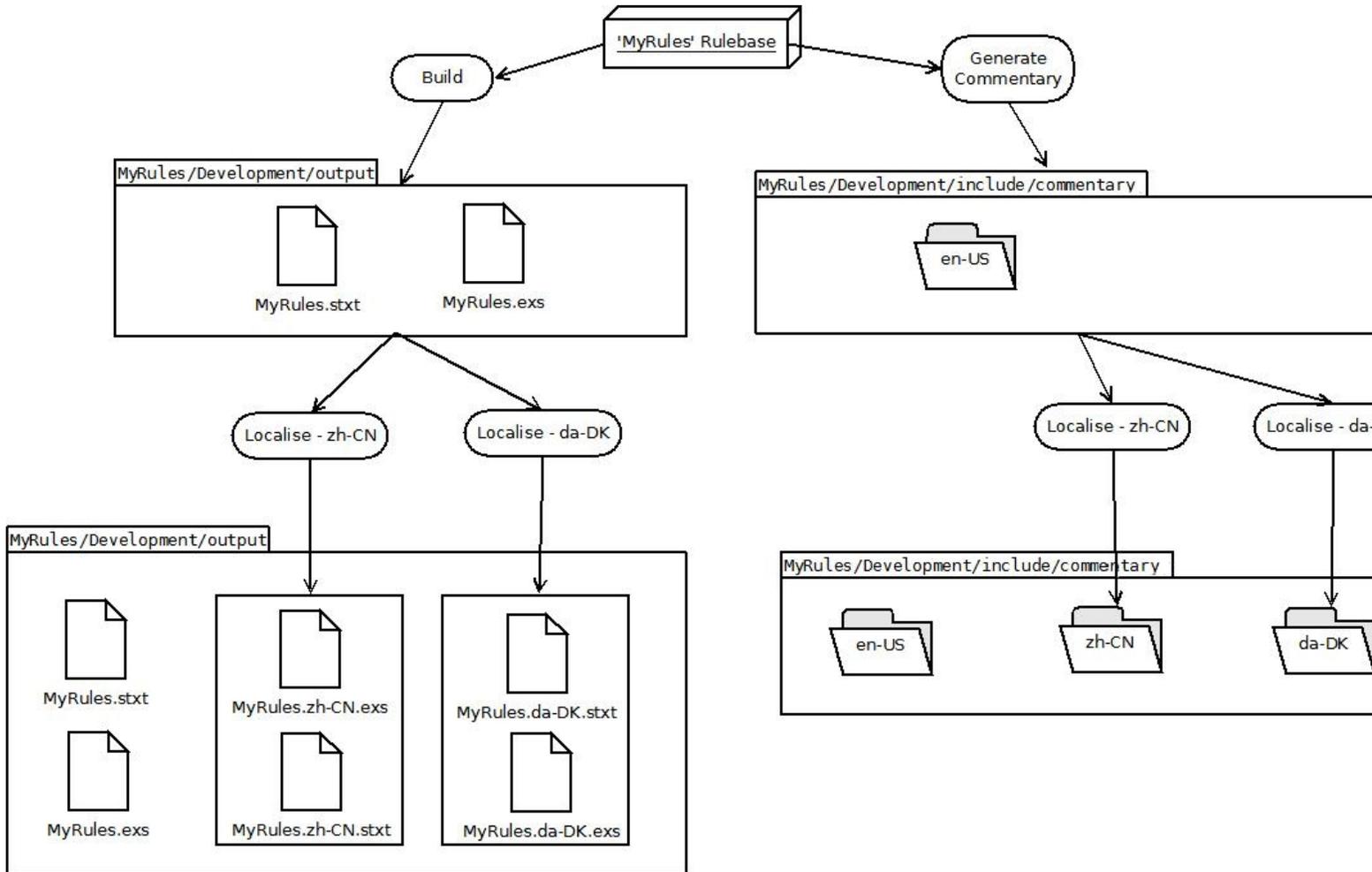
ORACLE Web Determinations

Select locale for SocialServicesScreening:

- [en-US](#)
- [da-DK](#)

Understand the process involved in localizing a rulebase and its commentary

The following diagram illustrates the process involved in localizing a rulebase and its commentary:



Localizing a rulebase

To successfully localize a rulebase, knowledge of XML, OPM xgen files, and Rule authoring is essential.

- Building the rulebase produces an stxt and exs file.
 - The .stxt file contains the entities/relationships/attributes.
 - The .exs file contains screens and screenflows (data from Screen Authoring).
- When localizing a rulebase - the .stxt and .exs file are localized (another copy is created, and displayed text are translated). The stxt and .exs files of the new locale has filenames that reflect the locale to which they belong.

Important note:

Subsequent changes to the rulebase; for example, a new attribute or screen input will require changes to the localized stxt/exs files. That is, they are **not updated** when a rulebase is modified and rebuilt.

It should also be noted that the .stxt and .exs files must be saved as UTF-8 otherwise invalid characters will result upon deployment.

Localizing commentary

- Generating commentary in Oracle Policy Modeling creates the 'commentary' folder inside the 'include' folder.
- Oracle Policy Modeling creates a folder inside the commentary folder, the folder name being the locale of the rulebase project (for this example - en-US).
- To localize commentary, the user needs to duplicate the 'en-US' folder and rename it as the locale. It sits within the same commentary folder.
 - The user then modifies displayed text in the HTML files within the new 'locale', translating en-US display commentary text to the new locale.

Important note:

Subsequent changes to the rulebase or its commentary files/content will require manual changes to the localized commentary files.

Understand how Oracle Web Determinations uses the localized files

As long as the files are placed in the correct folders, Oracle Policy Modeling will package the files in the rulebase zip file (for example, MyRules.zip) during the Build process.

Important note:

It is important for the user to run the 'Build' process after changes are made to either the stxt/exs localized files or the localized commentary. Running the Build process will update the contents of the rulebase zip with the new changes. If the Build process is not run, the changes will not be propagated into the rulebase zip file.

Localized rulebase files

When Oracle Web Determinations finds localized .stxt and .exs files,

- When the user runs the rulebase (MyRules), Oracle Web Determinations presents the user with choice of locale for the Interview (en-US, zh-CN, da-DK).
- Once the user selects a locale (for example, zh-CN), the Interview will be run using the stxt/exs file of the selected locale (MyRules.zh-CN.stxt and MyRules.zh-CN.exs).

Localized commentary files

A rulebase that has Commentary for the default locale does not necessarily need to provide commentary for its other localizations. For example, with the MyRules rulebase the author can choose to only provide Commentary for the default en-US locale.

This means that if the user chooses da-DK or zh-CN locale for an interview, no commentary will be available. Essentially, Oracle Web Determinations is able to detect if there is a corresponding commentary folder for the locale chosen.

Localize a rulebase

The following steps describe how to localize a rulebase. In this example, the rulebase to be localized is a rulebase called "MyRules" and it has a locale of en-US.

1. After authoring the rules of MyRules in Oracle Policy Modeling, run the Build process.
2. Go to the output folder of the MyRules rulebase.
 - i. By default this is located on **C:\projects\MyRules\Development\output**.
3. Inspect to ensure that the .stxt and .exs files (MyRules.stxt and MyRules.exs) have been generated from the Build process.

4. For each locale that the MyRules rulebase needs to be localized, do the following:
 - i. Create copies of the MyRules.stxt and MyRules.exs, and name it in a format MyRules.xx-XX.stxt and MyRules.xx-XX.exs where xx-XX is the new locale; for example, for zh-CN locale the filenames will be MyRules.zh-CN.stxt and MyRules.zh-CN.exs.
 - ii. For the .stxt file, note the following:
 - a. It is simpler to localize than the .exs screen file.
 - b. It is helpful to know the various states of Boolean and non-Boolean attributes as most of the translations to the new locale require translating the text of the various states; for example, Boolean positive/negative/question/uncertain.
 - c. Only translate text in between the XML tags; there is no need to translate the XML tags or its attributes.
 - d. Be careful to ensure the substitution tokens (for example, %app_name?%) are not removed when translating a text. It is very easy to delete the token, and will result in Oracle Web Determinations errors .
 - iii. For the .exs file, note the following:
 - a. **This is much more complicated to localize than the .stxt file.** This is mainly because text to be translated is not solely inside the XML tags. Instead, the text to be translated can be inside XML tags, or values of an attribute of an XML tag; for example, for "<screen type='view' id='s9@Screens' name='summary' title='Assessment Summary' entity='global'>" the title needs to be translated.
5. After creating the localized .stxt and .exs files, run the Build again to automatically package those new files into the MyRules.zip.

Localize commentary

The following steps describe how to localize the commentary for a rulebase. The rulebase to be localized is a rulebase called "MyRules" and it has a locale of en-US.

1. Produce commentary files for MyRules if it is not available or completed:
 - i. To generate commentary files for MyRules, in OPM go to 'Build->Generate Commentary Files'.
 - ii. Once the commentary files have been generated, go to **<MyRulesProjectFolder>/Development/include/commentary**.
 - iii. In the commentary folder, there should now be a folder with the name of the project's locale; for example, en-US.
 - iv. You can add to or modify the generated commentary HTML files.
2. Once commentary files for the MyRules are complete, it is now ready to be localized.
3. Create a folder inside **include/commentary** for each locale you want the commentary localized to; for example, **include/commentary/da-DK** and **include/commentary/zh-CN**.
4. Copy the contents inside the 'en-US' folder, and paste these contents inside the new commentary locales (**include/commentary/da-DK** and **include/commentary/zh-CN**).

5. For each locale, go into each of the HTML file and localize displayed text; note that to be able to determine which text is metadata and which is display text, a knowledge of HTML is desirable.
6. After localizing the commentary files, run the Build again to automatically package those new files into the MyRules.zip.

Project interchange

Topics in "Project interchange"

- [The Project Interchange XML schema](#)
- [The Project Interchange XML schema definition](#)

The Project Interchange XML schema

The Project Interchange XML schema ([OPAInterchange.xsd](#)) gives you the ability to:

- Import data from an XML file and create a new Oracle Policy Modeling project.
- Report on the differences between the project data model and an external data model specified in an XML file.
- Synchronize the project data model with an external data model specified in an XML file.
- Export project data to an XML file.

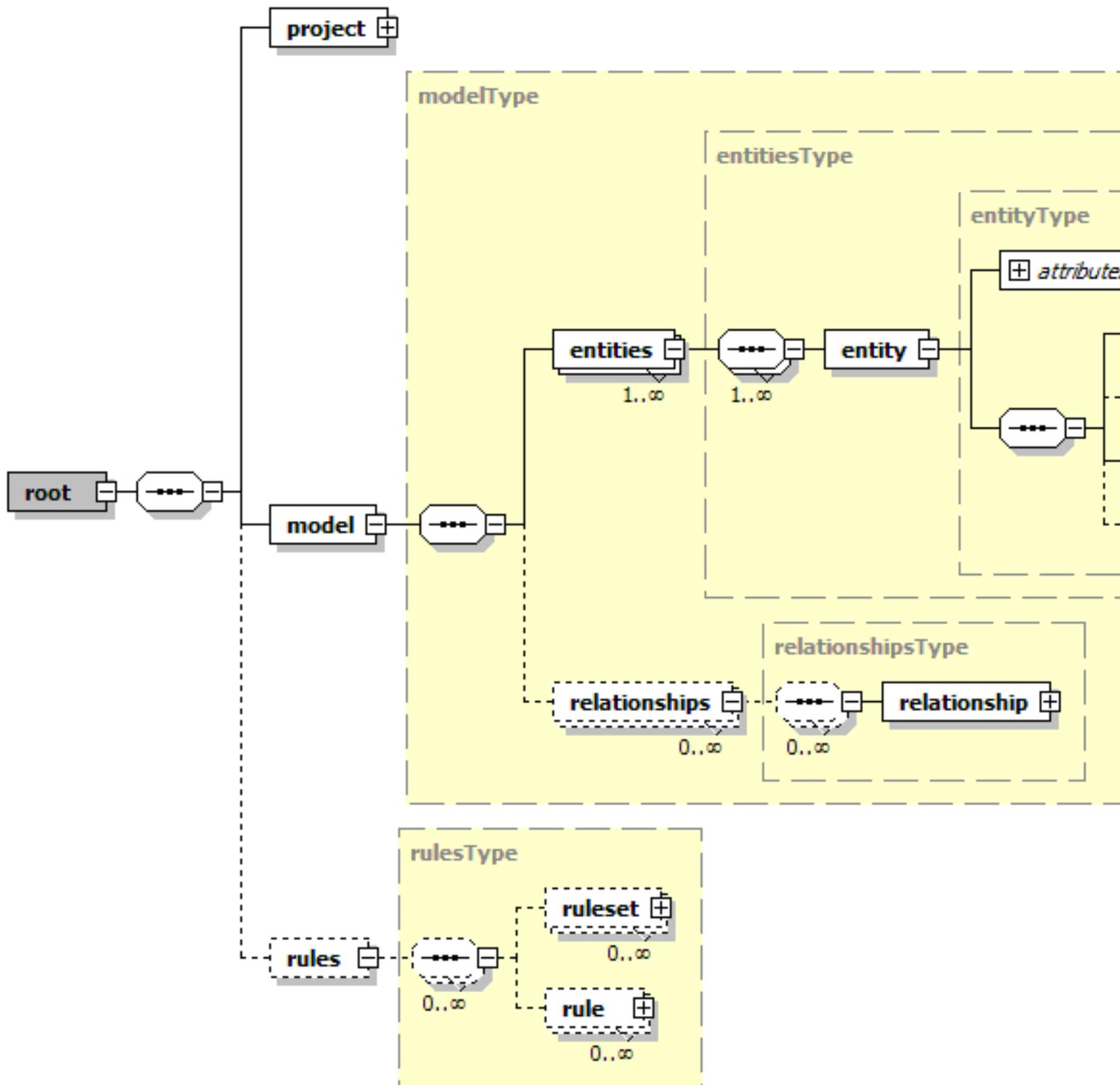
The XML file must comply with the [project interchange XML schema](#) which is available to all customers generally. Other vendors may then produce adapters that write to and read from this XML.

The project interchange XML schema comprises elements that describe the project, the project data model and the project rules. The project includes user attribution data plus templates for the custom properties used elsewhere in interchange files.

The model section allows for specification of the entities, attributes and relationships, plus associated metadata, in the project data model.

The rules section includes rules and rulesets (folders), plus metadata. Rule text in interchange files are represented in three different forms (XHTML, plain text, and engine-ready rule XML) to allow for rendering (or other processing) by external systems such as System Architect. A ruleset in an interchange file is mapped to a project folder when the interchange file is imported into Oracle Policy Modeling.

The schema includes new first class object properties added to support the BRRM business rules management methodology, plus synchronization identifiers for model and rule elements to allow for round-tripping of these data between Oracle Policy Modeling and external metadata management systems such as System Architect and Rochade.



The project interchange XML schema has been designed so that it:

- is sufficiently comprehensive to allow for importing and exporting of all essential Oracle Policy Modeling project data.
- is lightweight, insofar as it specifies only a minimal set of mandatory elements and attributes.
- accommodates the type of data expected to be persisted in repositories and required to be shared with Oracle Policy Modeling, without compromising its generality.
- allows for externally-specified identifiers for rules, data and other declarations, which Oracle Policy Modeling will persist, to support synchronization of these with instances shared with other applications.

Oracle Policy Modeling supports the importing of any project data model created against the Oracle Policy Automation Project Interchange XML schema ([OPAInterchange.xsd](#)) used in Oracle Policy Modeling.

The Project Interchange XML schema definition

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns-
s="http://xmlns.oracle.com/opa/10.4/modeling/interchange" xmlns:NS-
S="http://xmlns.oracle.com/opa/10.4/modeling/interchange"
targetNamespace="http://xmlns.oracle.com/opa/10.4/modeling/interchange" elementFormDefault="qualified" attrib-
uteFormDefault="unqualified" version="10.4:20120202">
  <!--Root element-->
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="project" type="projectType">
          <xs:annotation>
            <xs:documentation>Container for project and other interchange metadata, including the project name, the identity of
the user creating the interchange file, encryption settings, etc.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="model" type="modelType">
          <xs:annotation>
            <xs:documentation>Container for the rulebase data model, including entities, attributes and rela-
tionships.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="rules" type="rulesType" minOccurs="0">
          <xs:annotation>
            <xs:documentation>The collection of rules and rulesets that make up the rulebase.</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <!--Top level elements-->
  <xs:complexType name="projectType">
    <xs:all>
      <xs:element name="name">
        <xs:annotation>
          <xs:documentation>The name of the project.</xs:documentation>
        </xs:annotation>
        <xs:complexType>
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="lang" type="xs:language" use="optional"/>
            </xs:extension>
          </xs:simpleContent>
        </xs:complexType>
      </xs:element>
    </xs:all>
  </xs:complexType>
</xs:schema>
```

```

<xs:element name="date-created" type="xs:date" minOccurs="0">
  <xs:annotation>
    <xs:documentation>The date and time that the instance of the interchange file was created.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="created-by" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>The full name of the person who generated the instance of the interchange file.</xs:
s:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="region" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>The project Region Culture code.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="user-id" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>The user identifier of the person who generated the instance of the interchange file.</xs-
s:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="property-definitions" type="propertyDefinitionsType" minOccurs="0">
  <xs:annotation>
    <xs:documentation>The collection of definitions for custom properties used by assets in the pro-
ject.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="properties" type="propertiesType" minOccurs="0"/>
</xs:all>
</xs:complexType>
<xs:complexType name="modelType">
  <xs:annotation>
    <xs:documentation>The rulebase model, comprising entities, attributes and relationships.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="entities" type="entitiesType" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>The collection of entities in the rulebase model.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="relationships" type="relationshipsType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>The collection of relationships between the entities in the rulebase model.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>

```

```

</xs:complexType>
<xs:complexType name="rulesType">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="ruleset" type="rulesetType" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>A ruleset is a group of rules, and potentially, other rulesets, that are semantically related in some way. The grouping is user-definable and has no effect on the how the rulebase executes - eg. rulesets do not specify any scope for rules.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="default-structural-element" type="defaultStructElementType" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element name="rule" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:complexContent>
          <xs:extension base="ruleType"/>
        </xs:complexContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<!--Base element definitions-->
<xs:complexType name="baseAttributeTextType">
  <xs:sequence>
    <xs:element name="base">
      <xs:annotation>
        <xs:documentation>The text of the attribute.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="lang" type="xs:language" use="optional"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="statement-form" type="questionFormType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>If included and enabled, specifies the text that Oracle Policy Modeling should use for the statement form of the attribute instead of the base text.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="question-form" type="questionFormType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>If included and enabled, specifies the text that Oracle Policy Modeling should use for the question form of the attribute instead of the base text.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```

    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="baseAttributeType">
  <xs:annotation>
    <xs:documentation>Base type for attributes, including attribute aliases.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="definition" type="definitionType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>The definition of the attribute.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="attribute-text" type="attributeTextType">
      <xs:annotation>
        <xs:documentation>The text of the attribute, optionally including overrides for various natural language forms.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>The name of the rulebase attribute. This is mandatory for both import and export. For export, if an attribute is public, this will be the public name. If it is not public, it will be the build model identifier of the attribute (ie. the document identifier + '@' + document scope). Note that for non-public attributes, this name is not persistent, but is regenerated every time the rule document is compiled.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="synchronization-id" type="synchronizationIdType" use="optional">
    <xs:annotation>
      <xs:documentation>May be used by a system or application that supports generation and importing of project interchange files to assign a persistent mapping identifier to a rulebase artifact - eg. to support round-tripping.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="baseRuleType">
  <xs:annotation>
    <xs:documentation>Base type for rules and rulesets.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
    <xs:element name="name">
      <xs:annotation>
        <xs:documentation>The name of the rule or ruleset.</xs:documentation>
      </xs:annotation>
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">

```

```

        <xs:attribute name="lang" type="xs:language" use="optional"/>
    </xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="source" type="xs:string" minOccurs="0">
    <xs:annotation>
        <xs:documentation>The source of the rule, eg. legislative reference, policy document name, instruction number,
etc.</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="source-date-range" minOccurs="0">
    <xs:annotation>
        <xs:documentation>An optional commencement and/or end date for a rule or ruleset. These dates are documentary,
only: they have no effect on whether a rule or ruleset will fire in a compiled rulebase.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="from" type="xs:date" use="optional"/>
                <xs:attribute name="to" type="xs:date" use="optional"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="definition" type="definitionType" minOccurs="0">
    <xs:annotation>
        <xs:documentation>A definition or business expression applying to the rule or ruleset.</xs:documentation>
    </xs:annotation>
</xs:element>
<xs:element name="properties" type="propertiesType" minOccurs="0">
    <xs:annotation>
        <xs:documentation>The collection of named (key/value) properties applying to the rule or rule-
set.</xs:documentation>
    </xs:annotation>
</xs:element>
</xs:sequence>
<xs:attribute name="synchronization-id" type="synchronizationIdType" use="optional">
    <xs:annotation>
        <xs:documentation>May be used by a system or application that supports generation and importing of project interchange
files to assign a persistent mapping identifier to a rulebase artifact - eg. to support round-tripping.</xs:documentation>
    </xs:annotation>
</xs:attribute>
</xs:complexType>
<!--Derived element definitions-->
<xs:complexType name="attributeType">
    <xs:complexContent>

```

```

<xs:extension base="baseAttributeType">
  <xs:sequence>
    <xs:element name="properties" type="propertiesType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>A collection of arbitrary or user-defined key/value pairs associated with the attrib-
ute.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="aliases" type="attributeAliasesType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>A collection of attributes that are bound to this attribute. The bound attribute is an alias for the
primary attribute - it has no distinct or separate value.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="constraint" type="constraintType" minOccurs="0">
      <xs:annotation>
        <xs:documentation>If present, may be used to constrain the values that an attribute may be assigned at
runtime.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="type">
    <xs:annotation>
      <xs:documentation>The data type of the attribute - ie. boolean, text, currency, number or date.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:enumeration value="boolean"/>
        <xs:enumeration value="text"/>
        <xs:enumeration value="number"/>
        <xs:enumeration value="date"/>
        <xs:enumeration value="datetime"/>
        <xs:enumeration value="timeofday"/>
        <xs:enumeration value="currency"/>
      </xs:restriction>
    </xs:simpleType>
  </xs:attribute>
  <xs:attribute name="public" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>Indicates whether entity participates in the rulebase's public data model.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="module-only" type="xs:boolean" use="optional">
    <xs:annotation>
      <xs:documentation>Indicates if the attribute is defined only in a module (such attributes are exported, but not re-
imported).</xs:documentation>
    </xs:annotation>
  </xs:attribute>

```

```

    </xs:attribute>
  </xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="attributeAliasType">
  <xs:complexContent>
    <xs:extension base="baseAttributeType"/>
  </xs:complexContent>
</xs:complexType>
<xs:complexType name="attributeTextType">
  <xs:complexContent>
    <xs:extension base="baseAttributeTextType">
      <xs:sequence minOccurs="0">
        <xs:annotation>
          <xs:documentation>For boolean attributes only, the following elements may be specified.</xs:documentation>
        </xs:annotation>
        <xs:element name="uncertain-form" type="questionFormType" minOccurs="0">
          <xs:annotation>
            <xs:documentation>If included and enabled, specifies the text that Oracle Policy Modeling should use when the
attribute has an uncertain value.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="parse" type="xs:string" minOccurs="0">
          <xs:annotation>
            <xs:documentation>Applies to boolean attributes only, specifies the object, subject and verb elements of the attrib-
ute.</xs:documentation>
          </xs:annotation>
        </xs:element>
        <xs:element name="negative-form" type="questionFormType" minOccurs="0">
          <xs:annotation>
            <xs:documentation>If included and enabled, specifies the text that Oracle Policy Modeling should use when the
attribute is negative.</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="allow-subst" type="xs:boolean" default="false">
        <xs:annotation>
          <xs:documentation>If true, the value of the attribute is substituted for the attribute text when it appears in rulebase
statements. Applies to non-boolean attributes only.</xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="plural" type="xs:boolean" default="false">
        <xs:annotation>
          <xs:documentation>If true, the base attribute text is plural. Applies to non-boolean attributes only.</x-
s:documentation>
        </xs:annotation>
      </xs:attribute>

```

```

<xs:attribute name="gender" default="impersonal">
  <xs:annotation>
    <xs:documentation>Specifies the default gender of the attribute: impersonal (it), generic (he/she), male (he) or
female (she). Applies to non-boolean attributes only.</xs:documentation>
  </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="generic"/>
      <xs:enumeration value="male"/>
      <xs:enumeration value="female"/>
      <xs:enumeration value="impersonal"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="constraintType">
  <xs:attribute name="min" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>The minimum value the attribute may take. Applies to numeric, currency and date attributes only.</x-
s:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="max" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>The maximum value the attribute may take. Applies to numeric, currency and date attributes
only.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="reg-exp" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>A regular expression that will be applied to validate the attribute. Applies to text attributes only.</x-
s:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="validation-message" type="xs:string" use="optional">
    <xs:annotation>
      <xs:documentation>User defined error message to be displayed when validation fails.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="entityType">
  <xs:sequence>
    <xs:element name="entity-text">
      <xs:annotation>
        <xs:documentation>The textual form of the entity name.</xs:documentation>

```

```

</xs:annotation>
<xs:complexType>
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="lang" type="xs:language" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="definition" type="definitionType" minOccurs="0">
  <xs:annotation>
    <xs:documentation>The definition of the entity.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="attributes" type="attributesType">
  <xs:annotation>
    <xs:documentation>The collection of attributes defined for the entity.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="properties" type="propertiesType" minOccurs="0"/>
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required">
  <xs:annotation>
    <xs:documentation>This is the entity identifier and must be unique within the interchange document. This is a generated value based on the entity's text</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="public" type="xs:boolean" use="optional">
  <xs:annotation>
    <xs:documentation>Indicates if the entity is using a public name.</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="module-only" type="xs:boolean" use="optional">
  <xs:annotation>
    <xs:documentation>Indicates if the entity is defined only in a module (such entities and their attributes are exported, but not re-imported).</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="synchronization-id" type="synchronizationIdType" use="optional">
  <xs:annotation>
    <xs:documentation>May be used by a system or application that supports generation and importing of project interchange files to assign a persistent mapping identifier to a rulebase artifact - eg. to support round-tripping.</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="singleton" type="xs:boolean" use="optional" default="false">
  <xs:annotation>
    <xs:documentation>If true, there can be at most one instance of this entity. Otherwise, any number of instances is

```

allowed. In the case of the global entity, this attribute must be true, and indicates there is exactly one instance of this entity. </xs:documentation>

```
</xs:annotation>
</xs:attribute>
<xs:attribute name="containment-relationship-id" type="xs:string" use="optional">
  <xs:annotation>
    <xs:documentation>ID of the entity's containment relationship</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="containment-parent-id" type="xs:string" use="optional">
  <xs:annotation>
    <xs:documentation>ID of the entity's containment parent (i.e. the containing entity)</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="identifying-attribute-name" type="xs:string" use="optional">
  <xs:annotation>
    <xs:documentation>Public ID of the entity's identifying attribute</xs:documentation>
  </xs:annotation>
</xs:attribute>
</xs:complexType>
<xs:complexType name="relationshipType">
  <xs:sequence>
    <xs:element name="source" type="xs:string">
      <xs:annotation>
        <xs:documentation>The name of the parent or LHS entity in the relationship. </xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="target" type="xs:string">
      <xs:annotation>
        <xs:documentation>The name of the target, child or RHS entity in the relationship.</xs:documentation>
      </xs:annotation>
    </xs:element>
    <xs:element name="relationship-text">
      <xs:annotation>
        <xs:documentation>The textual form of the relationship name. This is normally expressed in plural form of the relationship and used by the natural language processor to identify the presence of the relationship in a quantification operation - eg. existential, universal, instance selection, (conditional) instance counting.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attribute name="lang" type="xs:string"/>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
    <xs:element name="reverse-text">
```

```

<xs:annotation>
  <xs:documentation>The textual form of the reverse relationship. </xs:documentation>
</xs:annotation>
<xs:complexType>
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="lang" type="xs:string"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="properties" type="propertiesType" minOccurs="0"/>
</xs:sequence>
<xs:attribute name="name" type="xs:string" use="required">
  <xs:annotation>
    <xs:documentation>The name of the relationship. This must be unique and comply with the object naming rules: ie. begin with a letter, contain only letters, numbers, '_', '@' and not include any spaces.</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="module-only" type="xs:boolean" use="optional">
  <xs:annotation>
    <xs:documentation>Indicates if the relationship is defined only in a module (such relationships are exported, but not re-imported).</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="reverse-name" type="xs:string" use="required">
  <xs:annotation>
    <xs:documentation>The name of the relationship. This must be unique and comply with the object naming rules: ie. begin with a letter, contain only letters, numbers, '_', '@' and not include any spaces.</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="public" type="xs:boolean" use="optional">
  <xs:annotation>
    <xs:documentation>Indicates the relationship name is a public name</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="reverse-public" type="xs:boolean" use="optional">
  <xs:annotation>
    <xs:documentation>Indicates the reverse relationship name is a public name</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="is-containment" type="xs:boolean" use="required">
  <xs:annotation>
    <xs:documentation>Indicates if the relationship is a containment relationship</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="synchronization-id" type="synchronizationIdType" use="optional">

```

```

    <xs:annotation>
      <xs:documentation>May be used by a system or application that supports generation and importing of project interchange
files to assign a persistent mapping identifier to a rulebase artifact - eg. to support round-tripping.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="type" type="xs:string" use="required">
    <xs:annotation>
      <xs:documentation>The relationship type. Single indicates a one-to-one relationship. Child indicates a one-to-many rela-
tionship from source to target. Parent indicates a path from a child entity to its parent. Multiple indicates a many-to-many rela-
tionship.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="defaultStructElementType">
  <xs:annotation>
    <xs:documentation>The file name of the document that the ruleset represents.</xs:documentation>
  </xs:annotation>
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="rule" minOccurs="0">
      <xs:complexType>
        <xs:complexContent>
          <xs:extension base="ruleType"/>
        </xs:complexContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" type="xs:string">
    <xs:annotation>
      <xs:documentation>Name of default structural element as defined in the document.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
</xs:complexType>
<xs:complexType name="rulesetType">
  <xs:complexContent>
    <xs:extension base="baseRuleType">
      <xs:sequence>
        <xs:element name="document" minOccurs="0">
          <xs:annotation>
            <xs:documentation>The file name of the document that the ruleset represents.</xs:documentation>
          </xs:annotation>
          <xs:complexType>
            <xs:simpleContent>
              <xs:extension base="xs:string">
                <xs:attribute name="scope" type="xs:string">
                  <xs:annotation>
                    <xs:documentation>Reserved for Oracle Policy Modeling. This is a scope identifier appended to attributes

```

```

defined in the document and used to disambiguate attributes auto-assigned the same base name.</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="default-structural-element" type="xs:string" use="optional">
  <xs:annotation>
    <xs:documentation>Default structural element as defined in the document.</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="document-type" use="optional" default="Word">
  <xs:annotation>
    <xs:documentation>The type of document that the rules in this ruleset were sourced from - eg. Word or
Excel. Useful on import when allocating rules to project documents. Defaults to Word.</xs:documentation>
  </xs:annotation>
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="Word"/>
    <xs:enumeration value="Excel"/>
    <xs:enumeration value="Visio"/>
    <xs:enumeration value="Module"/>
    <xs:enumeration value="other"/>
  </xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="rules" type="rulesType">
  <xs:annotation>
    <xs:documentation>The collection of rules and rulesets that make up this ruleset.</xs:documentation>
  </xs:annotation>
</xs:element>
</xs:sequence>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="ruleType">
  <xs:complexContent>
    <xs:extension base="baseRuleType">
      <xs:sequence>
        <xs:element name="rule-text" type="ruleTextType"/>
        <xs:element name="plain-text" minOccurs="0">
          <xs:annotation>
            <xs:documentation>A plain text form of the rule text. This form is guaranteed to include on ASCII alpha-numeric
characters, spaces, tabs, carriage returns and line feeds.</xs:documentation>
          </xs:annotation>
        </xs:element>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>

```

```

    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="lang" type="xs:string"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
<xs:element name="rule-xml" type="xs:string" minOccurs="0">
  <xs:annotation>
    <xs:documentation>Reserved for Oracle Policy Modeling. This is the compiled XML form of the rule. This schema is
defined elsewhere.</xs:documentation>
  </xs:annotation>
</xs:element>
<xs:element name="participating-attributes" type="participatingAttributesType" minOccurs="0">
  <xs:annotation>
    <xs:documentation>The collection of attributes that participate in the rule. Participation may be as a conclusion
attribute, an intermediate conclusion attribute or as a condition.</xs:documentation>
  </xs:annotation>
</xs:element>
</xs:sequence>
<xs:attribute name="rule-loop" type="xs:boolean"/>
<xs:attribute name="rule-type" use="optional">
  <xs:annotation>
    <xs:documentation>Used to indicate that a rule is a shortcut rule. Can be used in the future to identify different types
of rules.</xs:documentation>
  </xs:annotation>
<xs:simpleType>
  <xs:restriction base="xs:string">
    <xs:enumeration value="shortcut"/>
    <xs:enumeration value="standard"/>
  </xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:extension>
</xs:complexContent>
</xs:complexType>
<xs:complexType name="ruleTextType">
  <xs:sequence>
    <xs:any namespace="http://www.w3.org/1999/xhtml" processContents="skip" maxOccurs="unbounded"/>
  </xs:sequence>
  <xs:attribute name="format" use="optional" default="text">
    <xs:annotation>
      <xs:documentation>Describes the original format of the rule in its source document - eg. free-form paragraph style (text),
tabular or graphical. Only text and table are supported. Graph is reserved for future use.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:simpleType>
    <xs:restriction base="xs:string">

```

```

    <xs:enumeration value="text"/>
    <xs:enumeration value="table"/>
    <xs:enumeration value="graph"/>
  </xs:restriction>
</xs:simpleType>
</xs:attribute>
</xs:complexType>
<xs:complexType name="propertyDefinitionType">
  <xs:sequence minOccurs="0">
    <xs:element name="help-text" type="xs:string" minOccurs="0"/>
    <xs:element name="values" minOccurs="0">
      <xs:complexType>
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
          <xs:element name="value" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
  <xs:attribute name="name" use="required">
    <xs:annotation>
      <xs:documentation>The name of the property. Must be unique for an object type.</xs:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:attribute name="applies-to" use="required">
    <xs:annotation>
      <xs:documentation>The type of object to which a property definition applies - project, rule, entity, attribute, relationship,
screen, control</xs:documentation>
    </xs:annotation>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="project"/>
      <xs:enumeration value="ruleset"/>
      <xs:enumeration value="rule"/>
      <xs:enumeration value="entity"/>
      <xs:enumeration value="attribute"/>
      <xs:enumeration value="relationship"/>
      <xs:enumeration value="screen"/>
      <xs:enumeration value="interactivedocument"/>
      <xs:enumeration value="control"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
  <xs:attribute name="type" use="required">
    <xs:annotation>
      <xs:documentation>The data type of the property - text, number, boolean or list</xs:documentation>
    </xs:annotation>
  <xs:simpleType>

```

```

    <xs:restriction base="xs:string">
      <xs:enumeration value="boolean"/>
      <xs:enumeration value="number"/>
      <xs:enumeration value="list"/>
      <xs:enumeration value="text"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>
<xs:attribute name="default-value" type="xs:string" use="optional"/>
<xs:attribute name="allow-blank" type="xs:boolean">
  <xs:annotation>
    <xs:documentation>True if an empty value for the property is permitted.</xs:documentation>
  </xs:annotation>
</xs:attribute>
<xs:attribute name="runtime" type="xs:boolean">
  <xs:annotation>
    <xs:documentation>True if the property is available at Runtime.</xs:documentation>
  </xs:annotation>
</xs:attribute>
</xs:complexType>
<xs:complexType name="propertyType">
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="name" use="required">
        <xs:annotation>
          <xs:documentation>The name part of the property. When combined with the type of the element that owns the prop-
erty (eg. project, ruleset, rule, entity, attribute, relationship), selects a property-definition that specifies the property's data type,
default value and, for lists, allowed values.</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:complexType name="participatingAttributeType">
  <xs:annotation>
    <xs:documentation>An attribute that participates in a rule, either as a concluded attribute or as a contributing attribute (ie.
appearing in a premise)</xs:documentation>
  </xs:annotation>
  <xs:attribute name="role" use="required">
    <xs:annotation>
      <xs:documentation>The role of the attribute in rule - ie. either a conclusion, an intermediate conclusion or a premise.</xs-
s:documentation>
    </xs:annotation>
  </xs:attribute>
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="conclusion"/>
      <xs:enumeration value="intermediate"/>
    </xs:restriction>
  </xs:simpleType>
</xs:complexType>

```

```

        <xs:enumeration value="premise"/>
    </xs:restriction>
</xs:simpleType>
</xs:attribute>
<xs:attribute name="entity-name" use="required">
    <xs:annotation>
        <xs:documentation>The name (unique identifier) of the participating attribute's entity.</xs:documentation>
    </xs:annotation>
</xs:attribute>
<xs:attribute name="attribute-name" use="required">
    <xs:annotation>
        <xs:documentation>The name (unique identifier) of the participating attribute.</xs:documentation>
    </xs:annotation>
</xs:attribute>
<xs:attribute name="abbreviation" type="xs:string" use="optional"/>
</xs:complexType>
<!--Collections-->
<xs:complexType name="attributesType">
    <xs:sequence>
        <xs:element name="attribute" minOccurs="0" maxOccurs="unbounded">
            <xs:complexType>
                <xs:complexContent>
                    <xs:extension base="attributeType"/>
                </xs:complexContent>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="attributeAliasesType">
    <xs:sequence maxOccurs="unbounded">
        <xs:element name="alias" type="attributeAliasType">
            <xs:annotation>
                <xs:documentation>An attribute that is bound to another attribute.</xs:documentation>
            </xs:annotation>
        </xs:element>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="entitiesType">
    <xs:sequence maxOccurs="unbounded">
        <xs:element name="entity" type="entityType"/>
    </xs:sequence>
</xs:complexType>
<xs:complexType name="participatingAttributesType">
    <xs:annotation>
        <xs:documentation>A collection of attributes that participate in a rule.</xs:documentation>
    </xs:annotation>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">

```

```

    <xs:element name="participating-attribute" type="participatingAttributeType">
      <xs:annotation>
        <xs:documentation>An attribute that participates in the rule. Participation may be as a conclusion attribute, an inter-
mediate conclusion attribute or as a condition.</xs:documentation>
      </xs:annotation>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="propertyDefinitionsType">
  <xs:annotation>
    <xs:documentation>A collection of property definitions specifying a property's type, allowed values, default value, etc.</x-
s:documentation>
  </xs:annotation>
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="property-definition" type="propertyDefinitionType"/>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="propertiesType">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="property" minOccurs="0">
      <xs:annotation>
        <xs:documentation>The value of the property.</xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="propertyType"/>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:complexType name="relationshipsType">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="relationship" type="relationshipType"/>
  </xs:sequence>
</xs:complexType>
<!--Other types-->
<xs:complexType name="definitionType">
  <xs:annotation>
    <xs:documentation>The definition of an object.</xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="lang" type="xs:language" use="optional"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

```

```
<xs:complexType name="questionFormType">
  <xs:annotation>
    <xs:documentation>Properties of the various forms that a boolean attribute may take (eg. negative, uncertain, ques-
tion).</xs:documentation>
  </xs:annotation>
  <xs:simpleContent>
    <xs:extension base="xs:string">
      <xs:attribute name="lang" type="xs:language" use="optional"/>
      <xs:attribute name="enabled" type="xs:boolean" use="optional" default="false">
        <xs:annotation>
          <xs:documentation>If false, indicates that the default form should not be overridden. Use to persist, but not imme-
diately use, a potential override form.</xs:documentation>
        </xs:annotation>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="synchronizationIdType">
  <xs:annotation>
    <xs:documentation>May be used by a system or application that supports generation and importing of project interchange
files to assign a persistent mapping identifier to a rulebase artifact - eg. to support round-tripping.</xs:documentation>
  </xs:annotation>
  <xs:restriction base="xs:string"/>
</xs:simpleType>
</xs:schema>
```

Decision reports

Topics in "Decision reports"

- Understand decision reports
- Retrieve a decision report

Understand decision reports

A decision report can be used to show why a goal, or any inferred attribute or relationship has been set to a particular value. It can also be used to show why a goal is unknown, and what information is need to resolve a goal.

A decision report is a "map" of the rules traversed in the rulebase in order to prove the current conclusion. Attributes and relationships that are proven by other attributes are displayed hierarchically, down to the level at which the user has entered data. It shows the means by which a decision is reached, showing the attributes, entities and relationships that contributed to the outcome.

If the outcome is unknown, the decision report indicates the unknown values that contributed to the outcome, giving you enough information to know what extra information needs to be provided to reach a decision.

Decision reports are available through the various deployments in the following way:

- Oracle Policy Modeling in the Debugger, through Show Decision and Investigate on attributes in the Debugger.
- The Determinations Engine API through the **getDecisionReport** method on the Attribute and Relationship interfaces.
- Determinations Server by specifying an outcome style of "decision-report".
- Web Determinations they are available by clicking on the **[why]?** link for a goal.

Retrieve a decision report

What do you want to do?

[Retrieve a decision report from Determinations Server](#)

[Retrieve a decision report from the Determinations Engine](#)

[Retrieve a decision report in an interview](#)

Retrieve a decision report from Determinations Server

A decision report is a recursive structure that contains a series of nodes, and explains the series of steps that led to an attribute having a particular value.

Retrieve a decision report from the Determinations Engine

For information on retrieving a decision report from the Determinations Engine, refer to [Get a decision from Determinations Engine](#) in the [Embed determinations inside another application](#) topic.

Retrieve a decision report in an interview

If the Interview Goal is an **AttributeGoal**, then the underlying **Attribute** object encapsulated may be used to retrieve a decision report. This is done by making the following API call:

```
attributeGoal.getAttribute().getDecisionReport(attributeGoal.getContext().findEntityInstance(interviewSession));
```

In the above code snippet the following is done:

1. The **attributeGoal** variable is set to an instance of **InterviewGoal** (of the **AttributeGoal** runtime type).
2. A query is made to retrieve its determinations engine **Attribute**, from which the request for the decision report is made.

In the example above we have used the simplest of the overloaded **getDecisionReport** methods. The **getContext** method call on the attribute goal retrieves the **InterviewEntityInstanceContext** object attached to this attribute goal - a unique identifier for the **InterviewEntityInstance** attached to the interview goal. From there, the utility method **findEntityInstance** can be used (with the variable **interviewSession** which is set to the interview session of which the attribute goal object is a goal to be investigated) to retrieve the underlying determinations engine **EntityInstance** to which the context refers.

Performance tuning in Oracle Policy Automation

The following sections document the parameters that the Oracle Policy Automation team tune as part of their internal performance testing. The aim of this list is to provide you with an overview of the areas to tune before deploying your application into production, or when performance problems arise. We have deliberately avoided providing any actual values for these parameters, as rulebase design, server configuration, network configuration and general environmental factors can all affect performance differently, meaning that the parameters used will more than likely differ for each customer.

We recommend measuring the affect of each change, so that it can be determined whether or not the change is having a negative or positive impact on performance. It is also recommended that before deploying your application into production, general performance testing is carried out so that the overall performance of the application is known, and you can plan your resources accordingly.

For information regarding performance tuning in Oracle Policy Automation, go to:

[Tune the performance of the JVM, Application Server and HTTP Server](#)

[Tune the performance of the Operating System](#)

For suggestions regarding the performance tuning of specific applications, refer to the following topics:

[Tune the performance of Web Determinations](#)

[Tune the performance of Determinations Server](#)

See also:

- [Run a clustered environment](#)
- [Resolve issues with multi threaded applications](#)

Tune the performance of the JVM, Application Server and HTTP Server

JVM

Tuning the JVM settings will most likely improve overall Oracle Policy Automation performance by increasing the effectiveness of how the JVM and garbage collector operate, therefore decreasing application response times and CPU usage while increasing throughput. When tuning the JVM performance, the following should be considered:

- **Heap size:**

It is recommended that the initial and maximum heap sizes are increased from the default values, which differ depending on the JDK that is being used. When adjusting the initial and maximum heap size, use GC logging or a JVM monitoring tool such as *JRockit Mission Control* or *Tivoli Performance Viewer*, to measure the impact of the change; adjust accordingly.

Additionally, it is recommended that you set both the initial and maximum heap sizes to the same value; this will prevent the VM from resizing the heap dynamically and will force the server to use all of the allocated memory from start-up.

Note: Increasing the heap size too much can result in long pause times, which in turn will affect throughput and application response time. Having the heap too small can at worst, cause *out of memory* exceptions and at best, cause frequent garbage collections, which increases CPU usage and application response time.

- **PermSize and MaxPermSize:**

If you are using the Hotspot JVM (this option is not supported on JRockit) then it is recommended that you increase both the

PermSize and the **MaxPermSize** parameters. If the permanent generation is too small, the JVM will do a full garbage collection of the heap before resizing; again, use GC Logging or a JVM monitoring tool to view the impact of the changes and adjust accordingly.

- **Add the `-server` option:**

If you are using the Hotspot JVM (in JRockit, this option is `-jrockit` and is on by default), performance gains may be achieved if the `-server` option is added to the Java start-up arguments. Although the JVM takes slightly longer to start and generally has a larger memory footprint, the `-server` option has been tuned to maximize peak performance and is generally used on longer running server applications which need the fastest possible operating speed.

- **AggressiveHeap:**

Adding `-XX:AggressiveHeap` (on Hotspot) or `-XXaggressive` (on JRockit) to the Java start-up arguments can provide increased performance for the JVM. The **AggressiveHeap** option inspects the machine resources (memory size, number of processors) and attempts to set various parameters which should in turn increase performance for long running, memory intensive applications. The downside to using the **AggressiveHeap** option is that the JVM uses more internal resources at start-up; however, it should not take long to reach a stable state after which it can perform at high speed.

Garbage collection scheme

Depending on the size and complexity of the rulebase or rulebases being used as well as the type of JVM used, increased JVM performance can be achieved by tuning the garbage collection scheme. For example a small rulebase with a large number of calculations may perform better after switching the garbage collection scheme from "generational concurrent" to "parallel" or vice versa.

Application Server

Tuning the Application Server can improve Oracle Policy Automation performance by allowing an increased amount of connections to service incoming requests, resulting in a higher throughput. If the logging settings and ratio of managed servers to CPUs are optimal, it will also allow for a higher number of transactions to be processed by the application server before the requests are queued or the server returns an error when overloaded. When tuning the application server's performance, the following should be considered:

- **Ratio of managed servers to CPUs:**

On large servers, consideration must be given to the ratio of managed server instances to the number of available CPUs. Generally the ratio is one managed server for every two processor cores; however this should be tested and adjusted accordingly, to meet your requirements.

- **Thread Pool size:**

On WebLogic, this is less of an issue as it has quite a good self-tuning thread pool; however, in some rare instances it may be beneficial to manually increase the thread pool size. If the thread pool is too small, application response times may be high, but throughput will suffer and CPU usage will most likely be low. Alternatively, if the thread pool is too large, then threads will be competing for CPU and Memory, resulting in high CPU usage and possible resource exhaustion.

- **Server logging:**

It is recommended that server logging is set to *Warning* or above.

- **Application servers are set to *Production* mode:**

Although not applicable to all application servers, in some cases the default install for the application server can be set to run in *Development* mode. Although this ensures the quick start up of application servers, most logging is set to *Debug* mode which slows performance and the JVM arguments used are not generally optimized for performance; there is also a relaxed security configuration. It is therefore recommended that the application server is run in *Production* mode.

- **The correct JDK is used:**

For some application servers, the JDK that is used impacts whether or not the native performance pack is enabled for the

application server. It is recommended that before installing the JDK and application server, the configuration is checked in the application server's system requirements guide.

- **OPA applications are 'warmed-up' after restarting / recycling the application server pool:**

Whenever the application server pool is restarted or recycled, it is recommended that OPA applications are 'warmed up' before being placed under load. This will ensure there is no adverse impact on performance. To 'warm up' an OPA application, send a single request to the server to bootstrap the application and wait for a response. Once this first response has been received successfully, the application is ready to be placed under load. Failing to 'warm-up' the application prior to high load may result in an *out of memory* exception, because the memory requirements could be beyond those experienced under normal load for the same number of users.

HTTP Server

If performance is not as expected and it is applicable, one area to look at early in the diagnostic process is the HTTP Server. Firstly, if possible attempt to bypass the HTTP Server by pointing the URL or endpoint directly at the managed servers and check if the performance has improved.

If bypassing the HTTP server resulted in improved performance, then you may need to tune the HTTP server settings such as **KeepAlive**, **MaxKeepAliveRequests**, **KeepAliveTimeout**, **ThreadsPerChild**, **MaxClients**, and **MaxRequestsPerChild** in the *httpd.conf* file, to increase performance. If tuning these settings does not improve the performance, then the problem is most likely **not** related to the HTTP Server and further diagnostics will be required to find the root cause.

Tune the performance of the Operating System

Tuning the operating system's TCP and file descriptor settings can have a positive impact on Oracle Policy Automation performance by allowing an increased amount of throughput to the application before the system return errors and requests get queued or rejected due to overload which then causes application performance to degrade.

Windows

TCP Settings

On Windows (2000, XP and Server 2003), there is a limitation where TCP connections from ports that are greater than 5000, result in the error message:

"An operation on a socket could not be performed because the system lacked sufficient buffer space or because the queue was full."

A workaround for this issue has been explained in the Microsoft Knowledge Base article KB196271 (<http://support.microsoft.com/kb/196271>).

AIX

TCP Settings

On AIX it is recommended that the TCP settings are tuned to increase performance; the following is a list of settings to start with initially:

- TCP keepidle
- TCP keepint
- TCP keepintvl
- TCP finwait2

- TCP somaxconn
- TCP rfc1323
- TCP init_window
- TCP nagle_limit
- TCP nodelayack
- TCP sendspace
- TCP recvspace

Note: This is not a definitive list; it is recommended that the effect of each setting is measured so that it is known whether adjusting it has a positive or negative impact on performance. In addition to the TCP settings, some settings on the network adapter (via the **chdev** command) may also result in performance gains; these settings are:

- Rfc1323
- TCP sendspace
- TCP recvspace

File Descriptors

Check the **ulimit** settings; they should be set to *unlimited* where applicable.

Solaris

TCP Settings

The TCP settings that should be changed from their default on Solaris systems are:

- tcp_conn_req_max_q
- tcp_conn_req_max_q0
- tcp_xmit_hiwat
- tcp_recv_hiwat
- tcp_naglim_def
- tcp_time_wait_interval

Again this is not a definitive list. It is recommended that the effect of each setting is measured so it is known whether adjusting it has a positive or negative impact on performance.

File Descriptors

Check the **ulimit** settings; they should be set to *unlimited* where applicable (core file size, data seg size, file size, cpu time and virtual memory).

Tune the performance of Web Determinations

Before deploying Web Determinations to your application server, there are a few settings that should be checked to ensure the best possible performance.

Java

The following steps assume that if the Java version of Web Determinations is being used, the application has been unpacked prior to making any configuration changes:

1. Under `WEB-INF\classes\configuration` open the `log4j.xml` file and make sure that `level value` is set at the very least to `warn` (the default value); for example:

```
<level value="warn" />
```

This will ensure that logging is not set to `debug`, which slows the overall response time of the application.

2. Under `WEB-INF\classes\configuration` open the `application.properties` file and make sure the `enable.debugger` property is set to `false`; for example:

```
enable.debugger = false
```

This will ensure that Web Determinations does not allow connections from the debugger. If set to `true`, this adds a small overhead which in turn affects performance.

.NET

The following steps describe the settings that need to be checked and possibly changed to ensure the best possible performance of the .NET Web Determinations:

1. Under the root Web Determinations folder, open the `Web.config` file and make sure that `compilation debug` is set to `false`; for example:

```
<compilation debug="false"></compilation>
```

This will ensure that extra debugging symbols are not inserted into any of the compiled pages, significantly increasing the performance of Web Determinations.

2. Under `bin\configuration`, open the `log4net.xml` file and make sure that `level value` is set at the very least to `warn` (the default value); for example:

```
<level value="warn" />
```

This will ensure that logging is not set to `debug`, which slows the overall response times of the application.

3. Under `bin\configuration` open the `application.properties` file and make sure the `enable.debugger` property is set to `false`; for example:

```
enable.debugger = false
```

This will ensure that Web Determinations does not allow connections from the debugger. If set to `true` this adds a small overhead which in turn affects performance.

Tune the performance of Determinations Server

Before deploying Determinations Server to your application server, there are some settings that should be checked to make sure they are set to the correct value for the best performance.

Java

The following assumes that if the Java version of Determinations Server is being used, the application has been unpacked prior to making any configuration changes:

- Under `WEB-INF\classes\config` open the `log4j.xml` file and make sure that `level value` is set at the very least to `warn` (the default value); for example:

```
<level value="warn" />
```

This will ensure that logging is not set to `debug`, which slows the overall response time of the application.

.NET

The following steps describe the settings that need to be checked and possibly changed, for optimal performance of the .NET Determinations Server:

1. Under the root Determinations Server folder, open the `Web.config` file and make sure that `compilation debug` is set to `false`; for example:

```
<compilation debug="false"> </compilation>
```

This will ensure that extra debugging symbols are not inserted into any of the compiled pages, significantly increasing the performance of the Determinations Server.

2. Under `bin\config`, open the `log4net.xml` file and make sure that `level value` is set at the very least to `warn` (the default value); for example:

```
<level value="warn" />
```

This will ensure that logging is not set to `debug`, which slows the overall response times of the application.

Run a clustered environment

Note: The following information about running a clustered environment relates to **Generic** and **Specific Assess** Services only

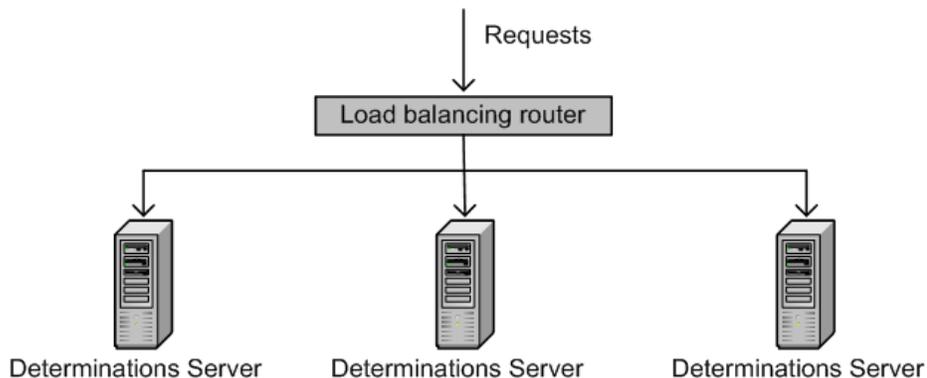
Determinations Server

It is very easy to increase the capability of the Determinations Server through clustering. In this case, clustering is a group of servers, providing the same Determinations Server service. Typically a cluster sits behind a router (either hardware or software) and passes requests to different servers in the cluster based on the idea of spreading the requests across all the servers the provide a large capacity for requests. This can be a very effective strategy for providing a high performance and high reliability service.

The Determinations Server operates stateless transactions. When it processes an operation, that operation is completed, leaving the Determinations Server in the same state as when it started. Because there is no state held between Determinations Server operations, it makes no difference which server in a cluster of Determinations Servers a particular client request uses.

Different Determinations Servers in the cluster do not need to communicate with each other, and there is no need for them to share session information, as no session information is kept between transactions. It is important to ensure that the rulebases are the same on each Determinations Server to ensure consistency between servers.

Clustering is done at the application server or server level. Most application servers have built in clustering.



Interview Service

Note that the Interview Service is stateful in order to run in a clustered environment, it requires an affinity based solution such as using "sticky sessions" on a load balancer.

Resolve issues with multi threaded applications

The following information outlines some of the issues surrounding multi threaded applications and the Oracle Policy Automation Determinations Engine API, and also presents a suggested application architecture that resolves these issues.

What do you want to do?

Understand the relationship between the Determinations Engine and installable handlers

Understand the relationship between rulebases and installable handlers

Understand the relationship between sessions Engine and installable handlers

Apply a recommended architecture solution

Understand the relationship between the Determinations Engine and installable handlers

The only class provided by the Oracle Policy Automation Determinations Engine API that can be created directly, is the Engine class. The Engine is a singleton object that is synchronized internally, so, even though its methods are not apparently synchronized by the Java API, all methods on the Engine are thread safe. In the case of the following methods:

- **addSessionListener**
- **removeSessionListener**
- **getSessionListeners**

These methods are synchronized against the Engine object.

Understand the relationship between rulebases and installable handlers

Rulebase object references are acquired by calling **GetRulebase()** against an Engine object.

The first time this method is called for a given rulebase path, the engine loads the rulebase file(s) from disk, and caches a reference to the loaded rulebase. Subsequent calls to **GetRulebase()** with the same path will return the cached reference.

Rulebase objects are the root of the static view of a rulebase, and as such, will be shared among threads in a multithreaded application. For this reason, all members of the rulebase are internally thread safe.

Note:

The rulebase path supplied to **GetRulebase()** is used as the key into the cached rulebases collection, and two different paths that refer to the same rulebase file will be considered as two different rulebase instances; that is, ".\MyRulebase.nrb" and "c:\zz\MyRulebase.nrb" might possibly refer to the same physical rulebase file, but Oracle will consider them to be different rulebase instances.

Understand the relationship between sessions Engine and installable handlers

Session object references are acquired by calling **CreateSession()** against an Engine object.

If the **SessionSynchronizationMode** passed in was SYNCHRONIZED, then all methods on the session and entity instance objects will be thread safe. Where synchronization is required, it will occur against the Session object.

If the **SessionSynchronizationMode** passed in was UNSYNCHRONIZED, then thread safety cannot be guaranteed by the engine.

The methods affected by synchronization include the following installable handler methods:

- **addInferencingListener**
- **removeInferencingListener**

- **getInferencingListeners**
- **addAttributeChangeListener**
- **removeAttributeChangeListener**
- **getAttributeChangeListeners**

If a managed session object is obtained by the **GetManagedSession()** method of an Engine object, the session will retain the same synchronization setting as at creation time. This can be queried by calling the **GetSynchronizationMode()** method of the Session object.

Apply a recommended architecture solution

The following section describes a technique that will ensure that rulebase handlers are only installed once per rulebase. It assumes that the application already has a synchronized 'bootstrap' routine, or can create a static initializer block on the main class.

Application initialization or 'bootstrap'

The application, during its initialization, should create an instance of the Engine(), and install a log handler, if desired, and install an instance of an **EngineEventHandler**.

EngineEventHandler:

```
// It is assumed that this method is called once per process, and is thread
safe.

void bootstrap()
{
    ...

    Engine engine = new Engine();

    // Optionally install a log handler
    If ( EngineLoggingEnabled )
    {
        engine.installLogHandler( new MyLogHandler() );
    }

    // And install an EngineEventHandler
    engine.installEngineEventHandler( new MyEngineEventHandler() );

    ...
}
```

The Engine Event Handler

```
public class MyEngineEventHandler
{
    public void OnRulebaseLoaded( Rulebase rb ) throws ExpertException
    {
        // This will only happen once per rulebase, so its perfect for installing the
        // handlers
        rb.installCustomFunctionHandler( new MyCustomFunctionHandler() );
        rb.setFormatter( new MyCustomFormatter() );
    }
}
```

Security

For information on securing your Web Determinations deployment, refer to the [Secure a Web Determinations deployment](#) topic, [Configure Web Determinations to run as a secure service](#).

See also:

[Secure particular rule sets](#)

[Secure the transport layer](#)

[Add user authentication to Web Determinations](#)

Secure the transport layer

Because the Determinations Server runs over standard web protocol, by default, information sent to and from the Determination Server is transmitted in plain text. If you have a requirement to secure communication to and from the Determinations Server, you should consider implementing a Secure Sockets Layer (SSL) protocol.

The implementation of SSL is generally done by the application server that is hosting the Determinations Server (IIS in the case of the .NET version of the Determinations Server), and you refer to the documentation of the application server you intend to use.

Implementing SSL generally involves obtaining a certificate to handle encryption, setting up the SSL, and then ensuring that all communications to the web application (in this case the Determinations Server) goes through the secure HTTPS instead of the unsecure HTTP port.

The Determinations Server will operate perfectly through HTTPS and requires no additional configuration for either Java or .NET.

Secure particular rule sets

What do you want to do?

Manage rules and responsibilities

Secure rulebases from unauthorized access and tampering

Manage rules and responsibilities

Oracle Policy Modeling provides tightly integrated source control for projects to support best practice on both individual and team-based projects. The source control functionality provided allows a great deal of flexibility.

Oracle Policy Modeling uses the Microsoft Visual Studio .Net SCCAPI to provide integration with a range of source control technologies.

Using Oracle Policy Modeling with Visual SourceSafe

To use Oracle Policy Modeling with Visual SourceSafe, simply install Visual SourceSafe on your development machine and set Visual SourceSafe as the default source control client for that machine (you do not need to do this if no other source control clients are installed).

Using Oracle Policy Modeling with other source control software

To use Oracle Policy Modeling with other source control software, that software must work with the Microsoft .Net SCCAPI. Oracle Policy Modeling has been tested using Source Offsite and Rational ClearCase and Microsoft Team Foundation Server, but other products using the SCCAPI should also work without any issues.

If you are using an alternative for source control, you must ensure that your source control client is set as the default source control client for your machine.

To use source control for your project:

- select the Subversion component during the Oracle Policy Modeling installation, or
- install Subversion (note that a command line Subversion client package is required for direct integration with Oracle Policy Modeling), or
- install one of the SCCAPI source control programs. If you have selected Microsoft Team Foundation Server as your SCCAPI program, see *Install Microsoft Team Foundation Server* in the *Oracle Policy Modeling User Guide* for more information.

Secure rulebases from unauthorized access and tampering

When a rulebase is deployed to Oracle Web Determinations or Determinations Server, it will either be inside the web application war file, or on a location in the file system that you have designated.

In order to ensure integrity of the application and the rules you should secure access to web application (Web Determinations or Determinations Server), the rulebase if it exists in a separate location. Only system administrators or those with responsibility for updating the deployed web application should have permission to access these files.

Authorized Access to a rulebase in Determinations Server

It is possible to restrict access to the operations of the Determinations Server through the security implemented on the application server on which the Determinations Server is running.

Add user authentication to Web Determinations

Oracle Web Determinations does not have any in-built support for authentication or secure access. To configure secure access, Web Determinations relies upon the security framework of the host Applications Server. Some pointers and procedures for configuring secure access for some application servers can be found in the [Secure a Web Determinations deployment](#) topic.

Technical reference

Topics in "Technical reference"

Paths to the API reference files

The various API Reference zip files that are installed with the application, can be found at the following locations:

Determinations Engine API

[C:\Program Files\Oracle\Policy Modeling\help\api\determinations-engine-doc.zip](#)

[C:\Program Files\Oracle\Policy Modeling\help\api\determinations-engine-net.zip](#)

Interview Engine API

Note that as of V10.4, some methods in the InterviewRulebase, ScreenOrder, ScreenOrderRegistry and EngineConstants classes have been deprecated. Details can be found in the API.

[C:\Program Files\Oracle\Policy Modeling\help\api\determinations-interview-engine-java-doc.zip](#)

[C:\Program Files\Oracle\Policy Modeling\help\api\determinations-interview-engine-net-doc.zip](#)

Determinations Server API

[C:\Program Files\Oracle\Policy Modeling\help\api\determinations-server-java-doc.zip](#)

[C:\Program Files\Oracle\Policy Modeling\help\api\determinations-server-net-doc.zip](#)

Masquerade API

[C:\Program Files\Oracle\Policy Modeling\help\api\masquerade-net.zip](#)

Web Determinations API

[C:\Program Files\Oracle\Policy Modeling\help\api\web-determinations-doc.zip](#)

[C:\Program Files\Oracle\Policy Modeling\help\api\web-determinations-net-doc.zip](#)

Also note that the default installation location "C:\Program Files\Oracle\Policy Modeling" may be changed by the user.

Decision Reports

[Return a Decision Report from the Determinations Server](#)

Determinations Server files

[Determinations Server configuration file](#)

[Determinations Server error catalog](#)

Determinations Server Services

[Server Service](#)

[Interview Service](#)

[Assess Service](#)

[Language, timezones and other localization concerns](#)

Determinations Engine .NET and Java API's

[Differences between the Determinations Engine .NET and Java API's](#)

[EventListener Interfaces](#)

[Capitalization](#)

[Enumerated Constants](#)

[Enumerators](#)

[Objects and Collections](#)

[Type Names](#)

Runtime - general information

[Handling of large decimal numbers](#)

Third party products and licenses

[Third party products and licenses](#)

Web Determinations configuration options

Note: As a general rule, to enable localization, UTF-8 encoding should always be used.

[Oracle Web Determinations URL API](#)

[Configure Oracle Web Determinations](#)

[Web Determinations configuration files](#)

[application.properties file](#)

[appearance.properties file](#)

[messages.<locale>.properties file](#)

[Manage rulebases](#)

[Rulebase listeners](#)

[IsHTML and Web Determinations customization](#)

[Velocity Templates Developer Guide](#)

[Template files](#)

[Oracle Web Determinations Template Reference Guide](#)

[Specify the user interface language for rule authoring on the command line](#)

Plugins

[Commentary - pseudo code](#)

[Data Adaptor - pseudo code](#)

[Formatter - pseudo code](#)

[List Provider - pseudo code](#)

[Understand the InterviewSession](#)

Legacy document generation

Custom Function extensions

Go to:

[extension.xml](#)

[Limitations](#)

[Class/Assembly loading of Handler classes](#)

[Requirements of CustomFunction class](#)

Extensions are located in a folder called *Extensions* in Oracle Policy Modeling's project folder. This folder does not exist by default, but must be created the first time an extension is added to the project.

The extensions folder can contain any number of other folders, each representing a self-contained extension that can be copied to another project. The extension folder is only required at build-time – once built, the *rulebase.zip* file is independent of any extensions it has used.

Every such extension folder must contain the following:

- **extension.xml** - an xml file listing the custom functions defined by the extension; this is used by Oracle Policy Modeling to verify the usage of custom functions called in the rule documents of the project.
- **lib/** - a folder containing the supporting JAR and/or DLLs required by the extension at runtime. The files in this folder are copied into the rulebase zip file when the project is built. The Determinations Engine then has access to these files to resolve class references in the extension.xml.

extension.xml

The extension.xml file is a place to store the list of functions exposed by the extension. Each function consists of:

Value	Description
name	The name used to invoke the function from a project rule document. The name must be unique and may not be the same as any built-in function.
return-type	The type of value returned by the function. Regardless of the return-type, a function can always return uncertain, unknown, or a temporal value (although each change point must be of the correct type).
arguments	Zero or more arguments that can be passed to the function, each with a name and a type. The name is not required but offered for documentation reasons. The type is required and is validated when the function is used, but unknown, uncertain or temporal values (with change points of the right type) are always acceptable values for an argument.
handler	One or two handler entries giving the Java and/or .NET class that provides the implementation of the custom function. A .NET class is recommended for compatibility with the debugger, but not required.
property	Zero or more key/value pairs (string values only) which are passed to the function's initialize() method. Properties must be uniquely named.

extension xml - full schema

The full schema for the extension.xml file is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
  xmlns:mstns="http://tempuri.org/Extension.xsd"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:simpleType name="value-type">
    <xs:restriction base="xs:string">
      <xs:enumeration value="boolean"/>
      <xs:enumeration value="text"/>
      <xs:enumeration value="number"/>
      <xs:enumeration value="date"/>
      <xs:enumeration value="datetime"/>
      <xs:enumeration value="timeofday"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="platform-type">
    <xs:restriction base="xs:string">
      <xs:enumeration value="java"/>
      <xs:enumeration value="dotnet"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:element name="extension">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="functions" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:element ref="function" minOccurs="0" maxOccurs="unbounded"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="function">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="arg" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="handler" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="property" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```

        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="return-type" type="value-type" use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="arg">
    <xs:complexType>
        <xs:attribute name="name" type="xs:string" use="optional"/>
        <xs:attribute name="type" type="value-type"/>
    </xs:complexType>
</xs:element>

<xs:element name="handler">
    <xs:complexType>
        <xs:attribute name="platform" type="platform-type" use="required"/>
        <xs:attribute name="class" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>

<xs:element name="property">
    <xs:complexType>
        <xs:attribute name="name" type="xs:string"/>
        <xs:attribute name="value" type="xs:string"/>
    </xs:complexType>
</xs:element>
</xs:schema>

```

Limitations

Custom functions have the following limitations:

- They cannot accept relationships as a parameter (a workaround is to pass the name of a relationship via a text parameter, however any rules invoking the custom function need extra conditions to ensure the engine can see their dependence on the relationship).
- Only one boolean parameter can be passed to a function, and it must be the last parameter. This is due to a limitation in the rule compiler. Any number of other parameters may be present however.
- If you have modules and want to use extensions, you must define them in the main project; extensions defined in modules will not be loaded.

Class/Assembly loading of Handler classes

The class named in the 'handler' element of the [extensions.xml](#) file is a fully qualified class including the package (Java) or namespace (.NET). The class must extend the CustomFunction abstract class from the com.oracle.determinations.engine package (Java) or Oracle.Determinations.Engine namespace (.NET).

When instantiating the class, the Determinations Engine looks in:

- JAR files that were present in the LIB folder of the extension (Java) when the rulebase was built.
- Assembly DLL files that were present in the LIB folder of the extension (.NET) when the rulebase was built.
- The environment of the calling application.

This also applies any dependencies the class has; for example, if the custom function class uses classes from a utility library, that library may be copied into the extension's LIB folder or added as a reference to the calling application.

The disadvantage of placing dependencies in the calling application is that the rulebase won't work in the Debugger, so including dependencies in the LIB folder is highly recommended. The one exception is the Determinations Engine itself, which will be a dependency due to the use of the CustomFunction class, but does not need to be included since it will always be present in memory when the custom function is loaded.

Requirements of CustomFunction class

The custom function class specified in the <handler> element of the extensions.xml file should have the following properties:

Property	Description
Thread-safe	The class is instantiated once per rulebase, and all sessions created through that rulebase share the same custom function object. This means any member variables (cached values, initialized state, etc) must be protected for thread-safety.
Efficient	The Determinations Engine is optimized on the assumption that re-evaluation of a rule is more efficient than preserving everything about how a rule was evaluated. This means sometimes a function is called multiple times over the course of a session, when it seemingly shouldn't need to. This optimization bears out for most functions, however a very inefficient custom function can undermine this optimization and overall performance of the engine will suffer as a result.
Consistency	The above assumptions about re-evaluation of a rule also mean that functions are expected to return the same value when it is called with the same parameters. This makes a custom function unsuitable for some purposes, eg. generating random numbers.

The following is a list of all methods that may/must be overridden and the reasons for doing so. Only the evaluate() method is a necessary override. The java method signature has been shown but the .NET signatures are essentially the same, except with standard .NET capitalization (for example, Evaluate instead of evaluate).

Method	Description
void initialize(Map properties)	The first method called on the custom function. The Map object contains the key/value pairs specified as <property> in the extensions.xml file, while the FilesContainer allows other files to be loaded from the compiled rulebase zip. This is an optional override and the default implementation does nothing.
Object evaluate (EntityInstance instance, Object [] parameters)	The method called by the engine to evaluate the function. The instance parameter provides the context for the evaluation, and the parameters array contains boxed types of each parameter (java.lang.Double, java.lang.Boolean, etc in Java or Oracle.Masquerade.Lang.Boolean, etc in

Method	Description
	<p>.NET). The method should return a boxed result.</p> <p>Note that when the <code>getTemporalMask()</code> method has been overridden for a parameter, the <code>TemporalValue</code> class may be used (when a value has change-points) instead of a boxed type. Each change point value will be of the required boxed-type however.</p> <p>An uncertain and unknown (either as a parameter or result) is represented by <code>Uncertain.INSTANCE</code> and <code>null</code> respectively, but the method will never be called with uncertain or unknown parameters unless the <code>requireKnownParameters()</code> method is overridden to return <code>false</code>.</p> <p>This method must be overridden.</p>
<code>boolean requireKnownParameters() ()</code>	<p>This method is called by the engine to decide if unknown or uncertain values should be passed through to <code>evaluate()</code>.</p> <p>If this method returns <code>true</code>, then they are not and a result of unknown or uncertain (as appropriate) is assumed for the function. Decision reports are also handled automatically and the <code>markRelevance()</code> method is never called (the engine assumes all parameters must be relevant because if any of them were unknown or uncertain, it would affect the result).</p> <p>If the method returns <code>false</code>, the <code>evaluate()</code> method must be written to deal with unknown and uncertain as possible parameters.</p> <p>This is an optional override and the default implementation returns <code>true</code>.</p>
<code>void markRelevance(EntityInstance instance, Object[] parameters, boolean[] relevance)</code>	<p>This method is used to flag values that are required for the function to be evaluated. This is used for:</p> <ul style="list-style-type: none"> • Decision reports (only relevant values are shown) • Investigating a goal (any unknown values marked as relevant will be asked to the user). <p>The instance and parameters are the same as they would be for the <code>evaluate()</code> method. The relevance is supplied the same length as the parameters array, and should be updated by the method. Each parameter within the parameters array that is considered to affect the result of the function should be flagged with the value <code>true</code> in the relevance array.</p> <p>As an example, consider a custom function called <code>InlineIf(<true-value>, <false-value>, <condition>)</code> that returns the <code>true-value</code> if the condition is true and the <code>false-value</code> if the condition is false. When the condition is true, the condition and the <code>true-value</code> are relevant but the <code>false-value</code> is not, because it cannot affect the result. When the condition is false, the condition and the <code>false-value</code> are relevant but the <code>true-value</code> is not.</p> <p>This method is an optional override and the default implementation simply flags all parameters as being relevant.</p>
<code>boolean[] getTemporalMask()</code>	<p>This method is used to tell the determinations engine which parameters to the custom function are expected to be temporal values (values with change-points). The method should return an array of <code>Booleans</code>, with one <code>Boolean</code> value per parameter.</p> <p>For example, a custom function <code>NumberOfDaysOfTruth(<start-date>, <end-date>, <condition>)</code> which counts the number of days the condition is true, should return:</p> <pre>new boolean[] { false, false, true }</pre>

Method	Description
	<p>This means start date and end date should not have change-points but for the condition, change-points are allowed (in fact desirable).</p> <p>Regardless of the return value of this method, all parameters to a custom function are allowed to be temporal values. However the evaluate method won't see change points for parameters whose temporal mask is set to false. Instead the determinations engine calls evaluate() separately for each period of time defined by the change-dates.</p> <p>This method is an optional override, and the default implementation selects no parameters as temporal.</p>

Differences between the Oracle Determinations Engine .NET and Java API's

As the Oracle Determinations Engine API is essentially the same for .NET as it is for Java, the one set of documentation has been provided for both deployments. There are however, some minor differences:

Oracle Determinations Engine for .NET API:

The default installation location for Oracle Determinations Engine is **C:\Program Files\Oracle\Determinations**.

The C# interface is in the Oracle.Determinations.Engine namespace, you should include the following statement in C# files:
`using Oracle.Determinations.Engine;`

Within the C# project, you must create a reference to the libraries containing the Oracle Determinations Engine, and associated runtime services, which are installed at:

`<installation-location>\Engine\Oracle.Determinations.Engine.dll`
`<installation-location>\Engine\Oracle.Determinations.Masquerade.dll`

Oracle Determinations Engine for Java API:

The default installation location for Oracle Determinations Engine is **C:\Program Files\Oracle\Determinations**.

Java programmers wishing to use the Java interface to Oracle Determinations Engine should include the following package:
`import com.oracle.determinations.engine.*;`

In order for the Java Virtual Machine to find the **determinations engine** classes, the **CLASSPATH** environment variable should be modified to include the location of the **determinations-engine.jar** file. This path will be:

`<installation-location>\Engine\determinations-engine.jar`

Depending on the version of Java being used, it may be necessary to include other libraries, namely the XML parsing libraries. These paths will be:

`<installation-location>\Engine\jsr173_api.jar`
`<installation-location>\Engine\log4j-1.2.16.jar`
`<installation-location>\Engine\sjsxp.jar`

Other differences between the .NET and Java API's are described in the following topics:

[Capitalization](#)

[Enumerators](#)

[Enumerated constants](#)

[Objects and collections](#)

[Type names](#)

Note: The various API Reference zip files are installed with the application at the following locations:

C:\Program Files\Oracle\Policy Modeling\help\api\determinations-engine-doc.zip

C:\Program Files\Oracle\Policy Modeling\help\api\determinations-engine-net.zip

C:\Program Files\Oracle\Policy Modeling\help\api\masquerade-net.zip

C:\Program Files\Oracle\Policy Modeling\help\api\web-determinations-doc.zip

Also note that the default installation location "C:\Program Files\Oracle\Policy Modeling" may be changed by the user.

EventListener Interfaces

There are three **EventListener** interfaces: the **DecisionReportFilter**, **Inferencing Listener** and **AttributeChangeListener**.

For a description of the **DecisionReportFilter**, see:

[Decision Report Filter](#)

[The DecisionReportFilter interface](#)

[Installing a DecisionReportFilter](#)

[Example: Hide eligibility criteria from a decision report depending on benefit applied for](#)

For a description of the **InferencingListener**, see:

[Customize the inferencing cycle with custom functions and inferencing listeners](#)

The **AttributeChangeListener** is for advanced usage and should not be implemented in a production rulebase. More information can be found in the Determinations Engine API.

Decision Report Filter

Decision report filters are custom classes (Java or C#) that are authored by a software engineer to filter a decision report using "silent" and "invisible". They are useful when the out-of-the-box decision report options in Oracle Policy Modeling ("silent if true", "silent if uncertain" and so on) are insufficient.

For example considering the following rule determines eligibility for two separate benefits and allows a user-choice as to which benefit is being assessed:

the applicant is eligible if

both

the applicant is applying for benefit A and
the applicant meets the criteria for benefit A

or

both the applicant is applying for benefit B and
the applicant meets the criteria for benefit B

With this rule, if the applicant does not meet the criteria for either benefit, then both sets of criteria are considered relevant and included in a decision report, regardless of which benefit the applicant was actually applying for. This is correct logically (both are false values that led to a false conclusion) but probably confusing to an end-user who expects to see only the criteria for the benefit they have chosen to apply for.

With a decision report filter, the criteria for each benefit can be hidden conditionally, based on the selected benefit.

The **DecisionReportFilter** interface

This interface consists of two methods: **getAttributeFilter** and **getRelationshipFilter** which are used to add additional silent and invisible flags to a particular attribute or relationship in a decision report. When generating a decision report, the engine will invoke one of these methods for every attribute instance or relationship instance that appears in the decision report. The return value (an object of type **DecisionReportFilterResult**) determines what additional flags will be applied. The most common options will be **DecisionReportFilterResult.SILENT_INVISIBLE** and **DecisionReportFilterResult.UNFILTERED**.

Note that if the engine has already decided to make the attribute/relationship silent or invisible (because they are flagged as such in Oracle Policy Modeling), then the decision report filter cannot remove that property. For example a return value of **DecisionReportFilterResult.SILENT** will make the decision report entry 'silent' but will not affect its 'invisible' status.

Installing a **DecisionReportFilter**

DecisionReportFilter objects are attached to a Session object in the determinations engine with the **Session.setDecisionReportFilter** method, and will then apply to every decision report requested from that session. Every session can share the same **DecisionReportFilter** object if necessary, which is recommended, to ensure efficiency and consistency of behaviour. It is recommended that a decision report filter have no mutable state after being initialised, as multiple sessions could invoke the listener at the same time. At the very least, the object must be thread-safe.

A **RulebaseListener** (see the topic [Rulebase Listeners](#)) is the simplest way to ensure the **DecisionReportFilter** is automatically set after the session is created, and also provides a place to do one-time initialization of the filter when the rulebase is first loaded.

Capitalization

Method names in Java start with an initial lowercase letter and the first letter of every embedded word in uppercase, whereas method names in .NET follow a different convention in which the initial letter of every word in a method name is always uppercase. For example:

Expert API for Java	Expert API for .NET
<code>mySess.getGlobalEntityInstance();</code>	<code>mySess.GetGlobalEntityInstance();</code>

When using the Java Guide to the Oracle Determinations Engine API, it is important to remember to capitalize the first letter of each method name.

Enumerators

This is probably the point of largest difference between using Oracle Determinations Engine in Java and using Oracle Determinations Engine in .NET. Both platforms include enumerators (or iterators) in the standard class library and the two different enumerators behave differently. Each of the Oracle Determinations Engine APIs emulates the behavior of the enumerator native to the platform.

Oracle Determinations Engine API for Java

Enumerating over all the child entity instances of an entity instance in Java would be done with the following code:

```
EntityInstanceEnumerator children = ei.children();
while (children.hasNext())
{
    EntityInstance child = children.next();
    // Do something to the child
}
```

Oracle Determinations Engine API for .NET

The IEnumerator interface in .NET provides a different interface. The language C# also provides a foreach statement to simplify use of lists. The Oracle Determinations Engine API for .NET supports this feature, which allows the above code to be written as:

```
foreach (IEntityInstance child in ei.Children)
{
    // Do something to the child;
}
```

This also works for methods that return multiple items but are not strictly collections, such as Session.SelectEntityInstances:

```
foreach(IEntityInstance currentInstance in sess.SelectEntityInstances("/p1"))
{
    // Do something to currentInstance;
}
```

Enumerated Constants

Java and .NET vary in their support of enumerated constants. Java does not provide native support for them and .NET does. The Oracle Determinations Engine API for Java supports enumerated constants through static instances of a class that represents the particular enumeration.

Oracle Determinations Engine API for Java

Constants in the Expert API are represented as final static instances of a class describing the type of constant. For example, the Expert API exposes a class called `SexType`, that provides four static members to represent the possible sex types, Neuter, Male, Female, and Generic. Therefore, to set the sex of an attribute instance, the following is required:

```
personAttr.setSex(SexType.Female);
```

Oracle Determinations Engine API for .NET

Oracle Determinations Engine API for .NET exposes a collection of enumerated types. These types include constants for each of the acceptable values, for example:

```
personAttr.SetSex(SexType.Female);
```

Objects and collections

This is probably the point of largest difference between using the Oracle Determinations Engine in Java and using the Oracle Determinations Engine in .NET. Both platforms include numbers, collections, and maps in the standard class library. Each of the Oracle Determinations Engine APIs emulates the behavior of the Java platform..

Oracle Determinations Engine API for Java

Enumerating over all the entity instances of a given entity in Java, setting a new value for each instance, would be done with the following code:

```
Attribute attr = entity.getAttribute("a1");
Object value = new Double(2.0d);
```

```
List <EntityInstance> entityInstances= entity.getEntityInstances(session);
```

```
for (EntityInstance ei : entityInstances) {
    attr.setValue(ei, value);
}
```

Oracle Determinations Engine API for .NET

The Oracle.Determinations.masquerade.util.List interface provided in the .NET engine provides the equivalent interface. The language C# provides an equivalent **foreach** statement to simplify use of lists. The Oracle Determinations Engine API for .NET supports this feature, which allows the above code to be written as:

```
RBAttr attr = entity.getAttribute("a1");
List entityInstances = entity.GetEntityInstances(session);
Object value = new Oracle.Determinations.Masquerade.Lang.Double(2.0d);
```

```
foreach (EntityInstance ei in entityInstances) {
    attr.setValue(ei, value);
}
```

Type names

The exposed classes are named differently in the Java and .NET APIs. In Java, there aren't any prefixes attached to type names. However in .NET, each of the exposed classes is really an interface that cannot be directly instantiated. Therefore, to follow the .NET convention, each type is prefixed by an "I"; for example:

Oracle Determinations Engine API for Java	Oracle Determinations Engine API for .NET
EntityInstance	IEntityInstance

Rulebase configuration file

The Oracle Determinations Engine provides the capability to load the Custom Function Handler, Inferencing Listener and Custom Formatters directly into the Session itself, thereby creating a unified configuration method for all client applications including:

- Oracle Policy Modeling
- Oracle Determinations Server
- Oracle Policy Modeling Standalone Debugger
- Oracle Policy Modeling Standalone Regression Tester
- Oracle Determinations Engine

Oracle Web Determinations requires custom formatters to be configured as a Web Determinations plugin to support both output and input; refer to the topic [Formatter plugin overview](#).

Configuration file location

Configuration for custom function handlers, inferencing listeners and formatters is done via the Oracle Determinations Engine configuration file. This can be either an xml file or a properties file, either of which can be created using a text editor such as notepad. In order for the engine to use this file must be located in the same directory as the rulebase to which it applies and have the name <rulebase>-config.xml or <rulebase>-config.properties. For example, if the rulebase was called *OPM Example Rulebase* then the corresponding XML configuration file must be called *OPM Example Rulebase-config.xml*.

Note:

To have the configuration file automatically included in the compiled rulebase zip file, it should be copied into the **include** folder for the project.

XML contents

A rulebase configuration that configures both an inferencing listener and a custom formatter will look something like:

```
<configuration>
  <formatter platform="Java" library-path="basic-formatter.jar"
    class="oracle.test.TestFormatter"/>
  <inferencing-listener platform="Java" library-path="basic-inferencing-listener.jar"
    class="oracle.test.TestInferencingListener"/>
</configuration>
```

where:

configuration

the root element of the configuration file.

formatter

the name of the element which contains the formatter configuration parameters.

platform

Specifies which platform this configuration this custom component is written for. The valid values are:

- Java – specifies that the custom component is to be used with the java engine.
- DotNet – specified that the custom component is to be used with the DotNet engine.

library-path

Specified the location of the library which implements the custom component and its dependencies. The library path can either be absolute or relative to the location of the configuration file. Multiple libraries can be specified as a `;` list and will be loaded in the order in which they are specified.

class

The class (including the package or namespace) which implements either the CustomFunctionHandler interface, the InferencingListener interface or the Formatter interface.

XML schema

The full schema for the custom function handler is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns="http://oracle.com/determinations/engine/relational/configuration"
  targetNamespace="http://oracle.com/determinations/engine/relational/configuration"
  elementFormDefault="qualified" version="10.1.0:20100301">
  <xs:simpleType name="platform">
    <xs:restriction base="xs:string">
      <xs:enumeration value="DotNet"/>
      <xs:enumeration value="Java"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:complexType name="ConfigurationType">
    <xs:attribute name="platform" type="xs:string" use="required"/>
    <xs:attribute name="library-path" type="xs:string" use="required"/>
    <xs:attribute name="classpath" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:element name="configuration">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="custom-function" type="ConfigurationType" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="formatter" type="ConfigurationType" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="inferencing-listener" type="ConfigurationType" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Properties contents

A rulebase properties configuration that configures both a custom function handler and a custom formatter will look something like:

```
custom-function.java.library-path = TestCustomFunction.jar
custom-function.java.class = oracle.testrulebase.TestCustomFunction
formatter.java.library-path = basic-formatter.jar
formatter.java.class = oracle.test.TestFormatter

inferencing-listener.java.library-path = basic-inferencing-listener.jar
inferencing-listener.java.class = oracle.test.TestInferencingListener
```

All property names follow the pattern <object>.<platform>.<setting> where:

object

Specifies which object this configuration property is for. The valid values are:

- inferencing-listener – specifies that the property configures an inferencing listener.
- custom-function – specifies that the property configures the active custom function handler.
- formatter – specifies that the property configures the active custom formatter.

platform

Specifies which platform this configuration this custom component is written for. The valid values are:

- java – specifies that the property is to be used with the Oracle Determinations Engine for Java.
- dotnet – specifies that the property is to be used with the Oracle Determinations Engine for .NET.

setting

Specifies which setting this configuration property is for. The valid values are:

- library-path – Specified the location of the library which implements the custom component and its dependencies. The library path can either be absolute or relative to the location of the configuration file. Multiple libraries can be specified as a ';' list and will be loaded in the order in which they are specified.
- class – The name of the class (with package name) which implements either the CustomFunctionHandler interface for custom function handlers, or the Formatter interface for custom formatters.

Rulebase Listeners

Rulebase Listeners are custom objects that are created when a rulebase is loaded, and called every time a new session is created. You can create a rulebase listener to perform early initialization of new sessions before they are returned to the calling application; for example, a rulebase listener can:

- Install a formatter, inferencing listener or a legacy custom function handler.
- Preload data into a new session.

include/rulebase-listeners.xml

The *rulebase-listeners.xml* file is used to configure rulebase listeners with Oracle Policy Automation. It should be present in the 'include' folder of an Oracle Policy Modeling project, which will cause it to be included in the compiled rulebase when the project is built. The structure of the file is as follows:

```
<listeners>
  <listener>
    <handler platform="java" class="..."/>
    <handler platform="dotnet" class="..."/>

    <property name="foo" value="123"/>
    <property name="bar" value="abc"/>
    ...
  </listener>
  ...
</listeners>
```

<listener> element

Each **<listener>** element describes a rulebase listener and there may be multiple **<listener>** elements if desired.

<handler> element

The **<handler>** element gives the fully qualified class name (for example, **com.oracle.example.SampleListener**) of the class that implements the **RulebaseListener** interface. This class should be implemented in a JAR or DLL from the "lib" folder (see below). The **<handler>** element can be omitted for an unsupported platform (in which case no rulebase listener will be created), however it is highly recommended to provide a .NET version if possible, to ensure the rulebase works in the Test Script Editor inside Oracle Policy Modeling.

<property> element

The **<property>** element supplies extra parameters to the **initialize()** method of the rulebase listener. There may be any number of **<property>** elements (or none at all), and the interpretation of these properties is entirely up to the rulebase listener.

include/lib folder

This folder should contain the .NET assembly DLLs and/or Java JAR files that are required by the rulebase listener, including the actual implementation of the listener and any dependencies (other than the determinations engine itself). The structure of the lib

folder is unimportant, as the determinations engine simply loads every DLL/JAR within this folder when the rulebase is loaded. DLL and JAR files outside of this folder will not be loaded.

Note that the libraries in the lib folder are not actually loaded from there – they are copied into the compiled rulebase and extracted at runtime, so the compiled rulebase has no dependency on the project's include folder.

RulebaseListener interface

The **RulebaseListener** interface contains an **initialize()** method that provides the initialization properties described above, and a FilesContainer to read files from the compiled rulebase. Files that were placed in the project's "include" folder will be available via the FilesContainer. This method will be called exactly once per rulebase, when the rulebase is first loaded but before it is returned to the host application.

The **sessionCreated()** method is called every time a new session is created, and is where custom initialization of the session may take place. Calling applications may create sessions in a multi-threaded manner (for example, Oracle Determinations Server may do so when serving multiple requests simultaneously), so this method must access shared data in a thread-safe manner.

See also:

[Example: Create a Rulebase Listener to preload reference data](#)

Handling of large decimal numbers

The maximum precision for a number or currency value in the rule engine is 13 significant figures.

Attempting to set values or perform calculations on numbers that contain a larger number of significant figures may cause precision to be lost and appear in the Debugger and Web Determinations rounded to the 13th significant figure.

Oracle Determinations Server configuration file

The Oracle Determinations Server can be configured using the *application.properties* file, located at:

- `<webroot>/bin/config` in .NET.
- `<webroot>/WEB-INF/classes/config` in Java.

Some important points to note are:

- Properties are declared according to the Java property file standard; essentially they're a key value list of values.
- ',' is interpreted as a list separator and so will normally need to be escaped. To do this write '\,' instead of just ','.
- All files must be saved as UTF-8 text files.
- Property names and values are case-sensitive.
- If you have set up your application to cache properties, templates and so on, then you will need to restart your application server.

Click on the appropriate link:

Rulebase loading properties:

Name	Description
load.rulebase.from.classpath	Java only. Set to true if you want the rulebases to be loaded from the classpath rather than the file system. This value must be set to true in order to deploy to Java application servers that do not explode the war file by default; for example, WebLogic
rulebase.path	The path to the directory to containing the rulebases. If load.rulebase.from.classpath = true then this path must be relative to the 'classes' directory. Otherwise the path can either be absolute or relative to the webroot.
cache.loaded.rulebases	If true rulebases will be cached and the server will required to be re-started in order to pick up rulebase changes. Otherwise it will operate in hot swapping mode which allows rulebase updates to be picked up without a server re-start. Note: if load.rulebase.from.classpath = true then this property will always effectively be true
screens.validate.html	Screens file content can include HTML authored by users in oracle policy modelling as static content. If this option is set to true, screen file content will be scanned at application start time to verify that all HTML tags contained in screens files are in the whitelist of allowable content. See the following option screens.html.tags.whitelist .
screens.html.tags.whitelist	This a semicolon-delimited list of HTML tags that are permitted within screens files. If screens.validate.html = true, any tag in a screens file that is not in this list will cause an exception to be thrown during rulebase loading and the rulebase will not be available.

General Determinations Server properties:

Name	Description
enable.request.validation	Set to true to enable validation of all requests made to the determination server against the relevant WSDL. This is for debugging purposes - the performance penalty for enabling this option is such that it should not be enabled in production.
enable.response.validation	Set to true to enable validation of all requests made to the determination server against the relevant WSDL. This is for debugging purposes - the performance penalty for enabling this option is such that it should not be enabled in production.
enable.second.person	Set to true to enable second person text substitution on rulebases that support it.
response.outcomes.only	Set to true to only return data in the response that matches the defined outcomes (return outcomes only). The default value is "false" to return all relevant data in the response.

Interview Service properties:

Name	Description
deploy.interview.service	This allows you to turn off the deployment of the interview service.
interview.session.timeout	This is the time in minutes for which an interview service session can be idle before it times out.

Plugins properties:

Name	Description
plugin.libraries	Java only. This property is used for web application servers where class path introspection is not possible such as WebLogic when the war file is not exploded. In that case, this property contains the ';' list of fully qualified plugin classes to load and the plugin libraries must reside in a directory that will automatically be loaded on the classpath

Determinations Server error catalog

The following tables contain the SOAP Server errors and the SOAP Client errors that are returned by the Determinations Server. They contain the fault code, description and message or messages which provide more detail relating to the description of the error. All errors have the path: `com.oracle.determinations.server.exceptions.<error code>`

Server errors

Error code	Description	Message(s)
DeterminationsServerSerializationException	An exception thrown due to an error attempting to serialise the response.	Could not write response: {reason}
DuplicateActionException	An error that has occurred due to an attempt to register multiple operations for the same soap action	An action has already been registered for SoapAction '{soap action}' in service '{service name}'.
DuplicateServiceEndpointException	An error caused by an attempt to register multiple services against the same endpoint.	Duplicate Endpoint detected for: {end point name}. Registered class: {registered plugin}. Duplicate class: {duplicate plugin}
MissingPropertyException	Thrown when a mandatory property is not specified in a configuration file	Missing mandatory property '{property}' in file '{file name}'.
MissingResourceException	Thrown when a required resource is missing	Could not find resource: {resource path}
ResourceLoadException	Caused by an error when trying to load a resource	Failed to load resource: {resource path}
ResponseValidationException	Caused by a the determinations server attempting to send a response that is not valid according to the WSDL	Invalid response: {reason}
RulebaseLoadException	Caused by a failure to load the specified rulebase	Rulebase '{rulebase ID}' cannot be loaded. Reason: {reason}
TemplateException	Represents an error attempting to create or merge a velocity template.	
ThinkException	An error that occurs during the inferencing cycle when	Could not reach a determination for rulebase '{rule-

Error code	Description	Message(s)
	conducting an assessment	base ID}'. Reason: {reason}
WSDLException	Caused by an error attempting to generate the WSDL	

Client errors

Error code	Description	Message
AttributeValueException	Exception thrown when an attempt is made to assign a wrong value to attribute	Invalid value '{value}' for attribute '{attribute ID}' in instance '{entity ID}{{instance name}}'. Reason: {reason}
DeterminationsServerParseException	An exception thrown by the due to an error that occurred parsing the request.	Cannot parse request: Reason: {reason}
EntityInstanceCreationException	An error caused while attempting to create an entity instance.	Cannot create entity instance '{entity ID}{{instance name}}'. Reason: {reason}
InvalidActionException	Thrown when a request specifies a SOAP Action that does not exist	No action for '{soap action}' exists in service '{service name}'.
InvalidLanguageException	Caused by requesting a session in a language not supported by the rulebase.	Invalid language '{language}' for rulebase '{rulebase name}'.
InvalidRelationshipTargetException	Caused by attempting to specify an invalid target of a relationship	'{instance ID}' is not a valid target of relationship '{relationship ID}' in entity instance '{source entity instance}'.
InvalidRequestException	Thrown due to an invalid request being made.	
InvalidTimeZoneException	Thrown if the specified time zone in the request is not valid	The specified timezone does not match the expected timezone. Expected '{expected timezone}', Actual: '{actual timezone}'.

Error code	Description	Message
NoSuchServiceException	<p>Thrown when a request is made to an endpoint for which there is no service, usually because the rulebase has not successfully deployed.</p> <p>This is essentially the SOAP version of a http 404 response.</p>	There is no service registered at: {service end point}
OperationNotSupportedException	<p>Thrown when a request invokes an operation that is no longer supported.</p>	<p>The operation '{operation name}' is not supported.</p> <p>The operation '{operation name}' is not supported. Try using '{alternative}' instead.</p>
OrphanedChildEntityException	<p>Caused when the request specifies a entity instance that has no parent</p>	<p>The following instance(s) of entity '{entity name}' have no parents: '{instances}'. Each instance of entity '{entity name}' must be set as a target of the relationship {relationship name} in entity {entity name}.</p>
RelationshipSetException	<p>Caused by an error attempting to set a relationship</p>	<p>Cannot set relationship '{relationship ID}' in entity '{entity ID}'. Reason: {reason}</p>
RequestValidationException	<p>Caused because the request is not valid according to the WSDL</p>	Invalid request: {reason}
ScreenUpdateDataException	<p>Thrown due to a problem with the screen update data received in an interview service request</p>	Invalid screen update data: {message}
SessionCreationException	<p>Caused when a rule session can not be created</p>	<p>Error creating session for rulebase '{rulebase ID}' with language '{language}'. Reason: {reason}</p>
UnknownAttributeException	<p>Exception thrown when trying to set an attribute that does not exist for an entity</p>	Attribute '{attribute ID}' does not exist in entity '{entity ID}'.
UnknownEntityException	<p>Exception thrown when trying access an entity that does not exist in the rulebase</p>	Entity '{entity ID}' does not exist.

Error code	Description	Message
UnknownEntityInstanceException	Occurs when attempting to access an entity instance that does not exist	Entity instance '{instance ID}' does not exist.
UnknownRelationshipException	Exception thrown when attempting to access a relationship that does not exist	Relationship '{relationship ID}' does not exist in entity '{entity ID}'.
UnknownRulebaseException	Caused by attempting to access a rulebase that does not exist	Rulebase '{rulebase ID}' does not exist.
XmlValidationException	Caused by an error when attempting to validate the request or response against the WSDL	

Oracle Determinations Server Services

There are three Services provided by the Oracle Determinations Server:

Server Service

Interview Service

Assess Service

Server Service

The Server Service provides general information about the Determinations Server, such as version information. It also provides a registry of all the available services and endpoints. The following operations are available:

[GetServerInfo](#)

[LoadRulebase](#)

[ListRulebases](#)

Server Service - GetServerInfo Operation

This operation returns the versions of Oracle Determinations Server and Determinations Engine that are currently running as well as the Determination Engine's timezone. This can be used for things such as reporting, diagnostics, and compatibility guarantees.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/10.4/server/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:get-server-info-request/>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/server/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+08:00</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:get-server-info-response>
      <typ:determinations-server-version>10.4.6.1</typ:determinations-server-version>
      <typ:determinations-engine-version>10.4.6.1</typ:determinations-engine-version>
      <typ:interview-engine-version>10.4.6.1</typ:interview-engine-version>
      <typ:determinations-engine-timezone>China Standard Time</typ:determinations-engine-timezone>
    </typ:get-server-info-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Remarks

Both the Determinations Server and Determinations Engine version string will be formatted as:

<major number>.<minor number>.<patch number>.<revision number>

Server Service - ListRulebases Operation

Provides details on the available rulebases and acts as a service registry for all the service endpoints for each rulebase.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/10.4/server/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:list-rulebases-request/>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/server/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+08:00</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:list-rulebases-response>
      <typ:rulebase name="SuperSimple">
        <typ:available-languages>
          <typ:language>fr_FR</typ:language>
          <typ:language>en_GB</typ:language>
          <typ:language>en_US</typ:language>
        </typ:available-languages>
        <typ:build-time>2012-07-05T04:50:35Z</typ:build-time>
        <typ:policy-modeling-version>10.4.1.60</typ:policy-modeling-version>
        <typ:available-services>
          <typ:service name="odsAssessServiceSpecific">http://localhost:8080/determinations-server-
1044/assess/soap/specific/SuperSimple?wsdl</typ:service>
          <typ:service name="odsAssessServiceGeneric">http://localhost:8080/determinations-server-
1044/assess/soap/generic/SuperSimple?wsdl</typ:service>
          <typ:service name="odsAssessServiceSpecific104">http://localhost:8080/determinations-server-
1044/assess/soap/specific/10.4/SuperSimple?wsdl</typ:service>
          <typ:service name="odsAssessServiceGeneric104">http://localhost:8080/determinations-server-
```

```
1044/assess/soap/generic/10.4/SuperSimple?wsdl</typ:service>
  <typ:service name="odsAssessServiceSpecific103">http://localhost:8080/determinations-server-
1044/assess/soap/specific/10.3/SuperSimple?wsdl</typ:service>
  <typ:service name="odsAssessServiceGeneric103">http://localhost:8080/determinations-server-
1044/assess/soap/generic/10.3/SuperSimple?wsdl</typ:service>
  <typ:service name="odsAssessServiceSpecific102">http://localhost:8080/determinations-server-
1044/assess/soap/specific/10.2/SuperSimple?wsdl</typ:service>
  <typ:service name="odsAssessServiceGeneric102">http://localhost:8080/determinations-server-
1044/assess/soap/generic/10.2/SuperSimple?wsdl</typ:service>
  <typ:service name="odsAssessServiceSpecific10">http://localhost:8080/determinations-server-
1044/assess/soap/specific/10.0/SuperSimple?wsdl</typ:service>
  <typ:service name="odsAssessServiceGeneric10">http://localhost:8080/determinations-server-
1044/assess/soap/generic/10.0/SuperSimple?wsdl</typ:service>
  <typ:service name="odsInterviewService">http://localhost:8080/determinations-server-1044/in-
terview/soap/SuperSimple?wsdl</typ:service>
  <typ:service name="odsInterviewService104">http://localhost:8080/determinations-server-
1044/interview/soap/10.4/SuperSimple?wsdl</typ:service>
  <typ:service name="odsInterviewService102">http://localhost:8080/determinations-server-1044/in-
terview/soap/10.2/SuperSimple?wsdl</typ:service>
</typ:available-services>
</typ:rulebase>
<typ:rulebase name="SimpleBenefits">
  <typ:available-languages>
  <typ:language>en_US</typ:language>
</typ:available-languages>
<typ:build-time>2013-07-17T01:11:14Z</typ:build-time>
<typ:policy-modeling-version>10.4.6.1</typ:policy-modeling-version>
<typ:available-services>
  <typ:service name="odsAssessServiceSpecific">http://localhost:8080/determinations-server-
1044/assess/soap/specific/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsAssessServiceGeneric">http://localhost:8080/determinations-server-
1044/assess/soap/generic/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsAssessServiceSpecific104">http://localhost:8080/determinations-server-
1044/assess/soap/specific/10.4/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsAssessServiceGeneric104">http://localhost:8080/determinations-server-
1044/assess/soap/generic/10.4/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsAssessServiceSpecific103">http://localhost:8080/determinations-server-
1044/assess/soap/specific/10.3/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsAssessServiceGeneric103">http://localhost:8080/determinations-server-
1044/assess/soap/generic/10.3/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsAssessServiceSpecific102">http://localhost:8080/determinations-server-
1044/assess/soap/specific/10.2/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsAssessServiceGeneric102">http://localhost:8080/determinations-server-
1044/assess/soap/generic/10.2/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsAssessServiceSpecific10">http://localhost:8080/determinations-server-
```

```
1044/assess/soap/specific/10.0/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsAssessServiceGeneric10">http://localhost:8080/determinations-server-
1044/assess/soap/generic/10.0/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsInterviewService">http://localhost:8080/determinations-server-1044/in-
terview/soap/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsInterviewService104">http://localhost:8080/determinations-server-
1044/interview/soap/10.4/SimpleBenefits?wsdl</typ:service>
  <typ:service name="odsInterviewService102">http://localhost:8080/determinations-server-1044/in-
terview/soap/10.2/SimpleBenefits?wsdl</typ:service>
  </typ:available-services>
</typ:rulebase>
</typ:list-rulebases-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Server Service - LoadRulebase Operation

Loads/Reloads a specified rulebase. The exact behavior of this operation is dependent on the underlying [Rulebase Resolver plugin](#) used by the Determinations Server, or more specifically the behavior of that plugin's **loadRulebase** method. For the default rulebase services provided by the Interview Engine, the behavior is:

- **FileRulebaseService:** (that is, the `load.rulebase.from.classpath` property is false), if no rulebase with that identifier has already been loaded, then it will attempt to load the rulebase from the specified rulebase directory. If the rulebase has already been loaded, it will forcibly reload that rulebase.
- **ClassLoaderRulebaseService:** (that is, the `load.rulebase.from.classpath` property is true). If no rulebase with that identifier has already been loaded, then it will attempt to load the rulebase from the specified classpath. If the rulebase has already been loaded, a call to **LoadRulebase** will do nothing since resources are static on the classpath when the application starts.

Note: because the classpath is static at startup it is not possible to add/replace rulebases after the application has been started and then use this method to load them.

Rulebase:

Located at `examples\rulebases\compiled\SimpleBenefits.zip` in the Oracle Policy Automation Runtime package.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/10.4/server/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:load-rulebase-request>
      <typ:rulebase>SimpleBenefits</typ:rulebase>
    </typ:load-rulebase-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/server/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+08:00</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
```

```
<SOAP-ENV:Body>  
  <typ:load-rulebase-response>  
    <typ:rulebase-loaded>SimpleBenefits</typ:rulebase-loaded>  
  </typ:load-rulebase-response>  
</SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

Possible errors

If there is an error reloading the rulebase, a SOAP Error with message indicating the type of error will be returned.

Interview Service

The Interview Service is a stateful web service that is used to conduct interviews using a rulebase. It can be used as an alternative to Oracle Web Determinations for developing a custom interview application. The following links take you to descriptions of the operations that are available in the Interview service:

[Open Session](#)

[Close Session](#)

[Investigate](#)

[List Goals](#)

[List Screens](#)

[Get Screen](#)

[Get Decision Report](#)

[Get Document](#)

[Get User Set Data](#)

[Save Case](#)

[Load Case](#)

[List Cases](#)

[Set Screen](#)

For an example of using the Interview Service operations to perform an investigation, see the topic, [Example: Investigation using the Determinations Server Interview Service](#).

For notes and tips on creating an Interview Service client in .NET, see the topic, [Example: Implement clients for the Interview Service](#).

Interview Service - Open Session Operation

Opens an interview service session.

Inputs

- Data with which to initialize the session. This data is represented in the same format as the generic assess service.
- Language - supplied inside soap header using *i18n:locale* - the language of the session. If subsequent requests using this session supply a locale in the soap header, it must match the one supplied here. Otherwise an error will be thrown. If no language is supplied, the session will be given the default language of the rulebase.

Outputs

- Success flag indicating whether or not the operation was successful.
- Any errors raised while initializing the session.
- Any warnings raised while initializing the session.
- ID for the new interview service session (only returned if operation was successful).

Note: if an attempt is made to set any inferred entity instances in the Open Session request, an error will result.

Rulebase:

Can be found at `examples\rulebases\compiled\InterviewServiceTest.zip` in the Oracle Policy Automation Runtime package.

Requests

Example Request - with no initialization data:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:open-session-request/>
  </soapenv:Body>
</soapenv:Envelope>
```

Example Request - with initialization data supplied:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:open-session-request>
      <typ:global-instance>
        <typ:attribute id="person_salary" type="currency">
```

```

    <typ:number-val>500.0</typ:number-val>
  </typ:attribute>
  <typ:entity id="school">
    <typ:instance id="school1">
      <typ:attribute id="school_num_students" type="number">
        <typ:number-val>4000.0</typ:number-val>
      </typ:attribute>
      <typ:attribute id="school_type" type="text">
        <typ:text-val>PRIMARY</typ:text-val>
      </typ:attribute>
      <typ:relationship id="schoolsstudents" state="known">
        <typ:target instance-id="child1"/>
      </typ:relationship>
    </typ:instance>
  </typ:entity>
  <typ:entity id="child">
    <typ:instance id="child1">
      <typ:attribute id="child_dob" type="date">
        <typ:date-val>1997-01-01</typ:date-val>
      </typ:attribute>
      <typ:relationship id="childsschool" state="known">
        <typ:target instance-id="school1"/>
      </typ:relationship>
    </typ:instance>
  </typ:entity>
</typ:global-instance>
</typ:open-session-request>
</soapenv:Body>
</soapenv:Envelope>

```

Example Request - with language supplied:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
    </i18n:international>
  </soapenv:Header>
  <soapenv:Body>
    <typ:open-session-request/>
  </soapenv:Body>
</soapenv:Envelope>

```

Responses

Example Response - with no initialization data:

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+08:00</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:open-session-response>
      <typ:success>true</typ:success>
      <typ:interview-session-id>ebab8833-0b4a-4d69-b681-4d5d9f0c7475</typ:interview-session-id>
    </typ:open-session-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Example Response - with initialization data supplied:

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+08:00</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:open-session-response>
      <typ:success>true</typ:success>
      <typ:interview-session-id>d95d6976-b74a-4b68-844e-04caad92d7d6</typ:interview-session-id>
    </typ:open-session-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Example Response - with language supplied:

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+08:00</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>

```

```
<SOAP-ENV:Body>  
  <typ:open-session-response>  
    <typ:success>true</typ:success>  
    <typ:interview-session-id>7b7ab88b-2e44-4b2c-95c8-3282d0fa82df</typ:interview-session-id>  
  </typ:open-session-response>  
</SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

Interview Service - Close Session Operation

Closes an interview service session.

Inputs

Interview service session ID.

Outputs

Return message showing that the session was successfully closed.

Rulebase:

Can be found at `examples\rulebases\compiled\InterviewServiceTest.zip` in the Oracle Policy Automation Runtime package.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:close-session-request>
      <typ:interview-session-id>7b7ab88b-2e44-4b2c-95c8-3282d0fa82df</typ:interview-session-id>
    </typ:close-session-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+08:00</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:close-session-response>
      <typ:interview-session-id>7b7ab88b-2e44-4b2c-95c8-3282d0fa82df</typ:interview-session-id>
    </typ:close-session-response>
  </SOAP-ENV:Body>
```

</SOAP-ENV:Envelope>

Interview Service - Investigate Operation

Used to investigate a specified goal (or flow).

Question screen data that has been updated by the user can be passed in with the request. The interview session will be updated using this data.

The response contains the next question screen in the investigation, or the summary screen if the investigation is complete.

Inputs

- Interview service Session ID.
- Goal state - identifies the goal that is being investigated and contains any necessary state information; see "Goal State" below.
- Investigate the flow - the goal state also contains information regarding the current position in the flow.
- Include commentary (optional) - include any available commentary for controls on the returned screen.
- Question screen data with values added for controls (optional).

Outputs

- Interview service session ID.
- Updated goal state - see *Goal State* below.
- Progress through the investigation (optional - returned only if this information is available for the goal being investigated.).
- Next screen - If no errors or warnings were raised when processing the submitted screen, the next screen in the investigation will be returned. If errors or warnings were raised, then the submitted screen will be returned, with the details of any errors or warnings added.

Errors and warnings

As mentioned above, if errors or warnings were raised when processing the submitted screen, the submitted screen will be returned, with the details of any errors or warnings added. If an error or warning is associated with a specific control, it will be listed with that control. Otherwise, it will be listed at the top of the screen definition.

Warnings listed for a specific control appear in "invalid-value-change" tags. These special warnings indicate that the value submitted for the control was discarded for reasons described in the supplied message.

If errors are raised, the screen data must be modified to resolve the errors then re-submitted in order to continue the investigation. If warnings are raised, it is not mandatory to modify the screen data unless this is desired. The investigation can be continued by resubmitting the unmodified screen data.

Goal state

The goal state passed to the investigate operation identifies the goal that is being investigated. When a flow is being investigated, the goal state also contains information about the current position in the flow.

When beginning a new investigation (that is, making the first request to the investigation operation), the initial goal state will be the goal id for the desired goal, as returned by the List Goals operation.

The response to an investigate request includes the updated goal state. This updated goal state needs to be extracted from the response and used when making the next request.

Request with no errors or warnings

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ-
="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:investigate-request>
      <typ:interview-session-id>79878fa7-3038-43e5-a270-71a866442b45</-
typ:interview-session-id>
      <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
      <typ:screen id="qs$s2@Interviews_Screens_xint$global$global" name-
e="s2@Interviews_Screens_xint" title="Input person's details" context-entity-id=-
="global" context-instance-id="global" type="question" is-automatic="false">
        <typ:currency-control is-visible="true" caption="What is the person's
annual salary?" caption-style="" is-html="false" css-class="" css-style="" attrib-
ute-id="person_salary" is-mandatory="true" is-read-only="false" selection-
style="">
          <typ:current-value>
            <typ:number-val>50000</typ:number-val>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:currency-control>
        <typ:text-control is-visible="true" caption="What is the person's nick-
name?" caption-style="" is-html="false" css-class="" css-style="" attribute-id="pe-
erson_nickname" is-mandatory="true" is-read-only="false" selection-style="">
          <typ:current-value>
            <typ:text-val>JohnB</typ:text-val>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:text-control>
      </typ:screen>
    </typ:investigate-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response with no errors or warnings

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
```

```

typ-
="h-
http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:investigate-response>
      <typ:interview-session-id>79878fa7-3038-43e5-a270-71a866442b45</-
typ:interview-session-id>
      <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
      <typ:progress>0.2857142857142857</typ:progress>
      <typ:screen id="qs$s3@Interviews_Screens_xint$global$global" name=
e="s3@Interviews_Screens_xint" title="Collect the children" context-entity-id=-
="global" context-instance-id="global" type="question" is-automatic="false">
        <typ:containment-relationship-control is-visible="true" caption-
style="" is-html="false" css-class="" css-style="" relationship-id="children"
source-entity-id="global" source-instance-id="global" target-entity-id="child"
relationship-type="OneToMany" display-style="Portrait">
          <typ:entity-instance-control is-visible="true" caption-style="" is-
html="false" css-class="" css-style="" entity-id="child" instance-id="--BLANK--">
            <typ:date-control is-visible="true" caption="What is the child's
date of birth?" caption-style="" is-html="false" css-class="" css-style="" attrib-
ute-id="child_dob" is-mandatory="true" is-read-only="false" selection-style-
e="default" input-style="d">
              <typ:current-value>
                <typ:unknown-val/>
              </typ:current-value>
              <typ:default-value>
                <typ:unknown-val/>
              </typ:default-value>
              <typ:properties>
                <typ:property key="CustomControlProperty"/>
              </typ:properties>
            </typ:date-control>
            <typ:properties>
              <typ:property key="CustomControlProperty"/>
            </typ:properties>
          </typ:entity-instance-control>
          <typ:properties>
            <typ:property key="CustomControlProperty"/>
          </typ:properties>
        </typ:containment-relationship-control>
        <typ:properties>
          <typ:property key="CustomScreenProperty"/>
        </typ:properties>
      </typ:screen>
    </typ:investigate-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

```

        </typ:properties>
    </typ:screen>
</typ:investigate-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Request with errors and warnings for specific controls

Example:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/ErrorTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:investigate-request>
      <typ:interview-session-id>636272c7-053f-4c0c-856e-53e9914969d3</-
typ:interview-session-id>
      <typ:goal-state>Attribute~parent_happy~global~global</typ:goal-state>
      <typ:screen id="qs$parent_info$global$global" name="parent_info" title-
e="Parent Info" context-entity-id="global" context-instance-id="global" type-
e="question" is-automatic="false">
        <typ:currency-control is-visible="true" caption="What is the parent's
salary?" caption-style="" is-html="false" css-class="" css-style="" attribute-id=-
="parent_salary" is-mandatory="true" is-read-only="false" selection-style="">
          <typ:current-value>
            <typ:number-val>50000</typ:number-val>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:currency-control>
        <typ:boolean-control is-visible="true" caption="Does the parent have a
good paying job?" caption-style="" is-html="false" css-class="" css-style=""
attribute-id="parent_good_job" is-mandatory="true" is-read-only="false" selection-
style="">
          <typ:current-value>
            <typ:boolean-val>true</typ:boolean-val>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:boolean-control>
      </typ:screen>
    </typ:investigate-request>
  </soapenv:Body>
</soapenv:Envelope>

```

Response with errors and warnings for specific controls

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/ErrorTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:investigate-response>
      <typ:interview-session-id>636272c7-053f-4c0c-856e-53e9914969d3</-
typ:interview-session-id>
      <typ:goal-state>Attribute~parent_happy~global~global</typ:goal-state>
      <typ:progress>0.3333333333333333</typ:progress>
      <typ:screen id="qs$parent_info$global$global" name="parent_info" title=
e="Parent Info" context-entity-id="global" context-instance-id="global" type=
e="question" is-automatic="false">
        <typ:currency-control is-visible="true" caption="What is the parent's
salary?" caption-style="" is-html="false" css-class="" css-style="" attribute-id=
="parent_salary" is-mandatory="true" is-read-only="false" selection-style=
e="default">
          <typ:current-value>
            <typ:number-val>50000.0</typ:number-val>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:currency-control>
        <typ:boolean-control is-visible="false" caption="Does the parent have
a good paying job?" caption-style="" is-html="false" css-class="" css-style=""
attribute-id="parent_good_job" is-mandatory="true" is-read-only="false" selection-
style="default">
          <typ:invalid-value-change>There was an attempt to change the value
of a control that is not visible. The submitted value for this control has been
discarded.</typ:invalid-value-change>
          <typ:current-value>
            <typ:unknown-val/>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:boolean-control>
      </typ:screen>
    </typ:investigate-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
</typ:investigate-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Request with errors and warnings listed at top of screen definition

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ="http://oracle.com/determinations/server/interview/10.4/ErrorTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:investigate-request>
      <typ:interview-session-id>636272c7-053f-4c0c-856e-53e9914969d3</typ:interview-session-id>
      <typ:goal-state>Attribute~parent_happy~global~global</typ:goal-state>
      <typ:screen id="qs$parent_info$global$global" name="parent_info" title="Parent Info" context-entity-id="global" context-instance-id="global" type="question" is-automatic="false">
        <typ:currency-control is-visible="true" caption="What is the parent's salary?" caption-style="" is-html="false" css-class="" css-style="" attribute-id="parent_salary" is-mandatory="true" is-read-only="false" selection-style="">
          <typ:current-value>
            <typ:number-val>-50000</typ:number-val>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:currency-control>
        <typ:boolean-control is-visible="true" caption="Does the parent have a good paying job?" caption-style="" is-html="false" css-class="" css-style="" attribute-id="parent_good_job" is-mandatory="true" is-read-only="false" selection-style="">
          <typ:current-value>
            <typ:unknown-val/>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:boolean-control>
      </typ:screen>
    </typ:investigate-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response with errors and warnings listed at top of screen definition

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/ErrorTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:investigate-response>
      <typ:interview-session-id>636272c7-053f-4c0c-856e-53e9914969d3</-
typ:interview-session-id>
      <typ:goal-state>Attribute~parent_happy~global~global</typ:goal-state>
      <typ:progress>0.3333333333333333</typ:progress>
      <typ:screen id="qs$parent_info$global$global" name="parent_info" title=
e="Parent Info" context-entity-id="global" context-instance-id="global" type=
e="question" is-automatic="false">
        <typ:error-list>
          <typ:error>You have entered a negative salary value.</typ:error>
        </typ:error-list>
        <typ:currency-control is-visible="true" caption="What is the parent's
salary?" caption-style="" is-html="false" css-class="" css-style="" attribute-id=-
="parent_salary" is-mandatory="true" is-read-only="false" selection-style=
e="default">
          <typ:current-value>
            <typ:number-val>-50000.0</typ:number-val>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:currency-control>
        <typ:boolean-control is-visible="false" caption="Does the parent have
a good paying job?" caption-style="" is-html="false" css-class="" css-style=""
attribute-id="parent_good_job" is-mandatory="true" is-read-only="false" selection-
style="default">
          <typ:current-value>
            <typ:unknown-val/>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:boolean-control>
      </typ:screen>
    </typ:investigate-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Interview Service - List Goals Operation

Lists all of the top-level goals (including flows) for the interview session.

Inputs

- Interview service Session ID.

Outputs

- Interview service Session ID.
- List of top-level goals organized by entity and entity instance. For each goal the following information will be listed:
 - goal-id: the goal's ID.
 - goal-text: the goal's text.
 - has-decision-report: true if a decision report can be obtained for the goal. This will be false if the goal is a flow.
 - attribute-id: the ID of the attribute associated with the goal. This will not be listed if the goal is a flow.

Rulebase:

Can be found at `examples\rulebases\compiled\InterviewServiceTest.zip` in the Oracle Policy Automation Runtime package.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:list-goals-request>
      <typ:interview-session-id>93c74cec-9530-497a-a123-e44c5af1d479</typ:interview-session-id>
    </typ:list-goals-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+08:00</i18n:tz>
```

```
</i18n:international>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <typ:list-goals-response>
    <typ:interview-session-id>93c74cec-9530-497a-a123-e44c5af1d479</typ:interview-session-id>
    <typ:entity id="global">
      <typ:instance id="global">
        <typ:goal>
          <typ:goal-id>Attribute~person_eligible~global~global</typ:goal-id>
          <typ:goal-text>Is the person eligible for education expenses assistance?</typ:goal-text>
          <typ:has-decision-report>true</typ:has-decision-report>
          <typ:attribute-id>person_eligible</typ:attribute-id>
        </typ:goal>
      </typ:instance>
    </typ:entity>
  </typ:list-goals-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Interview Service - List Screens Operation

Lists all of the available screens in an interview session. Filtering options can be applied to obtain all screens associated with a specified entity instance, or all screens on which a specified attribute or relationship can be set for a particular entity instance.

Inputs

- Interview service Session ID

All other inputs are optional:

- Entity ID (optional)
- Instance ID (optional)
- Attribute ID (optional)
- Relationship ID (optional)

To obtain all screens associated with a specified entity instance, supply Entity ID and Instance ID.

To obtain all screens on which a specified attribute can be set for an entity instance, supply Entity ID, Instance ID and Attribute ID.

To obtain all screens on which a specified relationship can be set for an entity instance, supply Entity ID, Instance ID and Relationship ID.

Outputs

A list of screens organized by entity and entity instance. The list of screens will be filtered according to the available filtering options described in the Inputs section.

Rulebase:

Can be found at [examples\rulebases\compiled\InterviewServiceTest.zip](#) in the Oracle Policy Automation Runtime package.

Requests

Example Request - no filtering:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ-  
typ-  
="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">  
  <soapenv:Header/>  
  <soapenv:Body>  
    <typ:list-screens-request>  
      <typ:interview-session-id>b8bbf3ea-a6b7-49da-a16f-32747684d443</-  
typ:interview-session-id>  
    </typ:list-screens-request>  
  </soapenv:Body>  
</soapenv:Envelope>
```

Example Request - with filter for entity instance:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ-
="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:list-screens-request>
      <typ:interview-session-id>b8bbf3ea-a6b7-49da-a16f-32747684d443</-
typ:interview-session-id>
      <typ:entity-id>global</typ:entity-id>
      <typ:instance-id>global</typ:instance-id>
    </typ:list-screens-request>
  </soapenv:Body>
</soapenv:Envelope>

```

Example Request - with filter for entity instance and attribute:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:list-screens-request>
      <typ:interview-session-id>93c74cec-9530-497a-a123-e44c5af1d479</typ:interview-session-id>
      <typ:entity-id>global</typ:entity-id>
      <typ:instance-id>global</typ:instance-id>
      <typ:attribute-id>person_salary</typ:attribute-id>
    </typ:list-screens-request>
  </soapenv:Body>
</soapenv:Envelope>

```

Example Request - with filter for entity instance and relationship:

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:list-screens-request>
      <typ:interview-session-id>93c74cec-9530-497a-a123-e44c5af1d479</typ:interview-session-id>
      <typ:entity-id>global</typ:entity-id>
      <typ:instance-id>global</typ:instance-id>
      <typ:relationship-id>children</typ:relationship-id>
    </typ:list-screens-request>
  </soapenv:Body>
</soapenv:Envelope>

```

Responses

Example Response - no filtering:

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"

```

```

xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ-
="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:list-screens-response>
      <typ:interview-session-id>b8bbf3ea-a6b7-49da-a16f-32747684d443</-
typ:interview-session-id>
      <typ:entity id="global">
        <typ:instance id="global">
          <typ:screen id="qs$s2@Interviews_Screens_xint$global$global" name-
e="s2@Interviews_Screens_xint" title="Input person's details" context-entity-id=-
="global" context-instance-id="global" type="question" is-automatic="false"/>
          <typ:screen id="qs$s3@Interviews_Screens_xint$global$global" name-
e="s3@Interviews_Screens_xint" title="Collect the children" context-entity-id=-
="global" context-instance-id="global" type="question" is-automatic="false"/>
          <typ:screen id="qs$s4@Interviews_Screens_xint$global$global" name-
e="s4@Interviews_Screens_xint" title="Collect the schools" context-entity-id=-
="global" context-instance-id="global" type="question" is-automatic="false"/>
          <typ:screen id="qs$summary$global$global" name="summary" title-
e="Assessment Summary" context-entity-id="global" context-instance-id="global"
type="summary" is-automatic="false"/>
          <typ:screen id="dr$DefaultScreenOrder$global$global" name="Default
Data Review" title="Data Review" context-entity-id="global" context-instance-id=-
="global" type="data-review" is-automatic="false"/>
        </typ:instance>
      </typ:entity>
    </typ:list-screens-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Example Response - with filter for entity instance:

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ-
="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>

```

```

</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <typ:list-screens-response>
    <typ:interview-session-id>b8bbf3ea-a6b7-49da-a16f-32747684d443</-
typ:interview-session-id>
    <typ:entity id="global">
      <typ:instance id="global">
        <typ:screen id="qs$s2@Interviews_Screens_xint$global$global" name-
e="s2@Interviews_Screens_xint" title="Input person's details" context-entity-id=-
="global" context-instance-id="global" type="question" is-automatic="false"/>
        <typ:screen id="qs$s3@Interviews_Screens_xint$global$global" name-
e="s3@Interviews_Screens_xint" title="Collect the children" context-entity-id=-
="global" context-instance-id="global" type="question" is-automatic="false"/>
        <typ:screen id="qs$s4@Interviews_Screens_xint$global$global" name-
e="s4@Interviews_Screens_xint" title="Collect the schools" context-entity-id=-
="global" context-instance-id="global" type="question" is-automatic="false"/>
        <typ:screen id="qs$summary$global$global" name="summary" title-
e="Assessment Summary" context-entity-id="global" context-instance-id="global"
type="summary" is-automatic="false"/>
        <typ:screen id="dr$DefaultScreenOrder$global$global" name="Default
Data Review" title="Data Review" context-entity-id="global" context-instance-id=-
="global" type="data-review" is-automatic="false"/>
      </typ:instance>
    </typ:entity>
  </typ:list-screens-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Example Response - with filter for entity instance and attribute:

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+08:00</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:list-screens-response>
      <typ:interview-session-id>93c74cec-9530-497a-a123-e44c5af1d479</typ:interview-session-id>
      <typ:entity id="global">
        <typ:instance id="global">
          <typ:screen id="qs$s2@Interviews_Screens_xint$global$global" name="s2@Interviews_Screens_xint" title-
e="Input person's details" context-entity-id="global" context-instance-id="global" type="question" is-auto-
matic="false"/>

```

```
</typ:instance>
</typ:entity>
</typ:list-screens-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example Response - with filter for entity instance and relationship:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+08:00</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:list-screens-response>
      <typ:interview-session-id>93c74cec-9530-497a-a123-e44c5af1d479</typ:interview-session-id>
      <typ:entity id="global">
        <typ:instance id="global">
          <typ:screen id="qs$s3@Interviews_Screens_xint$global$global" name="s3@Interviews_Screens_xint" title="Collect the children" context-entity-id="global" context-instance-id="global" type="question" is-automatic="false"/>
        </typ:instance>
      </typ:entity>
    </typ:list-screens-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Interview Service - Get Screen Operation

Gets the details of a screen outside of the context of an investigation of a particular goal. One common use for this operation would be to get a data review screen.

Inputs

- Interview service session ID.
- Screen ID.
- include commentary (optional) - include any available commentary for controls on the returned screen.

Outputs

- Interview service session ID.
- Screen data for requested screen.

Rulebase:

Can be found at [examples\rulebases\compiled\InterviewServiceTest.zip](#) in the Oracle Policy Automation Runtime package.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ-
="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:get-screen-request>
      <typ:interview-session-id>de5ee582-4ae6-43f6-86fc-0f299ff37178</-
typ:interview-session-id>
      <typ:screen-id>dr$DefaultScreenOrder$global$global</typ:screen-id>
    </typ:get-screen-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ-
="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
```

```

        <i18n:locale>en_US</i18n:locale>
        <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
    <typ:get-screen-response>
        <typ:interview-session-id>de5ee582-4ae6-43f6-86fc-0f299ff37178</-
typ:interview-session-id>
        <typ:screen id="dr$DefaultScreenOrder$global$global" name="Default Data
Review" title="Data Review" context-entity-id="global" context-instance-id=-
="global" type="data-review" is-automatic="false">
            <typ:data-review-control is-visible="true" cap-
tion="DefaultScreenOrder" caption-style="" is-html="false" css-class="" css-
style="">
                <typ:screen id="qs$s2@Interviews_Screens_xint$global$global" name-
e="s2@Interviews_Screens_xint" title="Input person's details" context-entity-id=-
="global" context-instance-id="global" type="question" is-automatic="false">
                    <typ:currency-control is-visible="true" caption="What is the per-
son's annual salary?" caption-style="" is-html="false" css-class="" css-style=""
attribute-id="person_salary" is-mandatory="true" is-read-only="false" selection-
style="default">
                        <typ:current-value>
                            <typ:number-val>50000.0</typ:number-val>
                        </typ:current-value>
                        <typ:default-value>
                            <typ:unknown-val/>
                        </typ:default-value>
                        <typ:properties>
                            <typ:property key="CustomControlProperty"/>
                        </typ:properties>
                    </typ:currency-control>
                    <typ:text-control is-visible="true" caption="What is the per-
son's nickname?" caption-style="" is-html="false" css-class="" css-style="" attrib-
ute-id="person_nickname" is-mandatory="true" is-read-only="false" selection-
style="default" line-count="1">
                        <typ:current-value>
                            <typ:text-val>JohnB</typ:text-val>
                        </typ:current-value>
                        <typ:default-value>
                            <typ:unknown-val/>
                        </typ:default-value>
                        <typ:properties>
                            <typ:property key="CustomControlProperty"/>
                        </typ:properties>
                    </typ:text-control>
                <typ:properties>
                    <typ:property key="CustomScreenProperty"/>
                </typ:properties>
            </typ:screen>
        </typ:data-review-control>
    </typ:screen>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

```
        </typ:screen>
      </typ:data-review-control>
    </typ:screen>
  </typ:get-screen-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Interview Service - Get Decision Report Operation

Gets the decision report for a top-level goal.

Inputs

- Interview service session ID.
- Goal ID.
- show-silent - flag indicating whether or not decision nodes should be shown for rules marked with the silent operator.
- show-invisible - flag indicating whether or not decision nodes should be shown for rules marked with the invisible operator.

Outputs

The decision report screen for the specified goal.

Rulebase:

Can be found at `examples\rulebases\compiled\InterviewServiceTest.zip` in the Oracle Policy Automation Runtime package.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ-  
typ-  
="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">  
  <soapenv:Header/>  
  <soapenv:Body>  
    <typ:get-decision-report-request>  
      <typ:interview-session-id>de5ee582-4ae6-43f6-86fc-0f299ff37178</-  
typ:interview-session-id>  
      <typ:goal-id>Attribute~person_eligible~global~global</typ:goal-id>  
      <typ:show-silent>>true</typ:show-silent>  
      <typ:show-invisible>>true</typ:show-invisible>  
    </typ:get-decision-report-request>  
  </soapenv:Body>  
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"  
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-  
typ-  
="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">  
  <SOAP-ENV:Header>  
    <i18n:international>
```

```
<i18n:locale>en_US</i18n:locale>
<i18n:tz>GMT+0800</i18n:tz>
</i18n:international>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <typ:get-decision-report-response>
    <typ:interview-session-id>de5ee582-4ae6-43f6-86fc-0f299ff37178</-
typ:interview-session-id>
    <typ:screen id="report$attr$person_eli-
gible$global$global$Relevant$true$true" name="report$attr$person_eli-
gible$global$global$Relevant$true$true" title="The person is eligible for
education expenses assistance." context-entity-id="global" context-instance-id=-
="global" type="decision-report" is-automatic="true">
      <typ:decision-report-control is-visible="true" caption="The person is
eligible for education expenses assistance." caption-style="" is-html="false" css-
class="" css-style="" ignore-silent="true" ignore-invisible="true">
        <typ:attribute-node-control is-visible="true" caption="The person
is eligible for education expenses assistance." caption-style="" is-html="false"
css-class="" css-style="" id="report$attr$person_eli-
gible$global$global$Relevant$true$true0" is-base-level="false" is-user-set="false"
is-known="true" attribute-id="person_eligible" entity-id="global" instance-id=-
="global">
          <typ:attribute-value>
            <typ:boolean-val>true</typ:boolean-val>
          </typ:attribute-value>
          <typ:attribute-node-control is-visible="true" caption="The per-
son's annual salary is $50,000.00." caption-style="" is-html="false" css-class=""
css-style="" id="report$attr$person_eligible$global$global$Relevant$true$true1"
is-base-level="true" is-user-set="true" is-known="true" attribute-id="person_
salary" entity-id="global" instance-id="global" set-on-screen="qs$s2@Interviews_
Screens_xint$global$global">
            <typ:attribute-value>
              <typ:number-val>50000.0</typ:number-val>
            </typ:attribute-value>
            <typ:properties>
              <typ:property key="NewProperty"/>
            </typ:properties>
          </typ:attribute-node-control>
          <typ:relationship-node-control is-visible="true" caption="the
children" caption-style="" is-html="false" css-class="" css-style="" id="r-
report$attr$person_eligible$global$global$Relevant$true$true2" is-base-level-
="true" is-user-set="true" is-known="true" relationship-id="children" source-
entity-id="global" target-entity-id="child" relationship-type="OneToMany" source-
instance-id="global" set-on-screen="qs$s3@Interviews_Screens_xint$global$global">
            <typ:entity-instance-node-control is-visible="true" cap-
tion="child1" caption-style="" is-html="false" css-class="" css-style="" id="r-
report$attr$person_eligible$global$global$Relevant$true$true3" is-base-
level="true" is-user-set="true" is-known="true" entity-id="child" instance-
```

```

id="child1">
    <typ:properties>
        <typ:property key="EntityProperty">foo</typ:property>
    </typ:properties>
</typ:entity-instance-node-control>
<typ:properties>
    <typ:property key-
y="RelationshipProperty">bar</typ:property>
</typ:properties>
</typ:relationship-node-control>
<typ:attribute-node-control is-visible="true" caption="The
child's date of birth is 25/07/96." caption-style="" is-html="false" css-class=""
css-style="" id="report$attr$person_eligible$global$global$Relevant$true$true4"
is-base-level="true" is-user-set="true" is-known="true" attribute-id="child_dob"
entity-id="child" instance-id="child1" set-on-screen="qs$s3@Interviews_Screens_
xint$global$global">
    <typ:attribute-value>
        <typ:date-val>1996-07-25</typ:date-val>
    </typ:attribute-value>
</typ:properties>
    <typ:property key="NewProperty"/>
</typ:properties>
</typ:attribute-node-control>
<typ:relationship-node-control is-visible="true" caption="the
child's school" caption-style="" is-html="false" css-class="" css-style="" id="r-
report$attr$person_eligible$global$global$Relevant$true$true5" is-base-level-
l="true" is-user-set="true" is-known="true" relationship-id="childsschool" source-
entity-id="child" target-entity-id="school" relationship-type="ManyToOne" source-
instance-id="child1" set-on-screen="qs$s5@Interviews_Screens_xint$child$child1">
    <typ:entity-instance-node-control is-visible="true" cap-
tion="school1" caption-style="" is-html="false" css-class="" css-style="" id="r-
report$attr$person_eligible$global$global$Relevant$true$true6" is-base-
level="true" is-user-set="true" is-known="true" entity-id="school" instance-
id="school1">
        <typ:properties>
            <typ:property key="EntityProperty"/>
        </typ:properties>
    </typ:entity-instance-node-control>
</typ:relationship-node-control>
<typ:attribute-node-control is-visible="true" caption="The
school's type is SECONDARY." caption-style="" is-html="false" css-class="" css-
style="" id="report$attr$person_eligible$global$global$Relevant$true$true7" is-
base-level="true" is-user-set="true" is-known="true" attribute-id="school_type"
entity-id="school" instance-id="school1" set-on-screen="qs$s4@Interviews_Screens_
xint$global$global">
    <typ:attribute-value>
        <typ:text-val>SECONDARY</typ:text-val>
    </typ:attribute-value>

```

```
        <typ:properties>
          <typ:property key="NewProperty"/>
        </typ:properties>
      </typ:attribute-node-control>
      <typ:attribute-node-control is-visible="true" caption="The
school's number of students is 1,000." caption-style="" is-html="false" css-
class="" css-style="" id="report$attr$person_eli-
gible$global$global$Relevant$true$true8" is-base-level="true" is-user-set="true"
is-known="true" attribute-id="school_num_students" entity-id="school" instance-
id="school1" set-on-screen="qs$s4@Interviews_Screens_xint$global$global">
        <typ:attribute-value>
          <typ:number-val>1000.0</typ:number-val>
        </typ:attribute-value>
        <typ:properties>
          <typ:property key="NewProperty"/>
        </typ:properties>
      </typ:attribute-node-control>
      <typ:properties>
        <typ:property key="NewProperty"/>
      </typ:properties>
    </typ:attribute-node-control>
  </typ:decision-report-control>
</typ:screen>
</typ:get-decision-report-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Interview Service - Get Document Operation

Generates a document.

Inputs

- Interview service session ID.
- Document ID - this is listed as an attribute of a BI Publisher document control as shown on a summary screen.

Outputs

- Interview service session ID.
- Document ID.
- Mime type of document.
- Document content- this is a base64 encoded string; it needs to be decoded in order for the content of the document to be viewed.

Rulebase:

Can be found at `examples\rulebases\compiled\InterviewServiceTest.zip` in the Oracle Policy Automation Runtime package.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:get-document-request>
      <typ:interview-session-id>932c2e97-73ed-4ca6-bee2-62c654d7e316</typ:interview-session-id>
      <typ:document-id>InterviewServiceTest</typ:document-id>
    </typ:get-document-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
```

```
<i18n:locale>en_US</i18n:locale>
<i18n:tz>GMT+08:00</i18n:tz>
</i18n:international>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <typ:get-document-response>
    <typ:interview-session-id>932c2e97-73ed-4ca6-bee2-62c654d7e316</typ:interview-session-id>
    <typ:document-id>InterviewServiceTest</typ:document-id>
    <typ:mime-type>text/html</typ:mime-type>
    <typ:document-con-
tent>PD94bWwg-
dmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4NCjwhRE9DVFIQRSBodG1sIFBVQkxJQyAiLS8vVzNDLy9EVEQgWEhUTUwgMS
content>
  </typ:get-document-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Interview Service - Get User Set Data Operation

Gets all data set by the user into the specified the interview session. Data is returned in the same format used by the [Assess service](#).

Inputs

- Interview service session ID.

Outputs

- All user set data in the interview session.

Rulebase:

Can be found at `examples\rulebases\compiled\InterviewServiceTest.zip` in the Oracle Policy Automation Runtime package.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:get-user-set-data-request>
      <typ:interview-session-id>932c2e97-73ed-4ca6-bee2-62c654d7e316</typ:interview-session-id>
    </typ:get-user-set-data-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+08:00</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:get-user-set-data-response>
```

```
<typ:interview-session-id>932c2e97-73ed-4ca6-bee2-62c654d7e316</typ:interview-session-id>
<typ:global-instance>
  <typ:attribute id="person_salary" type="currency">
    <typ:number-val>50000.0</typ:number-val>
  </typ:attribute>
  <typ:attribute id="person_nickname" type="text">
    <typ:text-val>JohnB</typ:text-val>
  </typ:attribute>
  <typ:entity id="child">
    <typ:instance id="child1">
      <typ:attribute id="child_dob" type="date">
        <typ:date-val>1996-07-25</typ:date-val>
      </typ:attribute>
      <typ:relationship id="childsschool" state="known">
        <typ:target instance-id="school1"/>
      </typ:relationship>
    </typ:instance>
  </typ:entity>
  <typ:entity id="school">
    <typ:instance id="school1">
      <typ:attribute id="school_type" type="text">
        <typ:text-val>SECONDARY</typ:text-val>
      </typ:attribute>
      <typ:attribute id="school_num_students" type="number">
        <typ:number-val>1000.0</typ:number-val>
      </typ:attribute>
      <typ:relationship id="schoolsstudents" state="known">
        <typ:target instance-id="child1"/>
      </typ:relationship>
    </typ:instance>
  </typ:entity>
</typ:global-instance>
</typ:get-user-set-data-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Interview Service - Save Case Operation

Saves all data input for the current interview engine session using a data adapter added to the interview session. A generic error (soap fault) will be returned if no data adapter has been installed in the interview engine.

Inputs

- Interview service Session ID.
- Case ID under which to save the session data.

Outputs

- Interview service Session ID.
- Case ID under which the session data was saved.
- Flag indicating whether or not saving the session data was successful.
- Any errors raised while saving the data.
- Any warnings raised while saving the data.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:save-data-request>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:case-id>investigation1</typ:case-id>
    </typ:save-data-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n"
  xmlns:typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1000</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
```

```
<typ:save-data-response>
  <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
  <typ:case-id>investigation1</typ:case-id>
  <typ:success>>true</typ:success>
</typ:save-data-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Interview Service - Load Case Operation

Loads session data that was saved using the save data operation into the supplied interview session. Uses a data adapter added to the interview session.

Inputs

- Interview service Session ID.
- Case ID.

Outputs

- Interview service Session ID.
- Case ID under which the session data was saved.
- Flag indicating whether or not loading the session data was successful.
- Any errors raised while loading the data.
- Any warnings raised while loading the data.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:load-data-request>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:case-id>investigation1</typ:case-id>
    </typ:load-data-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n"
  xmlns:typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1000</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
```

```
<typ:load-data-response>  
  <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>  
  <typ:case-id>investigation1</typ:case-id>  
  <typ:success>>true</typ:success>  
</typ:load-data-response>  
</SOAP-ENV:Body>  
</SOAP-ENV:Envelope>
```

Interview Service - List Cases Operation

Lists the ids of cases that have been persisted using the Save Data operation.

Inputs

- Interview service Session ID.

Outputs

- Interview service Session ID.
- List of ids for cases that have been saved.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:list-cases-request>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
    </typ:list-cases-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

Example:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n"
  xmlns:typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1000</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:list-cases-response>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:case-id>case1</typ:case-id>
      <typ:case-id>case2</typ:case-id>
      <typ:case-id>case3</typ:case-id>
    </typ:list-cases-response>
  </SOAP-ENV:Body>
```

</SOAP-ENV:Envelope>

Interview Service - Set Screen Operation

Sets data from the supplied screen into the supplied interview session. This is done outside of the context of an investigation.

Inputs

- Interview service Session ID.
- Question screen data with values added for controls.

Outputs

- Interview service Session ID.
- Flag stating whether setting of screen was successful or not.
- Screen data - if any warnings or errors are raised while submitting the screen data, then the screen data will be returned in the response with any errors or warnings included inline.

Rulebase:

Can be found at `examples\rulebases\compiled\InterviewServiceTest.zip` in the Oracle Policy Automation Runtime package.

Request

Example:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:set-screen-request>
      <typ:interview-session-id>7aea1549-411b-457a-91ae-f0dcb6e01818</typ:interview-session-id>
      <typ:screen id="qs$s3@Interviews_Screens_xint$global$global" name="s3@Interviews_Screens_xint" title=
e="Collect the children" context-entity-id="global" context-instance-id="global" type="question" is-auto-
matic="false">
        <typ:containment-relationship-control is-visible="true" caption-style="" is-html="false" css-class="" css-
style="" relationship-id="children" source-entity-id="global" source-instance-id="global" target-entity-id="child"
relationship-type="OneToMany" display-style="Portrait">
          <typ:entity-instance-control is-visible="true" caption-style="" is-html="false" css-class="" css-style=""
entity-id="child" instance-id="child1">
            <typ:date-control is-visible="true" caption="What is the child's date of birth?" caption-style="" is-htm-
l="false" css-class="" css-style="" attribute-id="child_dob" is-mandatory="true" is-read-only="false" selection-
style="" input-style="d">
              <typ:current-value>
                <typ:date-val>1996-01-01</typ:date-val>
              </typ:current-value>
              <typ:default-value>
```

```

        <typ:unknown-val/>
    </typ:default-value>
</typ:date-control>
</typ:entity-instance-control>
    <typ:entity-instance-control is-visible="true" caption-style="" is-html="false" css-class="" css-style=""
entity-id="child" instance-id="child2">
        <typ:date-control is-visible="true" caption="What is the child's date of birth?" caption-style="" is-html-
l="false" css-class="" css-style="" attribute-id="child_dob" is-mandatory="true" is-read-only="false" selection-
style="" input-style="d">
            <typ:current-value>
                <typ:date-val>1997-01-01</typ:date-val>
            </typ:current-value>
            <typ:default-value>
                <typ:unknown-val/>
            </typ:default-value>
        </typ:date-control>
    </typ:entity-instance-control>
</typ:containment-relationship-control>
</typ:screen>
</typ:set-screen-request>
</soapenv:Body>
</soapenv:Envelope>

```

Response

Example:

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/10.4/InterviewServiceTest/types">
    <SOAP-ENV:Header>
        <i18n:international>
            <i18n:locale>en_US</i18n:locale>
            <i18n:tz>GMT+08:00</i18n:tz>
        </i18n:international>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <typ:set-screen-response>
            <typ:interview-session-id>7aea1549-411b-457a-91ae-f0dcb6e01818</typ:interview-session-id>
            <typ:success>true</typ:success>
        </typ:set-screen-response>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Assess Service

Description

The Assess service provides a mechanism for using a rulebase through Oracle Determinations Server. This service enables you to list the goal attributes of a rulebase as well as perform an assessment.

An assessment allows you to:

- Send information to the rulebase and then use its rules to determine the value of one or multiple goals for any Global Goal or a goal within an Entity.
- Get Decision Reports to determine why a Goal is a particular value, or why its value is unknown.
- Get a Screen definition for the next question needed to determine a goal.

Operation

The Assess service exposes the following two operations to satisfy its purpose:-

- ListGoals operation – list the goals attributes of a rulebase
- Assess operation – perform an assessment

Refer to the following topics for more detailed information:

[ListGoals operation](#)

[Assess operation Request and Response elements](#)

[Example: Assess Request xml](#)

[Example: Implement Clients for the Assess service](#)

[Example: Implement Clients for the Interview service](#)

[Explicit Data Set - specific data model](#)

[Explicit Data Set - generic data model](#)

[Rulebase specific interface](#)

[Rulebase generic interface](#)

[Assess API events](#)

[Get an answer from the Determinations Server Assess Request](#)

[Process time-varying data in a web service call](#)

[Find out what data is required by the web service to get an answer](#)

Assess Operation Request and Response elements

Node:assess-request

The root container node for an assess request. The assess request contains an optional config node and a mandatory global-instance node. The global-instance element may be generic or specific.

Node:config

Contains configuration settings for the rulebase session. There are five settings.

Example:

```
<typ:config>
  <typ:show-properties>true</typ:show-properties>
  <typ:show-events>true</typ:show-events>
  <typ:show-silent>true</typ:show-silent>
  <typ:show-invisible>true</typ:show-invisible>
  <typ:resolve-indecision-relationships>true</typ:resolve-indecision-
    relationships>
</typ:config>
```

show-properties

This can be set to true or false. If it is true, custom properties of elements and attributes will be shown in decision reports, session-data and screens.

show-events

This can be set to true or false. If it is true, any events will be returned in an assess response.

show-silent

This can be set to true or false. If it is true, any silent operators attached to rule premises will be ignored for decision reports, resulting in the rule branch underneath the silent operator being included in the decision report. If it is false or not specified, the decision report will be shown with the silent operator having its usual effect of hiding all proving attributes.

show-invisible

This can be set to true or false. If it is true, any invisible operators attached to rule premises will be ignored for decision reports, resulting in the premise being included in the decision report. If it is false or not specified, the decision report will be shown with the invisible operator having its usual effect of hiding the attribute to which it is attached.

resolve-indecision-relationships

This can be set to true or false. If it is true, for all decision reports for an Attribute which is Unknown (an indecision report) the indecision report will look below an unknown relationship, and report on attributes and relationships.

Normally a decision report will stop at an Unknown Relationship.

Node:global-instance

The global-instance element represents the root node of the session data used to make an assessment and appears in both the Generic and Specific interfaces. In the generic format, the global instance has zero or more 'entity' nodes. However, in a specific format, the global instance has zero or more 'entity-list' nodes. The 'entity' and 'entity-list' node serves as a container for child instances of a common entity type.

In both the Generic and Specific use, the global instance element is named 'global-instance'.

Example

```
<typ:global-instance>  
...  
</typ:global-instance>
```

Node:list-entity

A list of entities of a common type. This node is only present in the specific format. An list-entity contains instances of a particular entity (see Node:instance below). Each entity has a list element. For example a person entity would have a "list-person" element.

Example (Specific):

In the same example in the specific format there is a specific list-person element, which contains person instances.

```
<typ:list-person>  
  <typ:person id="sue"/>  
  <typ:person id="bobjr"/>  
  <typ:person id="lilSue"/>  
</typ:list-person>
```

Node:entity

A container for entity instances of a common type. This node is only present in the generic format.

```
<typ:entity id="person">  
  <typ:person id="sue"/>  
  <typ:person id="bobjr"/>  
  <typ:person id="lilSue"/>  
</typ:entity>
```

attribute:id

The name of the entity.

Node:instance

An instance of an entity. An entity instance can have attributes (Node:attribute), relationships (Node:relationships) and child entities (Node:entity).

In the specific format an entity will be specifically named. For example, a entity toy will be an element <typ:toy id="plane">.

attribute:id

The entity name identifying the unique identifier (name) of this type of entity.

Example:

The entity set "toys" below has a single instance with a label 'plane'. Entity instance has an attribute 'wings'.

```
<typ:entity id="toy">
  <typ:instance id="plane">
    <typ:attribute id="wings">
      <typ:boolean-val>true</typ:boolean-val>
    </typ:attribute>
  </typ:instance>
</typ:entity>
```

Example (Specific):

```
<typ:list-toy>
  <typ:toy id="plane">
    <typ:wings>
      <typ:boolean-val>true</typ:boolean-val>
    </typ:wings>
  </typ:toy>
</typ:list-toy>
```

Node:attribute

An attribute element can be used to set an attribute value or configured to indicate it is an outcome to be determined. Configuring an attribute as an outcome can be done by specifying an *attribute:outcome-style*.

In the Specific model there is no attribute element. The element for an attribute (named for that attribute) serves to request a value (if the outcome-style is specified) or set the value. In the specific model, the attributes for attribute also exist on an attribute.

See also: [Important Note: do not set value and specify outcome style for same attribute.](#)

attribute:id

This is the requested attribute name.

attribute:outcome-style

This is the default outcome-style to use whether known or unknown. The outcome style attribute can be "value-only" (no decision report, just the value), "base-attributes" (a decision reports showing the relevant attributes, but only the base level ones), and "decision-report" a full decision report.

attribute:known-outcome-style

This is the outcome style to use when the attribute is known, and overrides the default style specified in outcome-style. This attribute takes the same values as outcome-style.

attribute:unknown-outcome-style

This is the outcome style to use when the attribute is unknown or uncertain, and overrides the default style specified in outcome-style. This attribute takes the same values as outcome-style.

Example – Configuring an attribute outcome:

In the generic and specific examples below, we are requesting an attribute called `parentsSatisfy` from the global instance.

```
<typ:global-instance>
  <typ:attribute id="parentsSatisfy"
    outcome-style="decision-report" />
</typ:global-instance>
```

Example – Configuring an attribute outcome (Specific):

```
<typ:global-instance>
  <typ:parentsSatisfy outcome-style="decision-report"/>
</typ:global-instance>
```

Example – Setting an attribute value:

This sets the value of the Boolean attribute `wings` to value of 'false'. See [Important Note: do not set value and specify outcome style for same attribute.](#)

```
<typ:attribute id="wings">
  <typ:boolean-val>false</typ:boolean-val>
</typ:attribute>
```

Example – Setting an attribute value (Specific):

This sets the value of the Boolean attribute `wings` to value of 'false'

```
<typ:wings>
  <typ:boolean-val>false</typ:boolean-val>
</typ:wings>
```

Node:boolean-val

A boolean value: the text of this node must be "true" or "false".

Node:date-val

A date value: the text of this node must be a valid date in yyyy-mm-dd format (<year>-<month as two digit number>-<day of month>).

Node:datetime-val

A date and time value: the text of this node must be a valid date in yyyy-mm-ddT^Thh:mm:ss format (<year>-<month as two digit number>-<day of month>T^Thh:mm:ss).

Node:time-val

A time-of-day value: the text of this node must be a valid date hh:mm:ss format (<hours 0-23>:<minutes>:<seconds>).

Node:number-val

A number value: the text of this node must be a valid number. The `number-val` element is used for all numeric values

including currency.

Node:text-val

A text value: supports any arbitrary string.

Node:unknown-val

This empty node indicates that the attribute has an unknown value. You never have to set this attribute. If you want the value of an attribute to be unknown, then do not set a value, or leave the attribute out entirely as the default value for an attribute is unknown.

Example:

This wings attribute has an unknown value. This could also be indicated by leaving the value out of the XML altogether.

```
<typ:attribute id="wings"/>
```

Node:uncertain-val

This empty node indicates that the attribute has an uncertain value. You can set this element to indicate that a particular attribute is uncertain.

Example:

This wings attribute has an uncertain value.

```
<typ:attribute id="wings">  
  <typ:uncertain-val/>  
</typ:attribute>
```

Node:change-point

Change points can be used to indicate a changing value over time.

Change-point:date

This indicates the date at which the value specified within the change point takes effect.

Example:

In this example, the happiness of the people changes from being unknown before 1 January 2007 to false on 1 January, and then changes to true on 28 February 2007.

```
<people_happy type="boolean">  
  <unknown-val/>  
  <change-point date="2007-01-01">  
    <boolean-val>>false</boolean-val>  
  </change-point>  
  <change-point date="2007-02-28">  
    <boolean-val>>true</boolean-val>  
  </change-point>  
</people_happy>
```

Node:relationship

This element represents a reference relationship between one entity instance and one or more entity instances. This

element is also used to request the value (the targets) of an inferred relationship (also known as a 'relationship outcome'). In the Specific model there is no relationship element. The element for an relationship (named for that relationship) serves to request a value (if the outcome-style is specified) or set the value.

Example – Configuring a Relationship Outcome:

In the generic and specific examples below, we are requesting a relationship called qualifying_claimants from the global instance.

```
<typ:global-instance>
...
  <typ:relationship id="qualifying_claimants" outcome-style="decision-report" />
</typ:global-instance>
```

Example – Configuring a Relationship Outcome (Specific):

```
<typ:global>
...
  <typ:relationships>
    <typ:qualifying_claimants" outcome-style="decision-report"/>
  </typ:relationships>
</typ:global>
```

The relationship node has one or more target elements (see Node:target) that represents the entities at the other end of the relationship. In the specific format a relationship will be represented by an element with a name matching the name of the relationship.

attribute:id

Is the name of the relationship. In specific format this attribute is not needed and does not exist.

Example – Specifying a relationship’s targets:

The relationship below is called children and the target is bobjr (who must be declared as an entity instance in the global instance).

```
<typ:relationship id="children">
  <typ:target instance-id="bobjr" />
</typ:relationship>
```

Example- Specifying a relationship’s targets (Specific):

```
<typ:relationships>
  <typ:children>
    <typ:target instance-id="bobjr" />
  </typ:children>
</typ:relationships>
```

attribute:state

The state of the relationship.

attribute:inferred

This attribute is a Boolean value and it indicates if the relationship was inferred. This attribute is populated by the Oracle Determination Server and it is returned in the response.

attribute:outcome-style

This is the default outcome-style to use whether known or unknown. The outcome style attribute can be "value-only" (no decision report, just the value), "base-attributes" (a decision reports showing the relevant attributes, but only the base level ones), and "decision-report" a full decision report.

attribute:known-outcome-style

This is the outcome style to use when the attribute is known, and overrides the default style specified in outcome-style. This attribute takes the same values as outcome-style.

attribute:unknown-outcome-style

This is the outcome style to use when the attribute is unknown or uncertain, and overrides the default style specified in outcome-style. This attribute takes the same values as outcome-style.

Node:target

This sub-node of relationship represents the entities at the other end of the relationship.

attribute:entity-id

Is a reference to the target entity instance. entity-id is an XSD:ID and so must be unique and must match an entity's id attribute (see [Node:entity](#)).

Example:

The target below can refer to any entity in the session-data. There must be an entity with an id="bob-jrr".

```
<typ:target instance-id="bobjr" />  
<!--refers to ... -->  
<typ:entity id="bobjr" >  
...  
</typ:entity>
```

Node:properties

This is a list of custom properties and can exist within a screen, screen control, attribute, or entity.

Custom properties are set in Oracle Policy Modeling, and are only displayed if show-properties is set to true in the [Node:config](#) section of the assess.

A properties list will have one or more property items in it.

Node:decision-report

A decision report is a sub-element of an attribute, and provides information on how the value of an attribute was derived (or why it is unknown). A decision report appears when a request for an attribute value is made, and the outcome-style attribute is set to anything except "value-only" (see [Node:attribute](#)).

A decision report is populated by Oracle Determinations Server and only ever appears in the Response. You do not create decision reports in an attribute in a request. A decision report contains [Node:attribute-decision-node](#) and relationship-node elements (see [Node:relationship-node](#)). The sub-elements are contributing attributes and relationships to the value of the attribute.

Attribute:report-style

The report style indicates the report style and will be the style requested in the attribute-outcome. Valid values are "decision-report" and "base-attributes".

Example:

```
<typ:global-instance>
  <typ:attribute id="parentsSatisfy" outcome-style="decision-
    report" />
  ...
```

This is requesting the value for the attribute "parentsSatisfy" but the returned value will be the attribute (if known), and a decision report.

The response below is the attribute "parentsSatisfy". We can tell that the boolean value is known. Below the value is the beginning of the decision report element (line 3).

```
1 <typ:attribute id="parentsSatisfy" type="boolean">
2   <typ:boolean-val>true</typ:boolean-val>
3   <typ:decision-report report-style="decision-report">
4     <typ:attribute-node id="childSatisfies"
5       instance-id="bob" entity-id="person" attribute-id="dn:0"
6       text="" type="boolean">
7       <typ:boolean-val>true</typ:boolean-val>
8     <typ:attribute-node attribute-id="condition"
9       instance-id="bobjr" entity-id="person" attribute-id="dn:1"
10      text="" type="boolean">
11      <typ:boolean-val>true</typ:boolean-val>
12    <typ:attribute-node attribute-id="wings"
13      instance-id="plane" entity-id="toy" attribute-id="dn:2"
14      text="" type="boolean">
15      <typ:boolean-val>true</typ:boolean-val>
16    </typ:attribute-node>
17    <typ:attribute-node attribute-id="wings"
18      instance-id="tip-truck" entity-id="toy" attribute-id="dn:3"
19      text="" type="boolean">
20      <typ:boolean-val>true</typ:boolean-val>
21    </typ:attribute-node>
22  </typ:attribute-node>
23 </typ:decision-report> 25 </typ:attribute>
```

Node:attribute-node

This sub-node of decision-report (see [Node:decision-report](#)) represents a contributing attribute to the decision report. It indicates an attribute of an entity. An attribute-node can contain other attribute-node and relationship-node

(see [Node: relationship-node](#)), if the attribute is itself inferred from other attributes. The entire inferencing structure for a decision-report is called a decision tree.

attribute:id

A unique identifier for this decision node. This is useful when an attribute-node occurs more than once in a decision-report. Rather than print the entire attribute-node again a reference will be made to the first appearance of the attribute decision-node.

Example:

In the example below, the second time the attribute node "dn:0" appears its contents is replaced by an "already-proven" element. Note that the id of both occurrences has the same id (dn:0) to make it easier to reference the original appearance of the attribute-decision-node.

```
<typ:attribute-node attribute-id="childSatisfies"
  entity-id="person" instance-id="bob" id="dn:0">
  ...
  <!-- this may contain a large complicated decision tree -->
</typ:attribute-node>
...

<typ:attribute-node attribute-id="childSatisfies"
  entity-id="person" instance-id="bob" id="dn:0">
  <typ:already-proven>See above dn:0</typ: already-proven>
</typ:attribute-decision-node>
```

attribute:entity-id

The type of entity to which the attribute belongs.

attribute:instance-id

The unique identifier for the entity to which the attribute belongs.

attribute:hypothetical-instance

Indicates whether the instance to which the entity belongs was explicitly created or implicitly created by the engine in order to resolve unknown relationships.

attribute:attribute-id

Name of the attribute.

attribute:type

The type of attribute. It will be one of "boolean", "text", "number", "currency" or "date".

attribute:text

The text associated with the attribute. This will usually be something like "child is not satisfied" (if false) or "child is satisfied" (if true).

attribute:inferred

This attribute is a Boolean value and it indicates if the relationship was inferred. This attribute is

populated by the Oracle Determination Server and it is returned in the response.

attribute:start-relevance

This is a Time Based Reasoning feature. It indicates when an attribute-node is only relevant for a particular time period. The start-relevance attribute indicates the start of the period on which this attribute's value is relevant to the goal. The absence of this attribute means that this attribute is relevant from the earliest possible date.

attribute:end-relevance

This is a Time Based Reasoning feature. The end-relevance attribute indicates the date at which the attribute value ceases to be relevant to the goal. The absence of this attribute means that this attribute is relevant until the latest possible date.

Node:relationship-node

This sub-node of relationship represents an unresolved relationship. A relationship-node will only occur in a decision report where an attribute value is not known (an in-decision report). A relationship-node and an attribute-node can contain other attribute-node and relationship-node. But only if the configuration element `resolve-indecision-relationships` is set to true (see [Node:config](#)). This is because normally, the engine decision trees stop when they reach an unresolved relationship, making it the last element in a decision tree.

attribute:id

A unique identifier for this decision node. For more information on how an id is used (see [Node:attribute-decision-node](#)).

attribute:relationship-id

The name of the relationship.

attribute:source-entity-id

The source entity for the relationship.

attribute:source-instance-id

The source entity instance for the relationship.

attribute:hypothetical-instance

Indicates whether the instance to which the entity belongs was explicitly created or implicitly created by the engine in order to resolve unknown relationships.

attribute:target-entity-id

The target entity for the other end of the relationship.

attribute:state

The state of the relationship. Because a relationship-node only occurs in a decision report if it is unknown, the state should always be "unknown".

attribute:inferred

Indicates whether the relationship is inferred.

attribute:start-relevance

This is a Time Based Reasoning feature. It indicates when a relationship-node is only relevant for a particular time period. The start-relevance attribute indicates the start of the period on which this relationship is relevant to the goal. The absence of this attribute means that this relationship is relevant from the earliest possible date.

attribute:end-relevance

This is a Time Based Reasoning feature. The end-relevance attribute indicates the date at which the relationship ceases to be relevant to the goal. The absence of this attribute means that this relationship is relevant until the latest possible date.

Example:

In the example below, we can see that the attribute-node "condition" is unknown because the relationship "plays-with" between a person and toy is unknown (line 4).

```
1 <typ:attribute-node attribute-id="condition"
2     entity-id="person" id="dn:2" text="" type="boolean">
3 <typ:unknown-val/>
4 <typ:relationship-node id="dn:3"
5     relationship-id="plays-with" state="unknown" target="toy">
6 <typ:attribute-node attribute-id="wings"
7     entity-id="toy" id="dn:4" text="" type="boolean">
8 <typ:unknown-val/>
9 </typ:attribute-node>
10 </typ:relationship-node>
11 </typ:attribute-node>
```

Node:events

The events element can contain one or more event elements (see [Node:event](#)). This element will occur in the response when an assess operation has triggered either a standard event, or an "error". If an error event has been triggered, then the Oracle Determinations Server will always return a SOAP Fault. The errors will be in the body of the SOAP Fault (see: [Node:config show-events](#)).

Example:

In this example, the SOAP Fault contains an error message, that a value passed in a request was -3.0. In this case the error was generated because the number is expected to be greater than 0.

```
1 <SOAP-ENV:Fault>
2 <faultcode>SOAP-ENV:Client</faultcode>
3 <faultstring>The Rulebase generated an error event</faultstring>
4 <detail>
5 <typ:error-response>
6 <typ:code>assess.request.event.error</typ:code>
7 <typ:message>The Rulebase generated
8     an error event</typ:message>
9 <typ:events>
10 <typ:event instance-id="my-attribute-holder" name="error">
11 <typ:message>"The event-field is negative" </typ:message>
12 <typ:decision-report report-style="base-attributes">
13 <typ:attribute-node attribute-id="event_field"
15     instance-id="my-attribute-holder"
16     entity-id="attrholder" id="dn:0"
```

```

17         text="Attrholder's event-field is -3.0."
18         type="number">
19         <typ:number-val>-3.0</typ:number-val>
20     </typ:attribute-node>
21 </typ:decision-report>
22 </typ:event>
23 </typ:events>
24 </typ:error-response>
25 </detail>
26 </SOAP-ENV:Fault>

```

Node:event

When the Oracle Determinations Server returns an event (see [Node:events](#)) an event element is generated. An event element can represent either a standard event, or an "error".

Any event defined in the Rulebase with the name error, when the event is triggered, will be returned in a SOAP:Fault. Error events will result in SOAP errors regardless of the "show-events" setting in the assessment configuration.

All other events are returned only when show-events is set to true.

An event will have a message sub-element, containing the text of the event message. It will also have a decision report, giving a decision tree of the cause of the event (see [Node:decision-report](#)).

attribute:name

This is the name of the event. Currently supported events in Oracle Determinations Server are "error" and "warning" events, so the name will be either of these.

attribute:instance-id

When an event is associated with a particular entity instance. The entity-id attribute will contain as XSD:IDREF to that entity.

Example:

In the example below, a warning event has been generated in a response.

```

1 <typ:events>
2   <typ:event entity-id="my-attribute-holder" name="Warning">
3     <typ:message>"The event-field is more than 100"</typ:message>
4     <typ:decision-report report-style="base-attributes">
5       <typ:attribute-node
6         attribute-id="event_field"
7         instance-id="my-attribute-holder"
8         entity-id="attrholder" id="dn:0"
9         text="Attrholder's event-field is 102.0." type="number">
10        <typ:number-val>102.0</typ:number-val>
11      </typ:attribute-node>
12      <typ:attribute-node attribute-id="boolean_field"
13        instance-id="my-attribute-holder"
14        entity-id="attrholder" id="dn:1"
15        text="Attrholder's boolean-field is true."
16        type="boolean">
17        <typ:boolean-val>true</typ:boolean-val>
18      </typ:attribute-node>

```

```
19     </typ:decision-report>
20   </typ:event>
21 </events>
```

See also:

- [Example: Assess Request xml](#)
- [Example: Assess Response xml](#)

Assess API Events

Go to:

[Technical notes](#)

[Examples](#)

The Assess API offers four events that fire at various points during the process flow, each of which is described in the following table; click on the appropriate event name in the table to go to related examples.

Name	Encapsulated objects	Description	Event Handler Interface
OnAfterThinkEvent	AssessConfig - The assess configuration object provided for this assess request Session - The Determinations Engine being used to conduct this assessment	This event fires immediately after a think has been conducted on the assessment. It provides an opportunity to add add/remove/or alter the data in the session before the assess results are generated and returned.	OnAfterThinkEventHandler
OnBeforeThinkEvent	AssessConfig - The assess configuration object provided for this assess request Session - The Determinations Engine Session being used to conduct this assessment	This event fires after the AssessData has been mapped into the session but before a think has been conducted. It provides the opportunity to augment the data in the session that was provided by the AssessData object.	OnBeforeThinkEventHandler
OnMapDataEvent	AssessConfig - The assess configuration object provided for this assess request Rulebase - the rulebase this assessment is going to be conducted using AssessData - the data that the assessment will be based upon	This event fires prior to the AssessData being mapped into the Determinations Engine Session. It provides an opportunity to augment the data used to conduct the assessment.	OnMapDataEventHandler
OnReturnResultEvent	AssessResult - the result of the assessment	This event fires after the assessment's results have been calculated but prior to that information being returned. It provides an opportunity to modify the assess results.	OnReturnResultEventHandler

Technical notes

- Installation and configuration is like all other plugins (see Extension installation, loading, invocation and discovery)
- Execution cycle is pretty much the same as they are for Interviews (see information about [Event and Event Handlers](#))

- Multiple event handlers can be registered against the same event handler. However, the event handler implementation(s) may not make any assumptions about the order in which those event handlers are registered or processed when an event is fired.
- It is permissible for all event handlers to modify the state of the objects passed into the event. However, those changes are cumulative, that is if multiple handlers are registered any changes to the state of the objects by one event handler, those changes will be reflected in the data passed through to subsequent handlers.

Examples

Go to:

[OnMapDataEventHandler](#)

[OnBeforeThinkEvenHandler](#)

[OnAfterThinkEventHandler](#)

[OnReturnResultEventHandler](#)

OnMapDataEventHandler

Usage scenarios

- Adding, removing or altering the data provided in the assess request
- Adding, removing or changing the configuration options of the outcome(s) provided in the assess request
- Modifying the configuration options passed in the assess request.

Example - Adding additional outcomes

This example uses a rulebase to calculate the total amount of pocket money a person has to pay. It adds an additional outcome for each child to show the amount of pocket money that child has earned.

Sample Java code can be found at:

`<DS_JAVA_RUNTIME_DIR>\examples\determinations-server\onmap-dataevent-handler`

Sample C# code can be found at:

`<DS_CSHARP_RUNTIME_DIR>\examples\determinations-server\onmap-dataevent-handler`

Example request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ-
typ-
="http://oracle.com/determinations/server/10.4/PocketMoneyComputation/assess/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:assess-request>
      <typ:config>
        <typ:show-silent>>false</typ:show-silent>
        <typ:show-invisible>>false</typ:show-invisible>
      </typ:config>
    </typ:assess-request>
  </soapenv:Body>
</soapenv:Envelope>
```

```

        <typ:show-properties>>false</typ:show-properties>
        <typ:show-events>>false</typ:show-events>
        <typ:resolve-indecision-relationships>>false</typ:resolve-indecision-
relationships>
    </typ:config>
    <typ:global-instance>
        <typ:person_name>
            <typ:text-val>Bob</typ:text-val>
        </typ:person_name>
        <typ:base_rate>
            <typ:number-val>2.0</typ:number-val>
        </typ:base_rate>
        <typ:total_money outcome-style="decision-report"/>
        <typ:list-child>
            <typ:child id="fred">
                <typ:child_name>
                    <typ:text-val>Fred</typ:text-val>
                </typ:child_name>
                <typ:child_age>
                    <typ:number-val>5.0</typ:number-val>
                </typ:child_age>
            </typ:child>
            <typ:child id="mary">
                <typ:child_name>
                    <typ:text-val>Mary</typ:text-val>
                </typ:child_name>
                <typ:child_age>
                    <typ:number-val>7.0</typ:number-val>
                </typ:child_age>
            </typ:child>
        </typ:list-child>
    </typ:global-instance>
</typ:assess-request>
</soapenv:Body>
</soapenv:Envelope>

```

Example response

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ-
="http://oracle.com/determinations/server/10.4/PocketMoneyComputation/assess/types">
    <SOAP-ENV:Header>
        <i18n:international>
            <i18n:locale>en_GB</i18n:locale>
            <i18n:tz>GMT+0800</i18n:tz>
        </i18n:international>
    </SOAP-ENV:Header>

```

```
<SOAP-ENV:Body>
  <typ:assess-response>
    <typ:global-instance>
      <typ:total_money type="currency" inferred="true">
        <typ:number-val>24.0</typ:number-val>
        <typ:decision-report report-style="decision-report">
          <typ:attribute-node id="dn:0" entity-id="global" instance-id=-
="global" hypothetical-instance="false" attribute-id="total_money" type="currency"
text="The total amount of pocket money Bob has to pay is $24.00." inferred="true">
            <typ:number-val>24.0</typ:number-val>
            <typ:relationship-node id="dn:1" source-entity-id="global"
source-instance-id="global" hypothetical-instance="false" target-entity-id="child"
relationship-id="children" state="known" inferred="false">
              <typ:target instance-id="fred"/>
              <typ:target instance-id="mary"/>
            </typ:relationship-node>
            <typ:attribute-node id="dn:2" entity-id="child" instance-id=-
="fred" hypothetical-instance="false" attribute-id="child_money" type="currency"
text="The amount of pocket money Fred has earned is $10.00." inferred="true">
              <typ:number-val>10.0</typ:number-val>
              <typ:attribute-node id="dn:3" entity-id="global" instance-
id="global" hypothetical-instance="false" attribute-id="base_rate" type="currency"
text="The pocket money base rate is $2.00." inferred="false">
                <typ:number-val>2.0</typ:number-val>
              </typ:attribute-node>
              <typ:attribute-node id="dn:4" entity-id="child" instance-
id="fred" hypothetical-instance="false" attribute-id="child_age" type="currency"
text="Fred's age is $5.00." inferred="false">
                <typ:number-val>5.0</typ:number-val>
              </typ:attribute-node>
            </typ:attribute-node>
            <typ:attribute-node id="dn:5" entity-id="child" instance-id=-
="mary" hypothetical-instance="false" attribute-id="child_money" type="currency"
text="The amount of pocket money Mary has earned is $14.00." inferred="true">
              <typ:number-val>14.0</typ:number-val>
              <typ:already-proven-node id="dn:3"/>
              <typ:attribute-node id="dn:6" entity-id="child" instance-
id="mary" hypothetical-instance="false" attribute-id="child_age" type="currency"
text="Mary's age is $7.00." inferred="false">
                <typ:number-val>7.0</typ:number-val>
              </typ:attribute-node>
            </typ:attribute-node>
          </typ:decision-report>
        </typ:total_money>
        <typ:base_rate type="currency">
          <typ:number-val>2.0</typ:number-val>
        </typ:base_rate>
      </typ:global-instance>
    </typ:assess-response>
  </SOAP-ENV:Body>
```

```

<typ:person_name type="text">
  <typ:text-val>Bob</typ:text-val>
</typ:person_name>
<typ:list-child inferred="false">
  <typ:child id="fred">
    <typ:child_money type="currency" inferred="true">
      <typ:number-val>10.0</typ:number-val>
      <typ:decision-report report-style="decision-report">
        <typ:attribute-node id="dn:0" entity-id="child" instance-
id="fred" hypothetical-instance="false" attribute-id="child_money" type="currency"
text="The amount of pocket money Fred has earned is $10.00." inferred="true">
          <typ:number-val>10.0</typ:number-val>
          <typ:attribute-node id="dn:1" entity-id="global"
instance-id="global" hypothetical-instance="false" attribute-id="base_rate" type-
e="currency" text="The pocket money base rate is $2.00." inferred="false">
            <typ:number-val>2.0</typ:number-val>
          </typ:attribute-node>
          <typ:attribute-node id="dn:2" entity-id="child"
instance-id="fred" hypothetical-instance="false" attribute-id="child_age" type-
e="currency" text="Fred's age is $5.00." inferred="false">
            <typ:number-val>5.0</typ:number-val>
          </typ:attribute-node>
        </typ:attribute-node>
      </typ:decision-report>
    </typ:child_money>
    <typ:child_age type="currency">
      <typ:number-val>5.0</typ:number-val>
    </typ:child_age>
    <typ:child_name type="text">
      <typ:text-val>Fred</typ:text-val>
    </typ:child_name>
  </typ:child>
  <typ:child id="mary">
    <typ:child_money type="currency" inferred="true">
      <typ:number-val>14.0</typ:number-val>
      <typ:decision-report report-style="decision-report">
        <typ:attribute-node id="dn:0" entity-id="child" instance-
id="mary" hypothetical-instance="false" attribute-id="child_money" type="currency"
text="The amount of pocket money Mary has earned is $14.00." inferred="true">
          <typ:number-val>14.0</typ:number-val>
          <typ:attribute-node id="dn:1" entity-id="global"
instance-id="global" hypothetical-instance="false" attribute-id="base_rate" type-
e="currency" text="The pocket money base rate is $2.00." inferred="false">
            <typ:number-val>2.0</typ:number-val>
          </typ:attribute-node>
          <typ:attribute-node id="dn:2" entity-id="child"
instance-id="mary" hypothetical-instance="false" attribute-id="child_age" type-
e="currency" text="Mary's age is $7.00." inferred="false">
            <typ:number-val>7.0</typ:number-val>
          </typ:attribute-node>
        </typ:attribute-node>
      </typ:decision-report>
    </typ:child_money>
    <typ:child_age type="currency">
      <typ:number-val>7.0</typ:number-val>
    </typ:child_age>
    <typ:child_name type="text">
      <typ:text-val>Mary</typ:text-val>
    </typ:child_name>
  </typ:child>
</typ:list-child>

```

```

        <typ:number-val>7.0</typ:number-val>
    </typ:attribute-node>
</typ:attribute-node>
</typ:decision-report>
</typ:child_money>
<typ:child_age type="currency">
    <typ:number-val>7.0</typ:number-val>
</typ:child_age>
<typ:child_name type="text">
    <typ:text-val>Mary</typ:text-val>
</typ:child_name>
</typ:child>
</typ:list-child>
</typ:global-instance>
</typ:assess-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

OnBeforeThinkEventHandler

Usage scenarios

- Loading reference data
- Adding additional data directly into the Determinations Engine Session.

Example - Adding reference data

This example uses a rulebase to calculate the total amount of pocket money a person has to pay. The calculation is based on the pocket money 'base rate' which is piece of reference data that gets loaded into the session using an event handler.

Sample Java code can be found at:

```
<DS_JAVA_RUNTIME_DIR>\examples\determinations-server\ onbefore-thinkevent-handler
```

Sample C# code can be found at:

```
<DS_CSHARP_RUNTIME_DIR>\examples\determinations-server\ onbefore-thinkevent-handler
```

Example request

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ-
="http://oracle.com/determinations/server/10.4/PocketMoneyComputation/assess/types">
    <soapenv:Header/>
    <soapenv:Body>
        <typ:assess-request>
            <typ:config>
                <typ:show-silent>>false</typ:show-silent>
                <typ:show-invisible>>false</typ:show-invisible>
                <typ:show-properties>>false</typ:show-properties>
            </typ:config>
        </typ:assess-request>
    </soapenv:Body>
</soapenv:Envelope>

```

```

        <typ:show-events>>false</typ:show-events>
        <typ:resolve-indecision-relationships>>false</typ:resolve-indecision-
relationships>
    </typ:config>
    <typ:global-instance>
        <typ:person_name>
            <typ:text-val>Bob</typ:text-val>
        </typ:person_name>
        <typ:total_money outcome-style="decision-report"/>
        <typ:list-child>
            <typ:child id="fred">
                <typ:child_name>
                    <typ:text-val>Fred</typ:text-val>
                </typ:child_name>
                <typ:child_age>
                    <typ:number-val>5.0</typ:number-val>
                </typ:child_age>
            </typ:child>
            <typ:child id="mary">
                <typ:child_name>
                    <typ:text-val>Mary</typ:text-val>
                </typ:child_name>
                <typ:child_age>
                    <typ:number-val>7.0</typ:number-val>
                </typ:child_age>
            </typ:child>
        </typ:list-child>
    </typ:global-instance>
</typ:assess-request>
</soapenv:Body>
</soapenv:Envelope>

```

Example response

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ-
="http://oracle.com/determinations/server/10.4/PocketMoneyComputation/assess/types">
    <SOAP-ENV:Header>
        <i18n:international>
            <i18n:locale>en_GB</i18n:locale>
            <i18n:tz>GMT+0800</i18n:tz>
        </i18n:international>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <typ:assess-response>
            <typ:global-instance>
                <typ:total_money type="currency" inferred="true">

```

```

        <typ:number-val>60.0</typ:number-val>
        <typ:decision-report report-style="decision-report">
            <typ:attribute-node id="dn:0" entity-id="global" instance-id=-
            ="global" hypothetical-instance="false" attribute-id="total_money" type="currency"
            text="The total amount of pocket money Bob has to pay is $60.00." inferred="true">
                <typ:number-val>60.0</typ:number-val>
                <typ:relationship-node id="dn:1" source-entity-id="global"
                source-instance-id="global" hypothetical-instance="false" target-entity-id="child"
                relationship-id="children" state="known" inferred="false">
                    <typ:target instance-id="fred"/>
                    <typ:target instance-id="mary"/>
                </typ:relationship-node>
                <typ:attribute-node id="dn:2" entity-id="child" instance-id=-
                ="fred" hypothetical-instance="false" attribute-id="child_money" type="currency"
                text="The amount of pocket money Fred has earned is $25.00." inferred="true">
                    <typ:number-val>25.0</typ:number-val>
                    <typ:attribute-node id="dn:3" entity-id="global" instance-
                    id="global" hypothetical-instance="false" attribute-id="base_rate" type="currency"
                    text="The pocket money base rate is $5.00." inferred="false">
                        <typ:number-val>5.0</typ:number-val>
                    </typ:attribute-node>
                    <typ:attribute-node id="dn:4" entity-id="child" instance-
                    id="fred" hypothetical-instance="false" attribute-id="child_age" type="currency"
                    text="Fred's age is $5.00." inferred="false">
                        <typ:number-val>5.0</typ:number-val>
                    </typ:attribute-node>
                </typ:attribute-node>
                <typ:attribute-node id="dn:5" entity-id="child" instance-id=-
                ="mary" hypothetical-instance="false" attribute-id="child_money" type="currency"
                text="The amount of pocket money Mary has earned is $35.00." inferred="true">
                    <typ:number-val>35.0</typ:number-val>
                    <typ:already-proven-node id="dn:3"/>
                    <typ:attribute-node id="dn:6" entity-id="child" instance-
                    id="mary" hypothetical-instance="false" attribute-id="child_age" type="currency"
                    text="Mary's age is $7.00." inferred="false">
                        <typ:number-val>7.0</typ:number-val>
                    </typ:attribute-node>
                </typ:attribute-node>
            </typ:attribute-node>
        </typ:decision-report>
    </typ:total_money>
    <typ:base_rate type="currency">
        <typ:number-val>5.0</typ:number-val>
    </typ:base_rate>
    <typ:person_name type="text">
        <typ:text-val>Bob</typ:text-val>
    </typ:person_name>
    <typ:list-child inferred="false">

```

```

        <typ:child id="fred">
            <typ:child_age type="currency">
                <typ:number-val>5.0</typ:number-val>
            </typ:child_age>
            <typ:child_name type="text">
                <typ:text-val>Fred</typ:text-val>
            </typ:child_name>
        </typ:child>
        <typ:child id="mary">
            <typ:child_age type="currency">
                <typ:number-val>7.0</typ:number-val>
            </typ:child_age>
            <typ:child_name type="text">
                <typ:text-val>Mary</typ:text-val>
            </typ:child_name>
        </typ:child>
    </typ:list-child>
</typ:global-instance>
</typ:assess-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

OnAfterThinkEventHandler

Usages

- Adding additional data into the session based some the value of some inferred attribute

Example - Conditionally adding additional data to the session

This example uses the OnAfterThinkEventHandler to add additional data to the session based on the value of an attribute that was inferred using the data that was submitted into the session.

Sample Java code can be found at:

```
<DS_JAVA_RUNTIME_DIR>\examples\determinations-server\ onafter-thinkevent-handler
```

Sample C# code can be found at:

```
<DS_CSHARP_RUNTIME_DIR>\examples\determinations-server\ onafter-thinkevent-handler
```

Example request

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ="http://oracle.com/determinations/server/10.4/IncomeRebate/assess/types">
    <soapenv:Header/>
    <soapenv:Body>
        <typ:assess-request>
            <typ:config>

```

```

        <typ:show-silent>false</typ:show-silent>
        <typ:show-invisible>false</typ:show-invisible>
        <typ:show-properties>false</typ:show-properties>
        <typ:show-events>false</typ:show-events>
        <typ:resolve-indecision-relationships>false</typ:resolve-indecision-
relationships>
    </typ:config>
    <typ:global-instance>
        <typ:total_rebate outcome-style="decision-report"/>
        <typ:person_income>
            <typ:number-val>30000</typ:number-val>
        </typ:person_income>
    </typ:global-instance>
</typ:assess-request>
</soapenv:Body>
</soapenv:Envelope>

```

Example response

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/IncomeRebate/assess/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_GB</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:assess-response>
      <typ:global-instance>
        <typ:total_rebate type="currency" inferred="true">
          <typ:number-val>270.0</typ:number-val>
          <typ:decision-report report-style="decision-report">
            <typ:attribute-node id="dn:0" entity-id="global" instance-id=-
="global" hypothetical-instance="false" attribute-id="total_rebate" type-
e="currency" text="The person's total annual rebate is $270.00." inferred="true">
              <typ:number-val>270.0</typ:number-val>
            <typ:attribute-node id="dn:1" entity-id="global" instance-id=-
="global" hypothetical-instance="false" attribute-id="eligible_income" type-
e="currency" text="The portion of the person's annual income eligible for a rebate
is $3,000.00." inferred="true">
              <typ:number-val>3000.0</typ:number-val>
            <typ:attribute-node id="dn:2" entity-id="global" instance-
id="global" hypothetical-instance="false" attribute-id="person_income" type-
e="currency" text="The person's annual gross income is $30,000.00." inferred-
d="false">
              <typ:number-val>30000.0</typ:number-val>
            </typ:attribute-node>
          </typ:decision-report>
        </typ:total_rebate>
      </typ:global-instance>
    </typ:assess-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

```
        </typ:attribute-node>
    </typ:attribute-node>
    <typ:relationship-node id="dn:3" source-entity-id="global"
source-instance-id="global" hypothetical-instance="false" target-entity-id=-
="monthly_rebate" relationship-id="personsmonthlyrebate" state="known" inferred-
d="false">
        <typ:target instance-id="0x61c4666f66cb053d"/>
        <typ:target instance-id="0x52268c854ed40345"/>
        <typ:target instance-id="0x601ffca0266f3a67"/>
    </typ:relationship-node>
    <typ:attribute-node id="dn:4" entity-id="monthly_rebate"
instance-id="0x61c4666f66cb053d" hypothetical-instance="false" attribute-id=-
="monthly_rebate_amount" type="currency" text="The person's monthly rebate amount
is $125.00." inferred="true">
        <typ:number-val>125.0</typ:number-val>
        <typ:already-proven-node id="dn:1"/>
        <typ:attribute-node id="dn:5" entity-id="monthly_rebate"
instance-id="0x61c4666f66cb053d" hypothetical-instance="false" attribute-id="r-
rebate_rate" type="currency" text="The monthly rebate rate is $0.50." inferred-
d="false">
            <typ:number-val>0.5</typ:number-val>
        </typ:attribute-node>
    </typ:attribute-node>
    <typ:attribute-node id="dn:6" entity-id="monthly_rebate"
instance-id="0x52268c854ed40345" hypothetical-instance="false" attribute-id=-
="monthly_rebate_amount" type="currency" text="The person's monthly rebate amount
is $82.50." inferred="true">
        <typ:number-val>82.5</typ:number-val>
        <typ:already-proven-node id="dn:1"/>
        <typ:attribute-node id="dn:7" entity-id="monthly_rebate"
instance-id="0x52268c854ed40345" hypothetical-instance="false" attribute-id="r-
rebate_rate" type="currency" text="The monthly rebate rate is $0.33." inferred-
d="false">
            <typ:number-val>0.33</typ:number-val>
        </typ:attribute-node>
    </typ:attribute-node>
    <typ:attribute-node id="dn:8" entity-id="monthly_rebate"
instance-id="0x601ffca0266f3a67" hypothetical-instance="false" attribute-id=-
="monthly_rebate_amount" type="currency" text="The person's monthly rebate amount
is $62.50." inferred="true">
        <typ:number-val>62.5</typ:number-val>
        <typ:already-proven-node id="dn:1"/>
        <typ:attribute-node id="dn:9" entity-id="monthly_rebate"
instance-id="0x601ffca0266f3a67" hypothetical-instance="false" attribute-id="r-
rebate_rate" type="currency" text="The monthly rebate rate is $0.25." inferred-
d="false">
            <typ:number-val>0.25</typ:number-val>
        </typ:attribute-node>
```

```

        </typ:attribute-node>
    </typ:attribute-node>
</typ:decision-report>
</typ:total_rebate>
<typ:person_income type="currency">
    <typ:number-val>30000.0</typ:number-val>
</typ:person_income>
<typ:list-monthly_rebate inferred="false">
    <typ:monthly_rebate id="0x61c4666f66cb053d">
        <typ:rebate_rate type="currency">
            <typ:number-val>0.5</typ:number-val>
        </typ:rebate_rate>
    </typ:monthly_rebate>
    <typ:monthly_rebate id="0x52268c854ed40345">
        <typ:rebate_rate type="currency">
            <typ:number-val>0.33</typ:number-val>
        </typ:rebate_rate>
    </typ:monthly_rebate>
    <typ:monthly_rebate id="0x601ffca0266f3a67">
        <typ:rebate_rate type="currency">
            <typ:number-val>0.25</typ:number-val>
        </typ:rebate_rate>
    </typ:monthly_rebate>
</typ:list-monthly_rebate>
</typ:global-instance>
</typ:assess-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

OnReturnResultEventHandler

Usages

- Augmenting the assess results

Example - Removing outcomes

This example goes through and removes any outcome that is not known, which will result in the service only returning known outcomes.

Sample Java code can be found at:

```
<DS_JAVA_RUNTIME_DIR>\examples\determinations-server\ onreturn-resultevent-handler
```

Sample C# code can be found at:

```
<DS_CSHARP_RUNTIME_DIR>\examples\determinations-server\ onreturn-resultevent-handler
```

Example request

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ-
typ-
="http://oracle.com/determinations/server/10.4/PocketMoneyComputation/assess/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:assess-request>
      <typ:config>
        <typ:show-silent>>false</typ:show-silent>
        <typ:show-invisible>>false</typ:show-invisible>
        <typ:show-properties>>false</typ:show-properties>
        <typ:show-events>>false</typ:show-events>
        <typ:resolve-indecision-relationships>>false</typ:resolve-indecision-
relationships>
      </typ:config>
      <typ:global-instance>
        <typ:person_name>
          <typ:text-val>Bob</typ:text-val>
        </typ:person_name>
        <typ:total_money outcome-style="decision-report"/>
      </typ:global-instance>
    </typ:assess-request>
  </soapenv:Body>
</soapenv:Envelope>

```

Example response

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ-
="http://oracle.com/determinations/server/10.4/PocketMoneyComputation/assess/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_GB</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:assess-response>
      <typ:global-instance>
        <typ:person_name type="text">
          <typ:text-val>Bob</typ:text-val>
        </typ:person_name>
      </typ:global-instance>
    </typ:assess-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Assess Service - ListGoals Operation

Description

The ListGoals service returns a list of goal attributes for a rulebase. The request and response for ListGoals is the same for the Specific and Generic service.

For an attribute to be identified as a goal attribute in a rulebase, it must:

- Be inferred by at least one other attribute
- Not be used to infer any other attributes
- Have a public name

A goal attribute can then be used in the outcome set of an Assess operation.

The input for the ListGoals operation is a string containing the name of the rulebase whose goals are to be listed .

Request

The Call is a simple SOAP Request with no parameters:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:list-goals-request/>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

The response is a complex XML type structured as:

"list-goals-response"

The response element containing all the goals

"entity" (optional, recurring element)

The entity that has at least one top level attribute. It has the following attributes:

"entity-id" - the name of the entity.

"attribute" (mandatory, recurring element)

at least one per entity, one for each goal attribute available. It has the following attributes:

"text" - specifies the question form of the text of the goal attribute.

"id" - specifies the rulebase attribute ID used to uniquely identify the goal attribute.

"type" - the type of value of the attribute. Can be "text-val", "number-val", "currency-val", "date-val", "datetime-val", "time-val" or "boolean-val".

Example Response:

In this example, there are two Global goals. One is, *eligible_teenage_allowance*, which is a boolean value (true or false). The other is *low_income_allowance_payment*, which is a currency (money) value. Note that this example uses the SimpleBenefits rulebase which can be found at `examples\rulebases\compiled\SimpleBenefits.zip` in the Oracle Policy Automation Runtime package.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:list-goals-response>
      <typ:entity entity-id="global">
        <typ:attribute id="eligible_teenage_allowance" text="Is the claimant eligible for the teenage child allowance?" type-
e="boolean"/>
        <typ:attribute id="low_income_allowance_payment" text="What is the claimant's low income allowance payment (per
month)?" type="currency"/>
      </typ:entity>
    </typ:list-goals-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Explicit Data Set - Generic Data Model

The Explicit Data Set is an XML data structure used to describe entities, attributes and relationships. The Generic Data Model uses the same XML schema for every rulebase, defining entities, attributes and relationships as generic types.

In the example below, the element global-instance defines all of the data to be passed into the rulebase session. The session data contains two lists describing three different types of entities: "household", "household_member" and "electricity_provider". Each entity is identified by a unique id "the_household", "fred", "AGL". Attributes set for the household are identified by their id, and Relationships are identified by the name attribute. The way in which entities are nested indicates the containment model of the rulebase: the "household_member" "fred" is declared as a child of the "household" "the_household", where as the "electricity_provider" "AGL" is declared as a child of the global entity instance and it's association with the "household" is specified via the relationship specified on the latter entity instance.

```
<typ:global-instance>
  <typ:entity id="household">
    <typ:instance id="the_household">
      <typ:attribute id="household_location">
        <typ:text-val>Alabama</typ:text-val>
      </typ:attribute>
      <typ:attribute id="household_shelter_situation">
        <typ:text-val>Housing Association</typ:text-val>
      </typ:attribute>
      <typ:attribute id="household_disabled_members">
        <typ:boolean-val>true</typ:boolean-val>
      </typ:attribute>
      <typ:relationships>
        <typ:relationship id="electricity_provider">
          <typ:target instance-id="AGL"/>
        </typ:relationship>
      </typ:relationships>
      <typ:list-entity id="household_member">
        <typ:entity id="fred">
          ...
        </typ:entity>
      </typ:list-entity>
    </typ:instance>
  </typ:entity>
  <typ:entity id="electricity_provider">
    <typ:instance id="AGL">
      ...
    </typ:instance>
  </typ:entity>
</typ:global-instance>
```

See also:

Rulebase generic interface

Explicit Data Set - specific data model

The Explicit Data Set is an XML data structure used to describe entities, attributes and relationships. The Specific Data Model uses a different XML schema for every rulebase, defining specific XML elements for entities attributes and relationships.

In the example below, the element global-instance defines all of the data to be passed into the rulebase session. The session data contains two lists describing three different types of entities: "household", "household_member" and "electricity_provider". Each entity is identified by a unique id "the_household", "fred", "AGL". The attributes, child entities and relationships set on the entity instance are identified by uniquely named element based on the public name of the attribute/entity/relationship. The way in which entities are nested indicates the containment model of the rulebase: the "household_member" "fred" is declared as a child of the "household" "the_household", where as the "electricity_provider" "AGL" is declared as a child of the global entity instance and it's association with the "household" is specified via the relationship specified on the latter entity instance.

The XML elements are specifically named for the entities' attributes, and relationships defined in the rulebase. So there is a household element which has elements defining the household entity's attributes and relationships. There is a list-household_member element which can contain more than one household_member entity.

```
<typ:global-instance>
  <typ:list-household>
    <typ:household id="the_household">
      <typ:household_location>
        <typ:text-val>Alabama</typ:text-val>
      </typ:household_location>
      <typ:household_shelter_situation>
        <typ:text-val>Housing Association</typ:text-val>
      </typ:household_shelter_situation>
      <typ:household_disabled_members>
        <typ:boolean-val>>true</typ:boolean-val>
      </typ:household_disabled_members>
      <typ:relationships>
        <typ:electricity_provider>
          <typ:target instance-id="AGL"/>
        </typ:electricity_provider>
      </typ:relationships>
      <typ:list-household_member>
        <typ:household_member id="fred">
          ...
        </typ:household_member>
      </typ:list-household_member>
    </typ:household>
  </typ:list-household>
  <typ:list-electricity_provider>
    <typ:electricity_provider id="AGL">
      ...
    </typ:electricity_provider>
  </typ:list-electricity_provider>
</typ:global-instance>
```

See also:

[Rulebase specific interface](#)

Rulebase Generic Interface

The WSDL for the Generic Interface remains the same regardless of which particular rulebase service is being called.

The data model is completely generic and does not allow for design time inspection of the inputs and outputs of each rulebase. Instead, the client application queries the Oracle Determinations Server at runtime for required inputs, allowing the client application to be completely loosely coupled as rulebase specific information is discovered at runtime. Note that changes to rules do not change the interface.

Schema sample:

```
<xsd:complexType name="Attribute">
  <xsd:sequence>
    <xsd:choice minOccurs="0">
      <xsd:element name="val" type="xsd:string" />
      <xsd:element name="boolean-val" type="xsd:boolean" />
      <xsd:element name="currency-val" type="xsd:decimal" />
      <xsd:element name="date-val" type="xsd:date" />
      <xsd:element name="number-val" type="xsd:decimal" />
      <xsd:element name="text-val" type="xsd:string" />
    </xsd:choice>
    <xsd:element name="properties" type="ListCustomProperties" minOccurs="0"/>
    <xsd:element name="user-data" type="UserData" minOccurs="0" />
    <xsd:element name="decision-report" type="DecisionReport" minOccurs="0" />
    <xsd:element name="screen" type="ScreenDefinition" minOccurs="0" />
  </xsd:sequence>
  <xsd:attribute name="id" type="xsd:string" use="required"/>
  <xsd:attribute name="state" type="AttributeStateEnum"/>
  <xsd:attribute name="inferencing-type" type="InferencingTypeEnum"/>
</xsd:complexType>
```

XML sample:

```
<typ:attribute id="household_location">
  <typ:text-val>Alabama</typ:text-val>
</typ:attribute>
<typ:attribute id="household_earned_income_deduction">
  <typ:currency-val>450.00</typ:currency-val>
</typ:attribute>
```

Summary of features:

- Loosely coupled (runtime) integration.
- Single interface and only one endpoint.

Rulebase Specific Interface

The WSDL for the Specific Interface changes depending on the rulebase, meaning that at design time, a strong contract for inputs and outputs is exposed by the WSDL.

Client applications using the WSDL are tightly coupled to the data model and any changes to the rules, results in a change to the interface. However, as the rulebase data model is fully exposed in the WSDL, graphical mapping and transformation tools can be used, meaning that hand coded integration is not necessary.

Schema sample:

```
<xsd:complexType name="household">
  <xsd:all>
    <xsd:element name="household_location" type="AttributeText"
      minOccurs="0" maxOccurs="1"/ >
    <xsd:element name="household_shelter_situation" type="AttributeText"
      minOccurs="0" maxOccurs="1"/ >
    <xsd:element name="household_disabled_members" type="AttributeBoolean"
      minOccurs="0" maxOccurs="1"/ >
    <xsd:element name="relationships" minOccurs="0" maxOccurs="1">
  </xsd:complexType>
</xsd:all>
  <xsd:element name="household_members" type="RelationshipInstance"
    minOccurs="0" maxOccurs="1"/>
</xsd:all>
</xsd:complexType>
</xsd:element>
</xsd:all>
<xsd:attribute name="id" type="xsd:ID" use="required"/>
</xsd:complexType>
```

XML sample:

```
...
<typ:household_location>
  <typ:text-val>Alabama</typ:text-val>
</typ: household_location>
```

Summary of features:

- Integration can be performed with the use of graphical mapping tools.
- Choice of tightly coupled (design time) or loosely coupled (runtime) integration.

Process time-varying data in a web service call

What do you want to do?

Supply attribute data that varies over time

Obtain and process a time-varying answer

Note that the rulebase used in the examples that follow can be located at:

`examples\rulebases\compiled\SimpleBenefits.zip` in the Oracle Policy Automation Runtime package.

Supply attribute data that varies over time

Determinations Server

You can use the Determinations Server to get answers on data that may vary over time. The value of an attribute may vary over time, and any determinations that are inferred based on that attribute may also vary over time.

The Determinations Server allows you to provide the values of an attribute for a period of time. You can use the change-point XML element to do this in an Determinations Server Assess request operation that uses session data, such as an Assess operation.

Example

In the example below we are trying to determine if a claimant is eligible for the low income allowance, and the amount of that allowance. The claimants income has varied over time. Initially, their declared annual income was \$13,000, but on 3 June 2009 due to a change in circumstances, it changed to \$18,000. It changed again on 15 September 2009 to \$35,000.

To provide all of the above information in a single request to the Determinations Server, we need to set change points in the attribute, and provide the new value that applies on and after the date specified in the change point.

So, the initial value of the attribute *claimant_income* is 13000.00, but after the change-point dated 2009-06-03 it becomes 18000.00 and again after 2009-09-15 it becomes 35000.00

Generic

```
<attribute id="claimant_income">
  <number-val>13000.00</number-val>
  <change-point date="2009-06-03">
    <number-val>18000.00</number-val>
  </change-point>
  <change-point date="2009-09-15">
    <number-val>35000.00</number-val>
  </change-point>
</attribute>
```

Specific

```
<claimant_income>
  <number-val>13000.00</number-val>
```

```

<change-point date="2009-06-03">
  <number-val>18000.00</number-val>
</change-point>
<change-point date="2009-09-15">
  <number-val>35000.00</number-val>
</change-point>
</claimant_income>

```

Obtain and process a time-varying answer

Determinations Server

Just as you can set data that changes over time, you can also receive answers (based on that data, that changes over time. Just as data going into the Determinations Server might vary over time, so can attribute outcomes returned in the response.

In the section above on *Supply attribute data that varies over time* we saw an example of setting attribute data in a request that changed over time. Now we can see how the returning result containing this sort of data might look.

Example

In the example below we are trying to determine two things: if the claimant is eligible for low income allowance, and what the value of that allowance might be. We can formulate our request with the claimants income which has changed over time:

Initially: 13000, after 2009-06-04: 18000, after 2009-09-15: 35000.

We specify two outcomes: *eligible_low_income_allowance* and *low_income_allowance_payment*.

If you look at the response, you can see the outcomes varying over time.

eligible_low_income_allowance.

Initially true, but after 2009-09-15: false- this is because a claimant must have an income less than 25000 to be eligible for low income allowance.

low_income_allowance_payment.

Initially 70.00, but after 2009-06-03: 20.00, and then after 2009-09-15: 0.00 - this is because the payment amount is calculated off the claimants income (which changed on 2009-06-03 and 2009-09-15. After 2009-09-15 the claimant no longer qualifies, so the amount changes to 0.

Request (generic)

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:assess-request>
      <typ:global-instance>
        <typ:attribute id="eligible_low_income_allowance" outcome-style="value-only"/>
        <typ:attribute id="low_income_allowance_payment" outcome-style="value-

```

```

only"/>
    <typ:attribute id="claimant_income">
      <typ:number-val>13000.00</typ:number-val>
      <typ:change-point date="2009-06-03">
        <typ:number-val>18000.00</typ:number-val>
      </typ:change-point>
      <typ:change-point date="2009-09-15">
        <typ:number-val>35000.00</typ:number-val>
      </typ:change-point>
    </typ:attribute>
    <typ:attribute id="claimant_public_housing_client">
      <typ:boolean-val>true</typ:boolean-val>
    </typ:attribute>
    <typ:attribute id="claimant_date_of_birth">
      <typ:date-val>1981-03-22</typ:date-val>
    </typ:attribute>
  </typ:global-instance>
</typ:assess-request>
</soapenv:Body>
</soapenv:Envelope>

```

Response (generic)

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:assess-response>
      <typ:global-instance>
        <typ:attribute id="eligible_low_income_allowance" type="boolean"
inferred="true">
          <typ:boolean-val>true</typ:boolean-val>
          <typ:change-point date="2009-09-15">
            <typ:boolean-val>false</typ:boolean-val>
          </typ:change-point>
        </typ:attribute>
        <typ:attribute id="low_income_allowance_payment" type="currency"
inferred="true">
          <typ:number-val>70.0</typ:number-val>
          <typ:change-point date="2009-06-03">
            <typ:number-val>20.0</typ:number-val>
          </typ:change-point>

```

```
<typ:change-point date="2009-09-15">
  <typ:number-val>0.0</typ:number-val>
</typ:change-point>
</typ:attribute>
<typ:attribute id="claimant_income" type="currency">
  <typ:number-val>13000.0</typ:number-val>
  <typ:change-point date="2009-06-03">
    <typ:number-val>18000.0</typ:number-val>
  </typ:change-point>
  <typ:change-point date="2009-09-15">
    <typ:number-val>35000.0</typ:number-val>
  </typ:change-point>
</typ:attribute>
<typ:attribute id="claimant_date_of_birth" type="date">
  <typ:date-val>1981-03-22</typ:date-val>
</typ:attribute>
<typ:attribute id="claimant_public_housing_client" type="boolean">
  <typ:boolean-val>true</typ:boolean-val>
</typ:attribute>
</typ:global-instance>
</typ:assess-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Find out what data is required by the web service to get an answer

What do you want to do ?

Get a list of the data needed to reach a decision

Find out the next screen that should be displayed to reach a decision

Note that the rulebase used in the examples that follow can be located at:

`examples\rulebases\compiled\SimpleBenefits.zip` in the Oracle Policy Automation Runtime package.

Get a list of the data needed to reach a decision

Determinations Server

When you send an Assess request to the Determinations Server it is possible that you may not have supplied enough information for the Determinations Server to return the known values for all the outcomes you have requested.

Any attribute outcomes that are unknown will have the `<unknown-val />` element as their value. Note: If you are receiving unknown values for attributes when you do not expect them, check that you have not also specified the relevant attribute as an outcome in the Assess Request (see [Important Note: do not set value and specify outcome style for same attribute](#)).

Any inferred relationship outcomes that are unknown will have a state attribute, which indicates if the inferred relationship is known, unknown, or uncertain.

If an outcome is unknown, you can use a decision report to determine what data you need to provide in order for that outcome to become known.

The procedure for determining what data needs to be provided to reach a decision is a simple sequence:

1. Make the initial request with known existing data. Make sure all outcomes will return a decision report if they are unknown. The base-attributes outcome style is the best style for getting all attributes that you need to provide.
2. Parse the decision reports in response, noting all information that needs to be collected.
3. Get that information, add it to the request and re-submit the request.
4. All outcomes should become known.

Example

In this example, we want to determine if the claimant is eligible for the low income allowance.

Request

The initial request contains an outcome for `eligible_low_income_allowance` attribute, but no data at all. The unknown outcome style is "base-attributes". This will give us a decision report with all the base attributes that we need to provide if the `eligible_low_income_allowance` is unknown. Because we are providing no data in the request, we expect it to be unknown.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <soapenv:Header/>
  <soapenv:Body>
```

```

    <typ:assess-request>
      <typ:global-instance>
        <typ:attribute id="eligible_low_income_allowance" known-outcome-style-
e="value-only" unknown-outcome-style="base-attributes"/>
      </typ:global-instance>
    </typ:assess-request>
  </soapenv:Body>
</soapenv:Envelope>

```

Response

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:assess-response>
      <typ:global-instance>
        <typ:attribute id="eligible_low_income_allowance" type="boolean"
inferred="false">
          <typ:unknown-val/>
          <typ:decision-report report-style="base-attributes">
            <typ:attribute-node id="dn:1" entity-id="global" instance-id=-
="global" hypothetical-instance="false" attribute-id="claimant_income" type-
e="currency" text="The claimant's annual income is unknown." inferred="false">
              <typ:unknown-val/>
            </typ:attribute-node>
            <typ:attribute-node id="dn:2" entity-id="global" instance-id=-
="global" hypothetical-instance="false" attribute-id="claimant_public_housing_cli-
ent" type="boolean" text="Is the claimant a public housing client?"
inferred="false">
              <typ:unknown-val/>
            </typ:attribute-node>
            <typ:attribute-node id="dn:3" entity-id="global" instance-id=-
="global" hypothetical-instance="false" attribute-id="claimant_date_of_birth" type-
e="date" text="The claimant's date of birth is unknown." inferred="false">
              <typ:unknown-val/>
            </typ:attribute-node>
          </typ:decision-report>
        </typ:attribute>
      </typ:global-instance>
    </typ:assess-response>
  </SOAP-ENV:Body>

```

```
</SOAP-ENV:Envelope>
```

From the response above, we can see that the attribute *eligible_low_income_allowance* is unknown, if we look at the decision report we can see that the reason why the outcome is unknown is because the three attributes *claimant_income*, *claimant_public_housing_client* and *claimant_date_of_birth* are unknown. If we provide these attributes in the next request the outcome should become known.

Request with input

Note that the new request has values for *claimant_income*, *claimant_public_housing_client* and *claimant_date_of_birth*.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:assess-request>
      <typ:global-instance>
        <typ:attribute id="eligible_low_income_allowance" known-outcome-style-
e="value-only" unknown-outcome-style="base-attributes"/>
        <typ:attribute id="claimant_public_housing_client">
          <typ:boolean-val>true</typ:boolean-val>
        </typ:attribute>
        <typ:attribute id="claimant_income">
          <typ:number-val>13000</typ:number-val>
        </typ:attribute>
        <typ:attribute id="claimant_date_of_birth">
          <typ:date-val>1981-03-22</typ:date-val>
        </typ:attribute>
      </typ:global-instance>
    </typ:assess-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response with known outcome

The response returns with the outcome *eligible_low_income_allowance* known and true.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:assess-response>
      <typ:global-instance>
        <typ:attribute id="eligible_low_income_allowance" type="boolean"
inferred="true">
```

```

        <typ:boolean-val>true</typ:boolean-val>
      </typ:attribute>
      <typ:attribute id="claimant_income" type="currency">
        <typ:number-val>13000.0</typ:number-val>
      </typ:attribute>
      <typ:attribute id="claimant_date_of_birth" type="date">
        <typ:date-val>1981-03-22</typ:date-val>
      </typ:attribute>
      <typ:attribute id="claimant_public_housing_client" type="boolean">
        <typ:boolean-val>true</typ:boolean-val>
      </typ:attribute>
    </typ:global-instance>
  </typ:assess-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Get a list of the data needed to reach a decision when a relationship is involved

When the decision may need to go through currently unknown relationships you should set the `resolve-indecision-relationships` to true, in the config section of the Assess operation. If this is set to true, the Determinations Server will look past the relationship to determine what other attributes or relationships might be needed. If it is not set, the Determinations Server will only tell you the relationship which is unknown

Example

In this example we want to determine if the claimant is eligible for the teenage children allowance.

Request

The initial request contains an attribute-outcome for `eligible_teenage_allowance` but no data at all. The unknown outcome style is "decision-report". This will give us a full decision report when `eligible_teenage_allowance` is unknown. Because we are providing no data in the request, we expect it to be unknown.

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:assess-request>
      <typ:config>
        <typ:resolve-indecision-relationships>true</typ:resolve-indecision-
relationships>
      </typ:config>
      <typ:global-instance>
        <typ:attribute id="eligible_teenage_allowance" known-outcome-style-
e="value-only" unknown-outcome-style="decision-report"/>
      </typ:global-instance>
    </typ:assess-request>
  </soapenv:Body>
</soapenv:Envelope>

```

Response

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:assess-response>
      <typ:global-instance>
        <typ:attribute id="eligible_teenage_allowance" type="boolean" inferred-
d="false">
          <typ:unknown-val/>
          <typ:decision-report report-style="decision-report">
            <typ:attribute-node id="dn:0" entity-id="global" instance-id=-
="global" hypothetical-instance="false" attribute-id="eligible_teenage_allowance"
type="boolean" text="Is the claimant eligible for the teenage child allowance?"
inferred="false">
              <typ:unknown-val/>
              <typ:relationship-node id="dn:1" source-entity-id="global"
source-instance-id="global" hypothetical-instance="false" target-entity-id="child"
relationship-id="claimantschildren" state="unknown" inferred-
d="false"></typ:relationship-node>
              <typ:attribute-node id="dn:2" entity-id="child" instance-
id="hypothetical_child" hypothetical-instance="true" attribute-id="child_teenager"
type="boolean" text="Is the child a teenager?" inferred="false">
                <typ:unknown-val/>
                <typ:attribute-node id="dn:3" entity-id="child" instance-
id="hypothetical_child" hypothetical-instance="true" attribute-id="child_age" type-
e="number" text="The child's age is unknown." inferred="false">
                  <typ:unknown-val/>
                </typ:attribute-node>
              </typ:attribute-node>
            </typ:decision-report>
          </typ:attribute>
        </typ:global-instance>
      </typ:assess-response>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>

```

From the response above, we can see that the attribute *eligible_teenage_allowance* is unknown, if we look at the decision report we can see that the reason why the outcome is unknown is because the relationship *claimantschildren* is unknown. Because we set *resolve-indecision-relationships* to true, the Determinations Server is also able to tell us that for all children that are targets of the relationship, we will also need to know the child's age.

Request with input

The new request has the claimants children linked to the claimant via the relationship *claimantschildren*. Each of the children has

their age provided (*child_age* attribute).

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:assess-request>
      <typ:config>
        <typ:resolve-indecision-relationships>true</typ:resolve-indecision-
relationships>
      </typ:config>
      <typ:global-instance>
        <typ:attribute id="eligible_teenage_allowance" known-outcome-style-
e="value-only" unknown-outcome-style="decision-report"/>
        <typ:entity id="child">
          <typ:instance id="child1">
            <typ:attribute id="child_age">
              <typ:number-val>8</typ:number-val>
            </typ:attribute>
          </typ:instance>
          <typ:instance id="child2">
            <typ:attribute id="child_age">
              <typ:number-val>11</typ:number-val>
            </typ:attribute>
          </typ:instance>
          <typ:instance id="child3">
            <typ:attribute id="child_age">
              <typ:number-val>16</typ:number-val>
            </typ:attribute>
          </typ:instance>
        </typ:entity>
      </typ:global-instance>
    </typ:assess-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response with known outcome

With the input provided, we can see that the claimant is eligible for the teenage child allowance.

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:i18n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+0800</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
```

```
<typ:assess-response>
  <typ:global-instance>
    <typ:attribute id="eligible_teenage_allowance" type="boolean" inferred-
d="true">
      <typ:boolean-val>true</typ:boolean-val>
    </typ:attribute>
    <typ:entity id="child" inferred="false">
      <typ:instance id="child1">
        <typ:attribute id="child_age" type="number">
          <typ:number-val>8.0</typ:number-val>
        </typ:attribute>
      </typ:instance>
      <typ:instance id="child2">
        <typ:attribute id="child_age" type="number">
          <typ:number-val>11.0</typ:number-val>
        </typ:attribute>
      </typ:instance>
      <typ:instance id="child3">
        <typ:attribute id="child_age" type="number">
          <typ:number-val>16.0</typ:number-val>
        </typ:attribute>
      </typ:instance>
    </typ:entity>
  </typ:global-instance>
</typ:assess-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Find out the next screen that should be displayed to reach a decision

For information on finding out the next screen that should be displayed to reach a decision, refer to the [Determine an Attribute's value](#) topic.

Language, timezones and other localization concerns

The following describes how languages, timezones and locales are specified and interpreted in the Oracle Determinations Server.

WS-I18N - a brief introduction

As of sometime around 10.0 the Oracle Determinations Server allowed timezones and languages to be specified using the [WS-I18N](#) draft standard. 10.2 also relies on WS-I18N to communicate timezone and locale information, however this mechanism has been standardized for this release. The spec itself provides a means of specifying locale and timezone information in the SOAP header of a web services call. It looks something like this:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:typ="http://oracle.com/determinations/server/10.2/server/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1000</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <!-- contents... -->
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

where the 'locale' element specifies the locale and the 'tz' element specifies the timezone.

All responses will supply both locale and timezone information relating to the current request. However, whether or not either a locale and/or a timezone must be specified in a request will depend on the individual service itself, more information on which can be found below.

i18n:locale

Where applicable the locale element is used in the request to specify the language to be used in the rulebase session, as distinct from the session's region settings (see below). The exact behavior for each service is:

- **Server Service:** the locale is optional and if specified in the request, it will be ignored. The locale for the server service is always 'en-US'.
- **Assess Service:** the locale is optional, but if specified determines the language to use for the rulebase session. Any specified locale must match one of the available languages for that rulebase. If no locale is specified, the default rulebase language will be used.
- **Interview Service:** the locale is optional, but the language for a given interview session can be specified by using this option in a start session request. If a locale is specified in a start session request it must match one of the available languages for that rulebase. If no locale is specified in a start session request, the default rulebase language will be used. Any locale specified for a request using an existing session must match the locale used to create that session.

The list of available languages for a given rulebase can be discovered by doing a [ListRulebases](#) .

i18n:tz

The time zone element is used to indicate the time zone of the Determinations Server itself which can be determined by making a [GetServerInfo](#) request. Specifying a time zone is optional in all requests, however, if a time zone is specified in a request, it must match the server's time zone or else an error will be returned. In Java the i18n:tz element can either be specified as an Olsen ID or an RFC 822 offset, however in .NET it must be specified as an RFC 822 offset.

Other localization concerns

Error messages

Any Determinations Server error message (that is, SOAP Fault messages) will be returned in the 'en-US' locale and can not be localized. Formatting and displaying error messages is a presentation layer concern.

DateTime values

Any DateTime value specified in either an Assess or Interview service request without a timezone will be interpreted as being in the timezone of the current session. Any DateTime that explicitly provides timezone information, that timezone must match the session's timezone, otherwise an error will be returned. Under **no** circumstances will the Determinations Server attempt to convert from one timezone to another.

Date and Time Of Day values

Since the Determinations Engine considers Dates and Time of Day values to be absolute points in time, that is, they are not relative to any particular timezone, any timezone information provided will be ignored.

Date range restriction for DateTime and Date values

For both DateTime and Date values, values prior to 0001-01-02 or after 9999-09-09 are not supported. This means that the following will cause errors:

- specifying Date values outside the acceptable range, such as 9999-12-31 or 0001-01-01 (or DateTime values which include out-of-range dates)
- specifying a Date or DateTime value with a minus sign.

Attribute values

All attribute values in a Determinations Server request and responses are never localized and are specified in the 'culture neutral format'. The formats are:

Type	Format	Example
boolean	xsd:boolean - [true false] [1 0]	'true', 'false', '1', '0'
date	xsd:date - YYYY-MM-DD	2010-12-24
datetime	xsd:dateTime - YYYY-MM-DDThh:mm:ss plus an optional timezone specified as a UTC offset	2010-12-24T20:30:55, 2010-12-24T20:30:55+10:00

Type	Format	Example
time	xsd:time - hh:mm:ss	23:59:59
text	xsd:string	'Hello, World', '???'', '???' ? '????', 'Bonjour tout le monde', 'Hola a todos', '??, ?????'
number	xsd:decimal	'42', '0.42' '-0.00042', '+420000.42'

Formatters

Even though attribute values are never entered nor returned in a localized manner, there are still some instances where the formatted attribute value appears, namely when that value has been used in text substitution - for example, attribute text, decision report node text, question screen labels and captions. In this case, the determinations server will use a default formatter which will provide default formatting for the specified session locale.

The default behavior can be altered by using a custom formatter.

Oracle Web Determinations URL API

Oracle Web Determinations is a web application that must interpret the user's request in order to determine what action is required. In order to provide a high level of security and to prevent unauthorized changes to interview session data, it is not possible to manipulate the interview data through the URL.

The general structure of the URL follows the Representational State Transfer (REST) schema, where the action comprises the first part of the URL and the resource to be acted upon constituting the rest.

Note that the very first request to Oracle Web Determinations must contain the `user=<username>` query parameter in the URL if a particular user name is to be assigned to an Oracle Web Determinations session. Also note that the encoding to be used both in the URLs and the request and response content objects is UTF-8.

Basic and advanced URL structures

Click on the following topic links to display information about the basic and advanced URL structures. Please refer to the [Placeholder Definitions](#) topic below for further information about each placeholder.

Basic URL structures

Basic URL structures

Go to:

[Select Rulebase screen](#)

[Locale Select screen](#)

[Start an interview session](#)

[Go to the Summary screen](#)

[Go to the Default Data Review screen](#)

[Retrieve an image](#)

[Retrieve a resource](#)

Select Rulebase screen:

<http://<webserver-url>/<web-determinations app>/>

Description

This URL directs the user to the *Select Rulebase* screen. From here, the user is able to select a rulebase to run.

Alternate outcomes

The user may not be presented with the *Select Rulebase* screen depending on the rulebases and their locales available to the Oracle Web Determinations. This is because Web Determinations will automatically select the rulebase and locale if there is only one choice to select.

Conditions	Outcome
Expected - many rulebases available	Select Rulebase screen
Only one rulebase available - but rulebase has multiple locales	Locale Select screen (rulebase auto-selected)
Only one rulebase available, and rulebase has one locale	Summary screen (rulebase/locale auto-selected)

Example

<http://localhost:8080/web-determinations/>

Locale Select screen:

<http://<web-determinations url>/startsession/<rulebase>>

Description

This URL allows selection of the rulebase via the URL, therefore causing Oracle Web Determinations to bring the user to the *Locale Select* screen.

Alternate outcomes

The user may not be presented with the *Locale Select* screen depending on whether the selected rulebase has more than one locale to choose from. This is because Oracle Web Determinations will automatically select the locale if there is only one choice to select.

Conditions	Outcome
Expected - multiple locales available for current rulebase	Locale Select screen
Only one locale available for current rulebase	Summary Screen (of current rulebase)

Example

<http://localhost:8080/web-determinations/startsession/My+Rulebase/>

Start an interview session:

<http://<web-determinations url>/startsession/<rulebase>/<locale>/>

Description

This URL allows selection of the rulebase and locale via the URL, therefore causing Oracle Web Determinations to start a session using the rulebase and locale provided. This brings the user to the default first screen - the *Summary* screen.

Example

<http://localhost:8080/web-determinations/startsession/My+Rulebase/en-US/>

Go to the Summary screen:

<http://<web-determinations url>/screen/<rulebase>/<locale>/summary>

Description

In a current interview session, this URL goes to the *Summary* screen.

Note: this URL will not work if there is no current interview session.

Example

<http://localhost:8080/web-determinations/screen/My+Rulebase/en-US/summary>

Go to the Default Data Review screen:

<http://<web-determinations url>/screen/<rulebase>/<locale>/Default+Data+Review>

Description

In a current interview session, this URL goes to the *Data Review* screen.

Note: this URL will not work if there is no current interview session.

Example

<http://localhost:8080/web-determinations/screen/My+Rulebase/en-US/Default+Data+Review>

Retrieve an image:

<http://<web-determinations url>/images/<image-filename>>

Description

This is the URL structure for retrieving the specified image file inside the 'images' folder of the web application, using the <image-filename>.

A developer can add a new image into the 'images' folder, then use that new image by using the URL above to reference it.

Location of 'images' folder

The 'images' folder is located in [/<webapp root>/WEB-INF/classes/images](#), where <webapp root> is the root of the Web Determinations application; for example: [C:\Tomcat\webapps\web-determinations\WEB-INF\classes\images](#)

Example

http://localhost:8080/web-determinations/images/oralogo_small.gif

Retrieve a resource:

<http://<web-determinations url>/resources/<resource-filename>>

Description

This is the URL structure for retrieving a resource file inside the 'resources' folder of the web application, using the <resource-filename>.

A resource is classified as files that are needed by the Oracle Web Determinations as part of displaying the web pages, such as CSS files, Javascript files.

A developer can add a new resource file into the 'resources' folder, then use that new resource by using the URL above to reference it.

Location of 'resources' folder

The 'resources' folder is located in /<webapp root>/WEB-INF/classes/resources, where <webapp root> is the root of the Web Determinations application; for example: C:\Tomcat\webapps\web-determinations\WEB-INF\classes\resources

Example

<http://localhost:8080/web-determinations/resources/reset.css>

Advanced URL structures

Advanced URL structures

URL's with the 'startsession' as the action are the only URLs that can be used without a current interview session. All others require usage within the context of an interview session, and will return an error page if used outside of an interview session.

Go to:

Start an interview session with a goal

Investigate a goal

Start a new interview session and load session data

Start a new interview session, load and investigate a goal

Go to an Authored screen

Go to the Decision Report screen

Go to the Save As screen

Save current interview session

End current interview session

Restart current interview session

Retrieve a document

Retrieve a commentary

Advanced usage of Goal ID

Start an interview session with a goal:

`http://<web-determinations url>/startsession/<rulebase>/<locale>/<goal-id>`

Description

Starts an interview session for the specified rulebase, with the specified goal-id. The first screen of the goal investigation will be displayed to the user.

Example

`http://localhost:8080/web-determinations/startsession/My+Rulebase/en-US/Attribute~a1~global~global`

Investigate a goal:

`http://<web-determinations url>/investigate/<rulebase>/<locale>/<goal-id>`

Description

Within a current interview session of the same rulebase - this URL structure can be used to investigate a goal.

The next screen for the goal is displayed. It is possible that some session data already exists, therefore for Attribute goals the screen that is displayed to the user may not be the first screen for the goal.

This URL structure works similar to *Start an interview session with a Goal*. The difference is that this URL is used once a session is started, so that current session data is not lost (as starting a new session resets session data). This URL cannot be used if:

- there is no current interview session in progress.
- if the current interview session is of a different rulebase.

Example

`http://localhost:8080/web-determinations/investigate/My+Rulebase/en-US/Attribute~a1~global~global?`

Start a new interview session and load session data:

`http://<web-determinations url>/startsession/<rulebase>/<locale>/?caseID=<case-id>`

Description

Allows starting a new interview session with session data loaded from a specified case ID.

The screen displayed is the *Summary* screen. When the user selects a goal, the next screen is displayed.

This works the same as *Start an interview session* but also loads a 'case ID' at the same time so the user does not need to manually load it.

Example

<http://localhost:8080/web-determinations/startsession/My+Rulebase/en-US/?caseID=JohnSmith>

Start a new interview session, load and investigate a goal:

<http://<web-determinations url>/startsession/<rulebase>/<locale>/<goal-id>?caseID=<case-id>>

Description

This starts a new session for the specified rulebase, loads session data from the specified case ID, and starts investigation on the specified goal ID.

This is exactly the same as *Start a new interview session and load session data* but selects a goal to investigate - therefore the first screen the user sees is the 'next screen' of the goal as opposed to the *Summary* screen.

This is a useful and more direct URL for users that need to start an interview session for a goal with pre-seeded data (which is usually data already known about the user).

Example

<http://localhost:8080/web-determinations/startsession/My+Rulebase/en-US/Attribute~a1~global~global?caseID=JohnSmith>

Go to an Authored screen:

<http://<web-determinations url>/screen/<rulebase>/<locale>/<authored-screen-name>?postredirect=<screen-post-url>>

Description

Directs the user to the specified screen <authored-screen-name>. If the screen has a POST action, the postdirect value <screen-post-url> is used. The postdirect is necessary for question screens, as question screens need to post the form to a URL different to itself. The postdirect is not necessary for data review screens.

Note: this URL needs to be used within a current interview session.

Example

encoded <screen-name>	qs%24s1%40Interviews_Screens_xint%24global%24global
unencoded <screen-name>	qs\$s1@Interviews_Screens_xint\$global\$global

encoded <screen-post-url>	%2fscreen%2fControlling%2bScreen%2bOrder%2fen-US%2-fre- port%2524attr%2524eligible%2524global%2524global%2524Relevant%2524false%2524false
double- unen- coded <screen-post-url>	/screen/Controlling Screen Order/en-US/re- port\$attr\$eligible\$global\$global\$Relevant\$false\$false

http://localhost:8080/web-determinations/screen/My+Rulebase/en-US/qs%24s1%40Interviews_Screens_xint%24global%24global?postredirect=%2fscreen%2fMy%2bRulebase%2fen-US%2-freport%2524attr%2524eligible%2524global%2524global%2524Relevant%2524false%2524false

Go to the Decision Report screen:

http://<web-determinations url>/screen/<rulebase>/<locale>/<decision-report-values>

Description

Opens the Decision Report screen. The <decision-report-values> acts as settings that are used by Oracle Web Determinations to tweak the Decision Report display.

Note: this URL needs to be used within a current interview session.

Example

http://localhost:8080/web-determinations/screen/My+Rulebase/en-US/report%24attr%24a1%24global%24global%24Relevant%24false%24false

Go to the Save As screen:

http://<web-determinations url>/save/<rulebase>/<locale>/save-as?postredirect=<screen-post-url>

Description

Opens the *Save As* screen. The postredirect value is used on the *Save Successful* screen - the screen *after* saving the current interview session successfully.

Example

http://localhost:8080/web-determinations/save/My+Rulebase/en-US/save-as?postredirect=%2fscreen%2fMy%2bRulebase%2fen-US%2fsummary%3fcaseID%3d123

Save current interview session:

<http://<web-determinations url>/save/<rulebase>/<locale>/save-as?caseID=<case-id>&postredirect=<screen-post-url>>

Description

Saves the current interview session. This URL will only work if the current interview session has been saved, and saved to the <case-id> specified. Otherwise the error 'Invalid value '2' for parameter 'caseID' in request' will be displayed

The postredirect value is used on the *Save Successful* screen - the screen *after* saving the current interview session successfully.

Example

<http://localhost:8080/web-determinations/save/My+Rulebase/en-US?caseID=1&postredirect=%2fscreen%2fMy%2bRulebase%2fen-US%2fsummary%3fcaseID%3d1>

End current interview session:

<http://<web-determinations url>/endsession/<rulebase>/<locale>?cancelURI=<relative-screen-url>>

Description

Goes to the screen that confirms the *ending* of the current interview session. When the user confirms, the next screen will be the *Select Rulebase* screen for default installations of Oracle Web Determinations.

If the user cancels, Oracle Web Determinations directs the page to the value of <relative-screen-url>.

Example

<http://localhost:8080/web-determinations/endsession/My+Rulebase/en-US?cancelURI=%2fscreen%2fMy%2bRulebase%2fen-US%2fsummary>

Restart current interview session:

<http://<web-determinations url>/endsession/<rulebase>/<locale>?okURI=<relative-screen-url>&cancel-URI=<relative-screen-url>>

Description

Goes to the screen that confirms the *restarting* of the current interview session. When the user confirms, Oracle Web Determinations displays the value of <relative-screen-url> for *okURI*.

If the user cancels, Oracle Web Determinations directs the page to the value of <relative-screen-url> for *cancelURI*.

Example

<http://localhost:8080/web-determinations/endsession/My+Rulebase/en-US?okURI=%2fstartsession%2fMy%2bRulebase%2fen-US&cancelURI=%2fscreen%2fMy%2bRulebase%2fen-US%2fsummary>

Retrieve a document:

To generate sample xml:

<http://<web-determinations url>/document/<rulebase>/<locale>/<document public name or global name>/true>

To generate a document:

<http://<web-determinations url>/document/<rulebase>/<locale>/<document public name or global name>/false>

Description

Allows direct linking to a *Document*. The document must first be made available - that is, it has been added in the Oracle Policy Modeling Screen Authoring in the *Summary* screen as a *New Document*. The URL structure above is the same as the **Document** link added in the *Summary* screen, therefore the simplest way to create this URL is to copy the URL from the Document link. Alternatively, the ID can be found in the Screen xgen file (.exs).

Examples

sample xml:

<http://localhost:8080/web-determinations/document/My+Rulebase/en-US/dbdcddb6-7a5a-42a1-8743-3a05bb9ac5b8/true>

the document:

<http://localhost:8080/web-determinations/document/My+Rulebase/en-US/dbdcddb6-7a5a-42a1-8743-3a05bb9ac5b8/false>

Retrieve a commentary:

<http://<web-determinations url>/commentary/<rulebase>/<locale>?target=<commentary-target>>

Description

Opens commentary content for the commentary-target in the current page. Note that the value of the commentary-target is encoded.

Example

<http://localhost:8080/web-determinations/commentary/My+Rulebase/en-US?target=default>

http://localhost:8080/web-determinations/commentary/My+Rulebase/en-US?target=screen%2fs2%40Screens_screens_xint

http://localhost:8080/web-determinations/commentary/My+Rulebase/en-US?target=attribute%2fschool_name

Advanced usage of Goal ID

Attribute Goals - investigating a non-global attribute as a goal

As mentioned in the *Goal ID Format* section below, Attribute goals are almost always of the format of `Attribute~<AttributeID>~global~global`.

It is also noted that it is possible to use the attribute of an entity instance as a goal. While the need for this would be rare, it is possible to do with the URL API.

Note that before using an entity instance attribute as a goal:

- the entity instance must exist in the current session data. This is ideal for situations where the interview session is pre-seeded with data using the 'load' feature - it ensures that the entity instance exists. Otherwise if the instance does not exist, an error will occur
- Since the format for the Attribute goal requires the entity instance name, this makes the URL dynamic. It is important to note that when pre-seeding sessions via the 'load' feature - entity instances from the load feature will have a predictable entity instance name based on the value from the save file.

To create the Goal ID, simply follow the format: `Attribute~<AttributeID> <entity> <entity-instance-name>`.

By default, instance names are 'instance #X' from the Oracle Policy Modeling Screen Authoring, where X is a number. This can be modified so that a more meaningful text is prepended to 'X' ; for example: 'personX'.

Procedural Goals (Flow) - investigating a non-global Flow as a goal

Flows are linked to an entity. The main flow will usually be connected to the global entity, and from there the main flow can have sub-flows.

Sub-flows are very useful for collecting entity instances and their attributes. It allows collecting of entity instances through several screens and also contain sub-flow themselves to collect child entities.

It is possible to go through a Flow of a specific entity instance. This can be done by specifying the flow, entity, and entity instance in the Goal ID.

To create the Goal ID, simply follow the format: `Procedural~<flow-name>{<entity> <entity-instance-name>}`. Using this as a Goal ID will bring the user to the start of the 'Flow' for the specified entity instance name.

Placeholder definitions

The following is a table containing the definitions for the placeholders to be used in the URL structures (**Note:** The encoding to be used both in the URLs and the request and response content objects is UTF-8):

Placeholder	Description	Notes
<web-	The address of	For a Tomcat install the address of the webserver would be:

Place-holder	Description	Notes
server-url>	the webserver http://localhost:8080/	http://localhost:8080/
<web-determinations app>	The name of the Oracle Web Determinations webapp inside the webserver	The default Oracle Web Determinations webapp name is 'web-determinations'
<web-determinations url>	The address of the Oracle Web Determinations webapp	Usually <webserver-url> and <web-determinations app> combined; for example: http://localhost:8080/web-determinations/
<rule-base>	The rulebase to use	usually the normal name of the rulebase (for example, My Rulebase) and convert space to + (e.g. My+Rulebase)
<locale>	The locale to use, follows a strict format	en-GB, fr-FR, zh-CN. For a list of the locale codes that Oracle Policy Automation supports out of the box, see: Implement support for non-English language For more information about the format see: Locale Format
<goal-id>	The goal-id, follows a strict format	For more information about the format see: Goal ID Format Examples: <ul style="list-style-type: none"> • Attribute~a1~global~global • Procedural~MyFlow{global~global~}
<authored-screen-name>	The screen name, has URL encoded chars. Follows a format	For more information about the format see: Authored Screen Name Format Example: http://localhost:8080/web-determinations/screen/Controlling+Screen+Order/en-US/qs%24s1%40Interviews_Screens_xint%24-global%24global?postredirect=%2fscreen%2fControlling%2bScreen%2bOrder%2fen-US%2-freport%2524attr%2524eligible%2524global%2524global%2524Relevant%2524false%2524false%3f <ul style="list-style-type: none"> • qs%24s1%40Interviews_Screens_xint%24global%24global • dr%24My+Data+Review%24global%24global
<screen-post-url>	The POST URL	For more information about the format see Screen Post URL Format

Placeholder	Description	Notes
	to be used by the screen being invoked. Follows a format	<p>Example: http://localhost:8080/web-determinations/screen/Controlling+Screen+Order/en-US/qs%24s1%40Interviews_Screens_xint%24-global%24global?postredirect=%2fscreen%2fControlling%2bScreen%2bOrder%2fen-US%2-freport%2524attr%2524eligible%2524global%2524global%2524Relevant%2524false%2524false%3fuse%2fscreen%2fControlling%2bScreen%2bOrder%2fen-US%2-fre-port%2524attr%2524eligible%2524global%2524global%2524Relevant%2524false%2524false%3fuse</p>
<non-authored-screen-name>	Screen names for non-authored screens	<p>To see the values for the non-authored screen names see: Non-authored Screen values</p> <p>Examples:</p> <ul style="list-style-type: none"> • Default+Data+Review • summary • report%24attr%24a1%24global%24global%24Relevant%24false%24false
<document-id>	identifies the document to display. This ID is generated during the Build process in OPM.	<p>An example of a document ID is: 6e2d1953-3a8c-493e-a36e-c82ebbb45b9b</p> <p>This ID can be found in the Screen xgen file (.exs) in the rulebase output folder, or can be extracted from the Document link itself in the Summary screen.</p>
<commentary-target>	specifies the commentary-target, the value is encoded. Follows a format	<p>For more information about the format see: Commentary Target Format</p> <p>Example: %2fs2%40Screens_screens_xint (URL decoded - /s2@Screens_screens_xint)</p>

When looking at the placeholder descriptions that follow, use the following table as a handy reference for URL character encoding:

Character	Encoding
\$	%24
@	%40
%	%25
?	%3d

Character	Encoding
+	%2b
;	%3b
&	%26
&	%26amp%3b
(whitespace)	%20
/ (slash)	%2f
œ	%c5%93

Note: in cases when an entity instance identifier contains a forward slash (for example: "person/0-1"), use %c5%93 instead of %2f, so that url parsing will not be broken.

Click on any of the following topics for a description of each of the placeholders that requires a strict format:

Locale format

The locale format follows the 'xx-XX' format e.g. en-US.

- xx is the language code as per ISO 639-1 (http://en.wikipedia.org/wiki/List_of_ISO_639-1_codes).
- XX is the country/region code as per ISO 3166-1 (http://en.wikipedia.org/wiki/ISO_3166-1_alpha-2).

Goal-ID format

There are two types of Goal-IDs: *Attribute* and *Procedural (Flows)*.

Attribute Goals

Attribute goals can be broken up into 4 parts, separated by tilde (~). An example Attribute Goal: Attribute~a1~global~global.

1. Type - This is the type for the Goal ID. For Attribute goals, this is 'Attribute'
2. Attribute ID/Public Name - This is the public name of the Attribute goal in the project. This would be set by the rule author; for example, a1, eligible or automatically generated; for example: **b1@Rules_doc**.
3. Entity Name - The name of the entity the attribute belongs to. Since Attribute goals must belong to the global attribute, this is always 'global'.
4. Entity Instance Name - The name of the entity instance. Since the entity is always 'global', the entity instance is also always 'global'.

Attribute goals are almost always of the format Attribute~<AttributeID>~global~global. It is possible to investigate an entity instance attribute for non-global entities, which is covered in *Advanced usage of Goal ID*.

Procedural Goals (Flows)

Procedural Goals (Flows) can be broken up into 3 parts. An example Procedural Goal: Procedural~MyFlow{global~global~}.

1. Type - This is the type for the Goal ID. For Procedural goals, this is 'Procedural'.
2. Flow Name - This is the name of the Flow for the Procedural goal; for example: MyFlow.

3. Context - This is the text inside the curly braces. There are three components.

- i. The first one is the entity name; for example: global.
- ii. The second is the entity instance name; for example: global.

Note: for non-global entities, this is dynamic. See *Advanced definition of Procedural Goals - Advanced usage of Goal ID*.

- iii. The third is a status - the value is either 'complete' or 'pending'. Usually this is left out.

Simple way for obtaining Goal IDs

There is a simple way to obtain Goal IDs. It is done by using the Goal links in the *Summary* screen. By right-clicking on the Goal links, it is possible to inspect the full URL of the Goal links, and extract the Goal ID.

If the Attribute goal that you want to build the URL for is not in the *Summary* screen, simply add it in Oracle Policy Modeling then run the rulebase.

1. In Oracle Policy Modeling, open the rulebase and go to the *Screen* file (or create a screen file if it doesn't exist).
2. In the *Screen* file, open the *Summary* screen (or create one if it doesn't exist).
3. In the *Summary* screen:
 - i. if it is a Goal ID for an Attribute, click on 'New Goal' and set the attribute.
 - ii. if it is a Goal ID for a Flow, click on 'New Flow' and set the flow (you must have created the Flow already before this).
4. Save the project, and Build.
5. Run the rulebase in Oracle Web Determinations (or alternatively in 'Build and Run').
6. In the *Summary* screen of the rulebase, find the Goal link for the goal added in step 3 above.
7. Right-click on the link, select 'Properties'.
8. Inspect the whole URL, and extract the Goal ID based on the Attribute/Procedural Goal format.

Authored Screen Name format

Authored Screen Names are URL-encoded, so characters like \$ and @ is converted to %24 and %40 respectively.

There are two types for Authored Screen Names: *Question Screen* and *Data Review*.

Question screen

The screen name for Question screens can be broken down into 4 parts. It is separated by the URL encoding %24 ('\$' when unencoded); for example: `qs%24s1%40Interviews_Screens_xint%24global%24global`.

1. Type - this designates the type. The value for Question screens is 'qs'.
2. Screen ID/Public Name - the public name is used if the Screen Author has given a public name to the *Question* screen. Otherwise an auto-generated Screen ID is used. The Public Name of the screen is accessible from the Oracle Policy Modeling Screen Authoring, but the Screen ID must be extracted from the Screen xgen file (.exs file in the rulebase 'output' folder).
3. Entity Name - The name of the entity the *Question* screen belongs to; for example: 'global'.
4. Entity Instance Name - The name of the entity instance; for example: 'global'.
For non-global entities, this is a dynamic value and therefore should not be hard-coded.

Data Review screen

The screen name for *Data Review* screens can be broken down into four parts. It is separated by the URL encoding %24 ('\$' when unencoded); for example: `dr%24My+Data+Review%24global%24global`.

1. Type - this designates the type. The value for *Data Review* screens is 'dr'.
2. Name - the 'Name' value provided for the *Data Review* screen in the Oracle Policy Modeling Screen Authoring. Spaces are converted to '+' for example: My+Data+Review.
3. Entity Name - The name of the entity the *Data Review* screen belongs to. Usually 'global'.
4. Entity Instance Name - The name of the entity instance. Usually 'global'.

Screen Post URL format

The Screen Post URL is an extra parameter added to the URL of a *Question* screen.

The URL is used by the form of the *Question* screen for when the screen submits the form as a POST.

The Screen Post URL is essentially a relative URL to a screen (that is, without the `http://<web-determinations-url>`). Therefore the Screen Post URL template would look like:

- `/screen/<rulebase>/<locale>/<authored-screen-name>`
- `/screen/<rulebase>/<locale>/<non-authored-screen-name>`

The target screen can be an authored or un-authored screen. The whole Screen Post URL is also encoded, resulting in the screen-name part becoming double-encoded. Encoding the templates above converts to:

```
%2fscreen%2f<rulebase>%2f<locale>%2f<authored-screen-name>
%2fscreen%2f<rulebase>%2f<locale>%2f<non-authored-screen-name>
```

Below are 3 steps that demonstrate an un-encoded screen-name being encoded to double-encoding. The sample screen-name below is of a *Question* Screen

1. `qs$s1@Interviews_Screens_xint$global$global` (un-encoded).
2. `qs%24s1%40Interviews_Screens_xint%24global%24global` (encoded).
3. `qs%2524s1%2540Interviews_Screens_xint%2524global%2524global` (double-encoded).

From the above, note that because encoding always has a '%' sign, for double-encoding this is further encoded encoded to '%25'; for example, 'qs\$s1' is encoded to 'qs%24s1'. Encoding again changes 'qs%24s1' to 'qs%2524s1'.

Decision Report format

Decision Report screen can take in values; for example, which attribute to use as the top-level attribute for the *Decision Report* screen.

The Decision Report value can be broken down into eight parts. The format is separated by the %24 (which is '\$' in encoded format). Each part is mandatory; that is, there should always be eight parts.

For example: `report%24attr%24a1%24global%24global%24Relevant%24false%24false`.

1. screen type - this will always be 'report' since we are opening the *Decision Report* screen; for example, this can be 'summary' for the *Summary* screen.
2. report type - this is the target type of the report, either 'attr' (attribute) or 'rel' (relationship).

3. id/name - the identifier for the report type. This would be the attribute id/public name or the relationship name.
4. entity - the entity the attribute belongs to. For relationship - this would be the entity source of the relationship.
5. entity instance - the specific entity instance of the entity in step 4 above.
6. all or relevant - this value would be 'all' or 'relevant'. 'all' shows all decision nodes including unknown attributes. 'relevant' shows only the ones relevant to the goal.
7. ignore silent - do not display silent attributes.
8. ignore invisible - do not display invisible attributes.

Commentary Target format

There are three items that can have commentary in Oracle Web Determinations - the rulebase, the current screen, or a control. Below are the format of the value to use for each item:

Item	Value Format	Example Value	Example Full URL
rulebase	default	default	http://localhost:8080/web-determinations/commentary/My+Rulebase/en-us?target=default
screen	screen%2f<screen-id>	screen%2fs2%40Screens_screens_xint (screen/s2@Screens_screens_xint)	http://localhost:8080/web-determinations/commentary/My+Rulebase/en-us?target=screen%2fs2%40Screens_screens_xint
control	attribute%2f<attribute-id>	attribute%2fb1 (attribute/b1)	http://localhost:8080/web-determinations/commentary/My+Rulebase/en-us?

Configure Oracle Web Determinations

The following is intended to describe how to configure Oracle Web Determinations, describing the functions of each of the configuration properties, as well as common configuration patterns. The configuration file is typically located in the following directory:

WEB-INF\classes\configuration\config.properties

If you make any changes to the *config.properties* file, be sure to restart your application server, because the configuration is cached.

Note: UTF-8 encoding should always be used.

Configuration examples

Classpath-based loading (Java only)

The following is an example of the *config.properties* file, configuring the Oracle Policy Automation Web Determinations component to load everything from the classpath.

It is assumed that the resources are placed into folders called **configuration**, **images**, **properties**, **resources**, **rulebases** and **templates** under the WEB-INF/classes folder in the web application's deployment folder.

Note: This would work for both an exploded and unexploded WAR deployment.

```
#####  
# Deployment Properties  
# Screen template to use for the question screen  
screen.template =question_screen.vm  
# Debug or Release  
release.context =DEBUG  
# The locale to default to if we're displaying a screen that is not attached to a specific locale.  
default.locale =en-US  
#####  
# Rulebase Loading Properties  
# Load rulebases as resource streams, as opposed to loading them  
# from the file system.  
load.rulebase.as.resource =true  
# If rulebases are to be loaded as resource streams, this  
# property specified whether or not the Java classpath  
# is to be used to load the resources.  
load.rulebase.from.classpath =true  
rulebase.path =rulebases  
cache.loaded.rulebases =false  
#####  
# Resourcing Properties  
load.messages.as.resource =true
```

```

messages.path =configuration
cache.messages =false
load.images.as.resource =true
images.path =images
cache.images =false
load.resource.as.resource =true
resources.path =resources
load.properties.as.resource =true
properties.path =properties
cache.properties =false
load.templates.as.resource =true
templates.path =templates
cache.templates =false
#####
# Plugins Properties
plugin.libraries=

```

File System-based loading (Java and .NET)

The following is an example of the *config.properties* file, configuring the Oracle Policy Automation Web Determinations component to load everything from the file system.

It is assumed that you are running the Java version of the application and that the resources are placed into folders called **configuration**, **images**, **properties**, **resources**, **rulebases** and **templates** under the WEB-INF/classes folder in the web application's deployment folder.

Notes:

- The following example **will not work** under Java for application servers that do not explode the WAR (for example, WebLogic) - the only option in such cases is to use classpath-based resource loading.
- Under .NET, the paths to the resources must be changed to the relative paths under the application's directory (for example, 'bin/rulebases', 'bin/configuration' and so on).
- The **cache.loaded.rulebases=false** property, combined with the fact that we are loading the rulebases from the file system, will cause a directory watcher to be invoked on the rulebases directory. This allows rulebases to be dynamically loaded, removed and updated as the contents of the rulebases directory changes.

```

#####
# Deployment Properties
# Screen template to use for the question screen
screen.template =question_screen.vm
# Debug or Release
release.context =DEBUG
# The locale to default to if we're displaying a screen that is not attached to a specific locale.
default.locale =en-US

```

```

#####
# Rulebase Loading Properties
# Load rulebases as resource streams, as opposed to loading them
# from the file system.
load.rulebase.as.resource =false
# If rulebases are to be loaded as resource streams, this
# property specified whether or not the Java classpath
# is to be used to load the resources.
load.rulebase.from.classpath =false
rulebase.path =/WEB-INF/classes/rulebases
cache.loaded.rulebases =false
#####
# Resourcing Properties
load.messages.as.resource =false
messages.path =WEB-INF/classes/configuration
cache.messages =false
load.images.as.resource =false
images.path =WEB-INF/classes/images
cache.images =false
load.resource.as.resource =false
resources.path =WEB-INF/classes/resources
load.properties.as.resource =false
properties.path =WEB-INF/classes/properties
cache.properties =false
load.templates.as.resource =false
templates.path =WEB-INF/classes/templates
cache.templates =false
#####
# Plugins Properties
plugin.libraries=

```

Notes:

Before you commence any configuration of Web Determinations, it is important to note that an application will not function properly if the date, time, or date/time value, as formatted according to the output format, cannot be understood according to the input format.

On Java, this means that the configured output format must match one of the configured input formats, and if one of the input formats might possibly be ambiguous the version matching the output format must come first.

On .NET, this means that the output format must be a legal and unambiguous format according to Microsoft's .NET runtime library. (We don't configure input formats on .NET.)

If you do not do this, dates will not be correctly understood and could result in a fatal error.

Web Determinations configuration files

The following provides details of the configuration options provided in Web Determinations. All configuration files can be found at the following locations:

- `<webroot>/bin/configuration` in .NET
- `<webroot>/WEB-INF/classes/configuration` in Java

Some important points to note are:

- Properties are declared according to the Java property file standard; essentially they're a key value list of values.
- ',' is interpreted as a list separator and so will normally need to be escaped. To do this write '\,' instead of just ','.
- All files must be saved as UTF-8 text files.
- Property names and values are case-sensitive.
- If you have set up your application to cache properties, templates and so on, then you will need to restart your application server.
- Specifying formats in *application.properties* will always take priority over any default values; for example, if you specify a date time output format with seconds, then seconds will be displayed even if the "show seconds" option is unchecked for that attribute in Oracle Policy Modeling.

Go to:

[application.properties](#)

[appearance.properties](#)

[messages.<locale>.properties](#)

See also:

[Example: Configuration for Classpath-based loading \(Java\)](#)

[Example: Configuration for File System-based loading \(Java and .NET\)](#)

application.properties

These are the core application properties that are used to configure the application and are not available templates.

Click on the appropriate link:

General properties:

Name	Description
default.locale	this is the default locale to use for screens that are not tied to a locale such as the rulebase and locale selection screens. For information on locale formats see Input and Output Codes .
enable.debugger	set to true if you want to enable debugging from the debugger, otherwise false

Rulebase Loading properties:

Name	Description
load.rulebase.from.classpath	Java only . Set to true if you want the rulebases to be loaded from the classpath rather than the file system. This value must be set to true in order to deploy to Java application servers that do not explode the war file by default; for example, WebLogic
rulebase.path	the path to the directory to containing the rulebases. If load.rulebase.from.classpath = true then this path must be relative to the 'classes' directory. Otherwise the path can either be absolute or relative to the webroot.
cache.loaded.rulebases	If true rulebases will be cached and the server will required to be re-started in order to pick up rulebase changes. Otherwise it will operate in hot swapping mode which allows rulebase updates to be picked up without a server re-start. Note: if load.rulebase.from.classpath = true, this property will always effectively be true
enable.second.person	set to true to enable second person text substitution on rulebases that support it

isHTML validation:

Name	Description
screens.validate.html	Screens file content can include html authored by users in oracle policy modeling as static content. These options determine whether to scan the content at application start time and verify that the tags deployed in the rulebase are in the whitelist of allowable content.
screens.html.tags.whitelist	If screens.validate.html = true any tag not on this list will cause an exception to be thrown during rulebase loading and the rulebase will not be available. if additional tags are required they must be added to this list.

Resource Loading properties:

Name	Description
load.messages.as.resources	Java Only . Set to true if you want to load message properties files from the classpath. This value must be set to true in order to deploy to Java application servers that do not explode the war file by default; for example, WebLogic.
messages.path	the path to the directory containing the messages.properties. If load.messages.as.resources = true then this path must be relative to the 'classes' directory. Otherwise the path can either be absolute or relative to the webroot.

Name	Description
cache.messages	set to true if you want message files to be cached.
load.images.as.resources	Java Only. Set to true if you want to load images from the classpath. This value must be set to true in order to deploy to Java application servers that do not explode the war file by default; for example, WebLogic.
images.path	the path to the directory containing the images If load.images.as.resources = true then this path must be relative to the 'classes' directory. Otherwise the path can either be absolute or relative to the webroot.
cache.images	set to true if you want message files to be cached.
load.resources.as.resources	Java Only. Set to true if you want to resources images from the classpath. This value must be set to true in order to deploy to Java application servers that do not explode the war file by default; for example, WebLogic
resources.path	the path to the directory containing the resources. If load.resources.as.resources = true then this path must be relative to the 'classes' directory. Otherwise the path can either be absolute or relative to the webroot.
load.properties.as.resources	Java Only. Set to true if you want to load the appearance.properties from the classpath. This value must be set to true in order to deploy to Java application servers that do not explode the war file by default; for example, WebLogic.
properties.path	the path to the directory containing the appearance.properties If load.properties.as.resources = true then this path must be relative to the 'classes' directory. Otherwise the path can either be absolute or relative to the webroot.
cache.properties	set to true if you want appearance.properties files to be cached.
load.templates.as.resources	Java Only. Set to true if you want to load the templates from the classpath. This value must be set to true in order to deploy to Java application servers that do not explode the war file by default; for example, WebLogic.
templates.path	the path to the directory containing the velocity templates If load.templates.as.resources = true then this path must be relative to the 'classes' directory. Otherwise the path can either be absolute or relative to the webroot.
cache.templates	set to true if you want velocity templates files to be cached.
load.static.content.as.resource	Java Only. Set to true if you want to load the static contents from the classpath. This value must be set to true in order to deploy to Java application servers that do not explode the war file by default; for example, WebLogic.
static.content.path	the path to the directory containing the static files (that is, documents, html files). If load.-static.content.as.resources = true then this path must be relative to the 'classes' directory. Otherwise the path can either be absolute or relative to the webroot.

Plugin properties:

Name	Description
plugin.libraries	a ';' list of the fully qualified plugin classes to load. By default, all libraries in the plugin directory will be searched, however specifying a list will restrict the plugin loader to only loading the specified classes. On Weblogic the jar files containing the plugins must be explicitly listed here in order to loaded.

Data:

Name	Description
xds.file.path	<p>when using the default data adaptor, this is the base directory to which the data will be saved. Can either be relative to the web root or absolute.</p> <p>Note: If you wish to deploy a WAR unexploded, and you want to use the default XDS data adaptor, you will need to change this to point to an absolute path for which Web Determinations has read and write access.</p>

Input and output formats

Click on the appropriate link:

Default formatter

Web Determinations allows the accepted input and output format(s) of Date, DateTime, Currency and Number attributes to be specified on a per locale basis using its default formatter. Note that these default formatter properties are manually entered by the user.

Property	Description	Additional Remarks
currency-symbol	the currency symbol to use	
grouping-separators	a quoted string of allowable grouping separators for number and currency values; for example a string of "\, " would mean that numbers	This property applies on the Java platform only. .NET number formatters do not allow you to specify an input pattern but rather make a best attempt to parse the input according to the specifics of the locale (see http://www.dotnet-culture.net/)

Property	Description	Additional Remarks
	such as 1 000 000 and 1,000,000 are accepted but not 1.000.000	
decimal-separator	the decimal separator value to use; for example, specifying '.' as a separator would interpret the currency value \$10.25 as ten dollars and twenty five cents. If the decimal separator was a ',' then the same number would be written as \$10,25	this property applies on the Java platform only. .NET number formatters do not allow you to specify an input pattern but rather makes a best attempt to parse the input according to the specifics of the locale (see http://www.dotnet-culture.net/)
output-currency-format	The format in which all currency values should be displayed. This format will automatically become an accepted currency input format.	This string is expressed using the Java Decimal format convention (see http://docs.oracle.com/javase/1.5.0/docs/api/java/text/DecimalFormat.html). Note: the currency symbol <i>must</i> be defined in addition to the output currency format if using Web Determinations in .NET. Failure to specify the currency symbol will cause the error <i>"Cannot specify an output currency format without also specifying the currency symbol."</i> to appear in the log file.
input-currency-formats	This is the ',' delimited list of input formats for currency values.	This property applies on the Java platform only. .NET number formatters do not allow you to specify an input pattern but rather make a best attempt to parse the input according to the specifics of the locale (see http://www.dotnet-culture.net/). Like the output currency format, the format itself is expressed using the Java Decimal Format convention. Notes:

Property	Description	Additional Remarks
		<ul style="list-style-type: none"> the user may include the currency symbol in the currency formats, and if they do a second copy with not be added. If the user wants a split format like \$#,##0.00;(\$#) then they must set it themselves that way, and add a second copy without the currency symbol e.g. input-currency-formats = #\,##0.00;(#),\$\,##0.00;(\$#) the format strings themselves are not localized, that is the format should always use , as the grouping separator and . as the decimal separator (e.g. #,##0.0#) even if the user has specified a space for the grouping separator. # ##0.00 will not be understood as a valid format even though 1 234.56 is a valid number according to that format. leading # characters don't do anything. #####0.0##### is the same as #0.0# the grouping size is set by the last separator. So, #,##0.0# is the same as ###,###,###,##0.0# is the same as ####,##,##,##0.0# in separate negative groupings, only the prefix and suffix matter, so \$#,##0.0#;(\$#,##0.0#) is the same as \$#,##0.0#;(\$#) is the same as \$#,##0.0#;(\$#,#####000.000000#)
output-number-format	The format in which all number values should be displayed. This format will automatically become an accepted number input format.	This string is expressed using the Java Decimal format convention (see http://docs.oracle.com/javase/1.5.0/docs/api/java/text/DecimalFormat.html).
input-number-formats	The ',' delimited list of input formats for number values.	This property applies on the Java platform only. .NET number formatters do not allow you to specify an input pattern but rather make a best attempt to parse the input according to the specifics of the locale (see http://www.dotnet-culture.net/). This string is expressed using the Java Decimal format convention (see http://docs.oracle.com/javase/1.5.0/docs/api/java/text/DecimalFormat.html).
output-date-format	The format in which all date values should be displayed. This format will automatically become an	This string is expressed using the Java Simple Date Format convention (see http://docs.oracle.com/javase/1.5.0/docs/api/java/text/SimpleDateFormat.html). Note: the output date format is always treated as the first input date format.

Property	Description	Additional Remarks
	accepted date input format.	
input-date-formats	This is the ',' delimited list of allowable input formats for date values.	This string is expressed using the Java Simple Date Format convention (see http://docs.oracle.com/javase/1.5.0/docs/api/java/text/SimpleDateFormat.html). Notes: <ul style="list-style-type: none"> this format will also be used to parse the date portion of a date time input where the date time input style has been set to two text boxes if you want to allow both two digit year format and four digit year format, the two digit year formats must be specified before those with four digit years (e.g. d/M/yy, d-M-yy, dd/MM/yyyy, dd-MM-yyyy) the output date format is always treated as the last input date format, and so the order is important.
output-datetime-format	The format in which all DateTime values should be displayed. This format will automatically become an accepted date time input format.	This string is expressed using the Java Simple Date Format convention (see http://docs.oracle.com/javase/1.5.0/docs/api/java/text/SimpleDateFormat.html).
input-datetime-formats	This is the ',' delimited list of allowable input formats for DateTime values.	This string is expressed using the Java Simple Date Format convention (see http://docs.oracle.com/javase/1.5.0/docs/api/java/text/SimpleDateFormat.html).

Locale codes

Locale codes use the two letter ISO 693-1 code followed by the two letter ISO 3166-1 country code separated by a hyphen.

Out of the box we provide localizations for the following languages:

Language	Locale code
Arabic (Saudi Arabia)	ar
Chinese (Simplified)	zh-CN

Language	Locale code
Chinese (Traditional)	zh-HK
Czech	cs
Danish	da
Dutch	nl
English (American)	en
English (Great Britain)	en-GB
Finnish	fi
French (France)	fr
German (Germany)	de
Hebrew	he
Italian	it
Japanese	ja
Korean	ko
Polish	pl
Portuguese (Brazilian)	pt-BR
Portuguese (European)	pt-PT
Russian	ru
Spanish (Modern)	es
Swedish	sv
Thai	th
Turkish	tr

appearance.properties

These properties control the general appearance of the application. Properties in this file can be overridden on a per-locale basis, if you create a `appearance.<locale>.properties` file; for example, `appearance.en-US.properties`

Click on the appropriate link:

Debugging properties:

Name	Description
show-attribute-question-identifiers	if true, shows attribute ids for controls on question screens
show-status-bar	if true, shows the status bar containing information about the session being run
show-case-id	if true, shows the current case id in the status bar
show-rulebase	if true, shows the name of the rulebase in the status bar
show-rulebase-version	if true, shows the rulebase version in the status bar Note that if the rulebase did not have a rulebase version stamped into it when it was built then nothing will be displayed.
show-rulebase-build-time	if true, shows the rulebase time in the status bar. The rulebase build time will always be formatted as an ISO 8601 date time value.
show-policy-modeling-version	if true, shows the version of Oracle Policy Modeling used to build this rulebase in the status bar.
show-locale	if true, show session locale in the status bar
show-user-id	if true, shows the current user id in the status bar
warn-unsubmitted-pages	if true, warns the user if they try to navigate away from a screen without submitting data, to prevent the lose of data.

Progress bar options:

Name	Description
show-progress-bar	if true, shows the investigation progress bar
show-progress-stages	if true, shows the investigation progress stages

Commentary options:

Name	Description
opa-commentary-type	This property controls the commentary display and can have the following values: <ul style="list-style-type: none"> • <i>frameset</i> - commentary and the investigation will be run in frames of a frameset • <i>popup</i> - commentary will open in a separate window • <i>iframe</i> - commentary will open in an iframe • <i>none</i> - commentary will not be shown

Name	Description
	The default value is set to <i>frameset</i> . If an invalid value is provided, <i>opa-commentary-type</i> will be set as <i>none</i> and no commentary will be shown.
opa-commentary-iframe-style	Use this property to set the various attributes for the iframe in which the commentary is to be displayed.
iframe-border	Use this property to set the border of the commentary iframe; 0 refers to no border. This is useful in cases where setting the frame-border using the style attribute does not work (for example, in Internet Explorer). Note: the border can also be set using the opa-commentary-iframe-style.
frameset-top-target	the frameset target
commentary-target	the commentary target

Colors, fonts and images

Click on the appropriate link:

General and main form body:

Name	Description
body-background-color	the background color for the screen body
body-text-color	the text color of the screen body
display-fonts	the list of available display fonts
attribute-id-text-color	the text color of the attribute ids
mandatory-marker-text-color	the text color of the marker used to signify a control is mandatory
link-text-color	the text color of any links in the screen body
link-hover-text-color	the text color of any links in the screen body when the mouse hovers over them
decision-report-leaf-image	the image to use for leaf nodes in decision report controls
decision-report-collapse-image	the image to use for collapsed nodes in decision report controls
data-review-collapse-image	the image to use for collapsed nodes in decision report controls
question-width	the percentage width of the question text on a question screen
info-width	the percentage width of the question info on a question screen

Name	Description
answer-width	the percentage width of the input control for a question of the question screen
screen-min-width	the minimum width a screen can be before scrolling, defined in pixels
text-area-cols	the default number of columns for multi-line text inputs
text-area-rows	the default number of rows for multi-line text inputs
listbox-width	the width of listbox controls
text-control-width	the width of text controls

Header:

Name	Description
header-image	the image displayed in the header
header-alt	the alt text for the header image
header-title	the header title
header-text-color	the color of the header text
headers-font-weight	the header font weight
hide-header	if set to "true" the header will be hidden

Progress bar and progress stages:

Name	Description
progress-bar-border-color	the progress bar border colour
progress-bar-complete-color	the colour of the completed stages in the progress bar
progress-bar-incomplete-color	the colour of the uncompleted stages in the progress bar
progress-stages-background-color	the background colour of of the progress stages
progress-stages-border-color	the border colour of the progress stages
progress-stages-text-color	the text colour of the progress stages
status-bar-background-color	the background colour of the status bar
status-bar-text-color	the colour of the status bar text
status-bar-border-color	the colour of the status bar border

Menu:

Name	Description
menu-background-color	the background color of the menu bar
menu-text-color	the text color of the menu items
menu-border-color	the menu items border color
menu-background-image	the background image for the menu items
menu-link-text-color	the color of the menu item links
menu-link-text-hover-color	the color of the menu item links when the mouse hovers over them
menu-left-width	the percentage of the left hand portion of the menu
menu-right-width	the percentage width of the right hand portion of the menu
show-save	controls the visibility of the Save button on the Oracle Web Determination menu bar
show-save-as	controls the visibility of the Save-as button on the Oracle Web Determinations menu bar
show-load	controls the visibility of the Load button on the Oracle Web Determinations menu bar

Important note:

Turning off the visibility of items in the menu bar does not mean that the functionality is completely disabled; to ensure security, functionality must be disabled via the Data Adaptor.

Footer:

Name	Description
footer-border-color	the color of the footer border
footer-text-color	the color of the footer text
footer-background-color	the background color of the footer
footer-background-image	the image to use in the footer
hide-footer	if set to "true" the footer will be hidden

Form controls:

Name	Description
input-border-color	the border color of input control

Name	Description
input-text-color	the text color for input controls
input-background-color	the background color of input controls
input-disabled-border-color	the border color of disabled input controls
input-disabled-text-color	the text color of disabled input controls
input-disabled-background-color	the background color of disabled input controls
input-readonly-border-color	the border color of read only input controls
input-readonly-text-color	the color of the text in read only input controls
input-readonly-background-color	the background color of read only input controls
input-seperator-color	the input of the separator between input controls
button-border-color	the border color of any buttons
button-background-color	the button background color
button-background-image	the background images to use for any inputs

Messages:

Name	Description
messages-border-color	the border color for any messages
messages-text-color	the messages text color
messages-error-text-color	the text color of error messages
messages-warning-text-color	the text color of warning messages
messages-background-color	the background color of the messages

Decision report options:

Name	Description
decision-report-default-depth	Sets the display depth of decision reports; the default is a display depth of 2 levels. Setting it to 0 will display all levels.

message.<locale>.properties

Contains locale specific properties like error messages, text, data formats and so on; see the topic, [Localization](#).

Click on the appropriate link:

Message properties:

Name	Description	Available properties
boolean-true	text for true boolean values	
boolean-false	text for false boolean values	
boolean-uncertain	text for uncertain values	
boolean-unknown	text for unknown values	
AttributeValueTypeError	Indicates that the value submitted for the attribute is of the wrong data type; for example, attempting to set a number value to 'bob'	<ul style="list-style-type: none">attributeId - the id of the attributeentityInstanceIdentifier - the entity instance this error was raised inmessage - the generic error messagevalue - the value that caused the error
AttributeValueError	Generic error that is raised due to an error attempting to set an attribute value	<ul style="list-style-type: none">attributeId - the id of the attributeentityInstanceIdentifier - the entity instance this error was raised inmessage - the generic error messagevalue - the value that caused the error
MissingValueError	Raised when no value (or an uncertain value) has been submitted for a mandatory value	<ul style="list-style-type: none">attributeId - the id of the attributeentityInstanceIdentifier - the entity instance this error was raised inmessage - the generic error message
UnknownAttributeError	Raised when attempting to access an attribute instance that does not exist	<ul style="list-style-type: none">attributeId - the id of the attributeentityInstanceIdentifier - the entity instance this error was raised inmessage - the generic error message
UnknownEntityError	Raised when attempting to access an entity that does not exist in the rulebase	<ul style="list-style-type: none">entityId - the name of the entitymessage - the generic error message
UnknownEntityInstanceError	Raised when attempting to access an	<ul style="list-style-type: none">entityId - the name of the entity

Name	Description	Available properties
	instance of an entity that does not exist in the session	<ul style="list-style-type: none"> ○ instanceName - the name of the entity instance ○ message - the generic error message
DuplicateEntityInstanceError	Raised when attempting to create an entity instance with the same name as one that already exists	<ul style="list-style-type: none"> ○ entityId - the name of the entity ○ instanceName - the name of the entity instance ○ message - the generic error message
InvalidRelationshipInstanceError	Caused by the failure to set an entity instance in the session	<ul style="list-style-type: none"> ○ relationshipId - the name of the relationship ○ source - the source entity instance ○ target - the target entity instance ○ message - the generic error message
UnknownRelationshipError	Caused by attempting to access a relationship that does not exist in the rule-base	<ul style="list-style-type: none"> ○ relationshipId - the name of the relationship ○ message - the generic error message
UnknownRelationshipInstanceError	Caused by attempting to access an instance of a relationship that does not exist in the session	<ul style="list-style-type: none"> ○ relationshipId - the name of the relationship ○ source - the source entity instance ○ target - the target entity instance ○ message - the generic error message
InvalidValueChangeWarning	Raised when the submitted value of a read only or invisible control differs to the current value of that attribute in the session.	<ul style="list-style-type: none"> ○ attributeId - the id of the attribute ○ entityInstanceIdentifier - the entity instance this error was raised in ○ message - the generic warning message ○ value - the value that caused the error
GenericError	This is the fall-back generic error message for errors occurring during the submission of a screen that is displayed.	<ul style="list-style-type: none"> ○ message - the generic error message
ControlValueFormatError	Raised when the submitted value is in an invalid format for the specified control.	<ul style="list-style-type: none"> ○ message - the generic error message ○ value - the value submitted ○ controlId - the id of the control which contained the invalid value
FailedToLoadCaseError	Raised when the specified case could not be loaded.	<ul style="list-style-type: none"> ○ message - the generic error message ○ caseId - the Id of the case attempting

Name	Description	Available properties
		to be loaded
InvalidActionError	Raised when the submitted url action is invalid.	<ul style="list-style-type: none"> message - the generic error message action - the name of the invalid action
MissingResourceError	Raised when a required resource file is missing.	<ul style="list-style-type: none"> message - the generic error message fileName - the name of the missing file
NoActiveSessionError	Raised when attempting to perform an action that requires an active session but no session has been started.	<ul style="list-style-type: none"> message - the generic error message
NoResourcesForLocaleError	Raised when at least one of the required localized resources files (screen.-properties, styling.properties or messages.properties) can not be found for the specified locale.	<ul style="list-style-type: none"> message - the generic error message locale - the specified locale
NoSuchMessageError	Raised when attempting to display an error message in a given locale for which no localized message has been provided.	<ul style="list-style-type: none"> message - the generic error message msgKey - the message key locale - the locale
ResourceLoadError	Raised when the required resource file can be found but could not be loaded for some reason.	<ul style="list-style-type: none"> message - the generic error message fileName - the name of the resource file
TemplateParseError	Raised when the specified template file could not be parsed.	<ul style="list-style-type: none"> message - the generic error message templateName - the name of the template file
CaseIDNotValidError	Raised when the specified case id is invalid	<ul style="list-style-type: none"> message - the generic error message caseID - the invalid case id
CaseIDNotFound	Raised when no case can be loaded for the specified case id	<ul style="list-style-type: none"> message - the generic error message caseID - the invalid case id
TemplateParseError	Raised when the specified template file could not be parsed.	<ul style="list-style-type: none"> message - the generic error message templateName - the name of the template file
FormatValueError	Raised when the formatter is unable to format the given value.	<ul style="list-style-type: none"> message - the generic error message value - the value that could not be formatted
InputValueParseException	Raised when the formatter can not parse the specified value.	<ul style="list-style-type: none"> message - the generic error message value - the value that could not be

Name	Description	Available properties
		parsed
InvalidRequestError	Raised when the url request is invalid	<ul style="list-style-type: none"> message - the generic error message param - the invalid parameter value - the invalid value
InvalidScreenTemplateError	Raised when the specified template is invalid	<ul style="list-style-type: none"> message - the generic error message templateName - the name of the invalid template
DataSaveError	Raised when the case failed to save	

Months

The names of the months that are displayed in the drop down list for Date and DateTime controls where the input types are set to 'Year, Month and Day Edits' and 'Year, Month, Day, Hour Minute and Second Edits' respectively can also be localized through the following properties:

Month	Property Key
January	month-jan
February	month-feb
March	month-mar
April	month-apr
May	month-may
June	month-jun
July	month-jul
August	month-aug
September	month-sep
October	month-oct
November	month-nov
December	month-dec

Other text values not authored in Oracle Policy Modeling.

All other text values not authored in Oracle Policy Modeling can also be localized via the messages.properties file:

Property	Description
tree-leaf-alt	alt text for the leaf node icon in tree controls
tree-collapse-alt	alt text for the collapse node icon in tree controls
tree-expand-alt	alt text for the expand node icon in tree controls
data-review-no-controls	text for screens that appear on the data review screen, that have no child controls to display
decision-report-why-text	text for the decision report link on summary screens
decision-report-already-proven	text for the already proven nodes in a decision report
frame-interview-title	tooltip for the interview frame when displaying commentary using a frameset
frame-commentary-title	tooltip for the commentary frame when displaying commentary using a frameset
select-rulebase-screen-title	the text to display on the select rulebases screen
LocaleSelectionTitleInfoMessage	the text to display on the select locale screen
confirm-end-session-text	the message to display when ending a session
save-failed-title	the title of the save failed screen
save-failed-label	the text to display on the save failed screen
case-saved-label	the text to display on the screen confirming that the case saved
RestoreCaseLabelInfoMessage	the text to display on the restore saved file
SavedCasesAvailableForUserInfoMessage	the text to display when getting the list of available saved cases
NoSavedCasesAvailableForUserInfoMessage	the text to display when no saved cases are available to load
load-failed-title	the title of the load failed screen
load-case-title	the title of the load case screen
footer-left-text	the left footer text
footer-right-text	right footer text
delete-check-box-text	caption for the delete check box on the entity collect controls
delete-button-text	caption for the delete entity instance(s) button
add-button-text	caption for the add entity instances button
submit-button-text	caption for the submit button
ok-button-text	caption for the ok button
end-session-button-text	caption for the end session button

Property	Description
cancel-button-text	caption for the cancel button
save-case-label	caption for the case ID input on the save case screen
save-button-text	caption for the save button
continue-button-text	caption for the continue button
second-selection-alt	alt text for the seconds selection input for date time and time inputs
minute-selection-alt	alt text for the minute selection input for date time and time inputs
hour-selection-alt	alt text for the hour selection input for date time and time inputs
day-selection-alt	alt text for the day selection input for date and date time inputs
month-selection-alt	alt text for the month selection input for date and date time inputs
year-selection-alt	alt text for the year selection input for date and date time inputs
date-selection-alt	alt text for the date input for date time inputs
time-selection-alt	alt text for the time input for date time inputs
mandatory-text	the text to display next to mandatory fields
save-text	caption for the save button on the menu bar
save-as-text	caption for the save as button on the menu bar
load-text	caption for the load button on the menu bar
close-text	caption for the close button on the menu bar
restart-text	caption for the restart button on the menu bar
summary-text	caption for the summary screen button on the menu bar
data-review-text	caption for the data review button on the menu bar
case-id-text	caption for the case ID element in the status bar
rulebase-text	caption for the rulebase element in the status bar
rulebase-version-text	caption used when displaying the rulebase version in the status bar
rulebase-build-time-text	caption used when displaying the rulebase build time in the status bar
policy-modeling-version-text	caption used when displaying the Oracle Policy Modeling version used to build the rulebase in the status bar
locale-text	caption for the locale element in the status bar
user-id-text	caption for the user ID element in the status bar

Property	Description
auto-screen-title	screen title used for all automatic screens

IsHTML and Web Determinations customization

Web Determinations supports customization of the user interface displayed to users via the entry of HTML content for controls. In order to use this feature it's recommended that users have a basic working knowledge of HTML and web development. A good primer is available at <http://www.w3schools.com/html/> .

Entry of HTML tags in this manner has certain security implications and by default the modeling and web determinations applications limit the set of HTML tags that can be entered via a whitelist. If extra HTML tags are required then this is configurable in both applications.

Oracle Policy Modeling

This whitelist in modeling is configurable via **File > Project properties; Common Properties > General** and then in the Web Determinations section *HTML tags allowed in screen content* field. If a tag is used that is not in this comma separated list, then a build error will occur. The error reads "*The HTML tag 'tagname' is not an allowable HTML tag for use in a caption*"

The current set of tags allowed are any of the following "b,i,del,s,-div,p,span,pre,table,td,tr,ol,ul,li,blockquote,font,a,h1,h2,h3,h4,h5,h6,img,hr,br"

By default the set of tags includes tags that are known to not have security implications and can be safely displayed in Web Determinations. Notable absence is the 'script' tag which means javascript cannot be entered via the IsHTML option. If a tag is required that is not in the default set, it can be added by adding it to the list; for example, including the script tag would result in the following content "b,i,del,s,div,p,span,pre,table,td,tr,ol,ul,li,blockquote,font,a,h1,h2,h3,h4,h5,h6,img,hr,br,**script**"

Web Determinations

As Web Determinations is deployed as an application which can host many rulebases, it has certain security requirements and if the tag is not supported it will fail to load the rulebase. If an additional tag is required then the tag is also required to be added to the whitelist for the Web Determinations application.

This is configurable in the application.properties and the property 'screens.html.tags.whitelist' property. If html validation should be turned off in entirety it can be turned off by setting the 'screens.validate.html' property to false

By default it looks like:

```
# Screens file content can include html authored by users in oracle policy modelling as static content.  
# These options determine whether to scan the content at application start time and verify that the  
# tags deployed in the rulebase are in the whitelist of allowable content.  
screens.validate.html =true  
# any tag not on this list will cause an exception to be thrown during rulebase loading and the rulebase will not be available.  
# if additional tags are required they must be added to this list.  
screens.html.tags.whitelist =b;i;del;s;div;p;span;pre;table;td;tr;ol;ul;li;blockquote;font;a;h1;h2;h3;h4;h5;h6;img;hr;br
```

Velocity Templates Developer Guide

This Velocity Templates Developer Guide is intended as a guide for those wishing to write templates for rendering screens and controls into HTML. Ideally, no more knowledge besides HTML skills (along with the documentation provided here) will be needed to develop and customize templates.

Velocity Templating Language (VTL)

The markup language used to author the templates to be rendered using the Velocity and NVelocity templating engines is called the **Velocity Templating Language (VTL)**. A complete specification of this language can be found in the [Velocity User Guide](#); an essential reference for any velocity template developer.

Template Context

The set of variables and objects provided to, and referenced by templates is referred to as the **Template Context**. By default, this includes the following:

- The properties in the *screens.<locale>.properties* and *styling.<locale>.properties* files.
- The built screen model object, referenced as its default reference name *screen*.
- The context path of the application, referenced as the variable called *context-path*.
- The URL extension for the URLs on the particular platform (Java or .NET). This extension should be added to the end of all URLs generated by the templates (form actions, and so on). This variable is called *url-path-extension*.

Conventions and best practices

The following is a list of identified conventions and the best practices to employ when authoring or maintaining templates.

Separation of Model from View

Velocity was designed with the separation of Model from View in mind - templates should not build, manipulate or in any way interfere with the model to be rendered (in this case, the conceptual idea of a 'model' is represented by the Template Context).

While it may be possible to call methods that change the state of the model from the template code, do not do it. The results are likely to negatively impact other templates aggregated by the offending template, as well as lead to a possibly inconsistent model. It is likely that if you find yourself needing to change the objects in the template context provided to you from the templates themselves, then the template context is wrong and/or incomplete.

Match template hierarchy with screen model hierarchy

Velocity provides the ability to import a template into another template, basically in-lining it and providing it with the same template context as the parent template. This is very useful, as it maps nicely to the idea of Screen objects encapsulating ScreenControl objects.

This approach has a lot of benefits, with one of the most obvious being maintainability. If a problem is discovered with a template that renders a particular control, or the control object has been updated and the template needs to reflect this, the change may be made in one template and will carry on to all templates that import the control's template.

Extract common logic into a specific template

Following on from the previous point, the template import functionality can be leveraged to extract commonly used functionality into a separate template, which is then imported by templates as required.

This simplifies the job of less experienced or less technically-inclined template developers, since commonly used functionality which might otherwise be complex to implement, may be provided as a set of library functions (called 'macros' in VTL) in a specific template. By doing this, entire APIs and abstraction layers to the template context may be provided to template developers, in order to deliver an interface to the data model at the right level of complexity.

Do not use the '.' character in property names

It is common practice, especially in build scripts, to use the '.' character as a word-separator in property names. This becomes a problem, however, when the properties in various property files (*screens.<locale>.properties*, *styling.<locale>.properties*, and so on) are to be accessible from the VTL code implemented in templates. This is due to the way in which Velocity and NVelocity resolve references to variables that exist in the template context. For example, for the VTL fragment:

```
#{screen.title}
```

the first thing that the parser will try to do is look for the property *title* inside the object referenced by *screen*. Failing this, it will then look for the method *screen.getTitle()*.

Velocity does not allow the '.' character as a part of variable names, which is why it reads the above fragment as 'the property 'title' or the method 'getTitle()' on the object 'screen', **not as** 'the variable 'screen.title'.

According to the [Velocity User Guide](#), a variable placed into the template context, or one declared in the VTL, must conform to the following rules to be valid:

- A VTL identifier must start with an alphabetical character (a..z or A..Z).
- A VTL identifier is limited to the following types of characters: alphabetical (a..z or A..Z), numeric (0..9), hyphen ('-') and underscore ('_').

Examples

The following is a set of examples for performing commonly required operations and implementing commonly required logic in the templates:

Referencing the Screen model's methods in templates

This is accomplished using the screen's reference name as a VTL variable. Typically, this is *screen*, although custom logic may override this.

So, for example, the screen object is accessed using:

```
#{screen}
```

A public method exposed on the screen model (such as *getControls()*) may be accessed as:

```
#{screen.getControls() }
```

Notice that the method call looks exactly the same as the implementation language (in this case Java). This is because Velocity and NVelocity parse the code inside the `{}` tags as plain Java/.NET code. Note that in NVelocity, the method calls, object references and variable names are NOT case-sensitive. So,

```
#{screen.GetControls() }
```

and

```
#{screen.getControls() }
```

are parsed as exactly the same method calls on the same object, allowing templates written for the Java implementation of Oracle Web Determinations to be re-used for the .NET implementation. Note however that templates written for the .NET app **will not** work for the Java implementation if the capitalization on the method names is different. This is due to NVelocity being case-insensitive while Velocity expects the method names to be in the correct case.

Note also that the code inside the `{}` tags is resolved in its entirety, including object references and so on, just like in normal Java/.NET code. For example, if I wanted the ID of the first control in the screen's control list, I would do the following:

```
#{screen.getControls().get(0).getControlID() }
```

Loops, Conditional, Assignments and Delegation

There are many things going on in this example:

```
## VTL code to loop through the controls on the Screen object. We do this by rendering
## each control in turn, depending on its type.

## Grab the list of controls from the screen
#set( $controlList = ${screen.getControls()} )
#foreach( $control in $controlList )
    #if( ${control.getControlType().equals("LabelControl")} )
        #set( $labelControl = $control )
        #parse( "LabelControl.vm" )
    #end
    #if( ${control.getControlType().equals("QuestionControl")} )
        #set( $questionControl = $control )
        #parse( "QuestionControl.vm" )
    #end
#end
```

1. You can see that `##` is the single-line comment directive. The multi-line comment directive to start is `***` and to end is `*#`.
2. We have an example of an assignment operator that is quite self-explanatory. The only thing to watch out for with `#set` (as with all VTL directives) is to ensure that there is no white space between the `#` and the name of the directive. That is, `# set(` will fail whereas `#set(` will work. You will also notice that there is no need to allocate the new variable anywhere; simply having it set to something will allocate it and place it in scope. Obviously, there is no typing either.
3. The `#foreach` loop is actually the only loop provided by VTL. Its working should be obvious: `$controlList` is set to the list of controls that belong to the screen, and for each iteration of the loop, the `$control` variable provides a reference to the current element in this list.

4. The `#if` statement is a little more interesting in that the use of the `${ ... }` notation allows us to test a boolean statement generated by a call on the actual object represented by the `$control` variable. In this case, we check if the `$control` variable references a `LabelControl` object. If so, the first thing we do is set a new variable called `$labelControl` to reference the `$control` loop variable, since this object is expected by the `LabelControl.vm` template we are about to delegate to. Any variables that are in scope in the parent template are automatically added to the delegation template's template context. This is because the `#parse` directive just copies the contents of the referenced template, and parses the content in-line as VTL.
5. Do the same for the `QuestionControl`, and the loop is complete. Notice that the block directives (`#if` and `#foreach`), each have a corresponding `#end` directive.

Object typing

The Velocity Templating Language does not have typecast or type checking operators. Hence, the current way to get around this, at least for the screen controls, is to provide a method `getControlType()` on the `ScreenControl` objects, which returns a `String`. This method returns the (non-absolute) name of the screen control Class. This can be considered the replacement of the `instanceof` (in terms of Java) operator in VTL.

So, for example, if I wanted to check whether a control object (which I have set to the variable `$control`) is a `TextInputControl` object:

```
#if( ${control.getControlType().equals("TextInputControl")} )
    <!-- Do something here... -->
#end
```

Caveats and Gotchas

No null reference assignment

Once a variable has been assigned to some value using `#set`, it cannot then be removed from the context (set to `null`). If the RHS of a `#set` directive evaluates to a `null` reference, that `#set` directive will be skipped. For example, if the screen object's title field is set to `null` before it is added to the context, evaluation of the following snippet:

```
#set( $title = "dummy title" )
The screen's title is : $title

#set( $title = ${screen.getTitle()} )
The screen's title is : $title
```

will result in the following output:

```
The screen's title is : dummy title
```

```
The screen's title is : dummy title
```

Macro support in NVelocity

Macro support in NVelocity is currently quite limited; for example, a simple macro to set a text variable based on a single if condition worked under Velocity but failed under NVelocity. At present, it is probably wise not to rely on macros in NVelocity.

Template files

The overall concept of templates (at least those used to render screens and screen controls) is to have them closely match the hierarchical structure of the Screen and ScreenControl objects in the code. That is, one or more top-level templates are used to render the Screen objects, which then in turn delegate (using the VTL `#parse(<template name>)` directive) to render the controls on that screen.

In much the same way that controls aggregate other controls (for example, the QuestionControl object has a LabelControl and InputControl attached to it), these control templates can delegate to other control templates depending on what type of control is being rendered. Hence, the naming scheme for the control templates is obvious: the control represented by the **SomeControl** object in the code is rendered using the **SomeControl.vm** template.

It is the responsibility of higher-level templates to determine which bottom-level template to delegate to (that is, the one that actually outputs the HTML) to render a particular control. Hence, multiple bottom-level templates can be provided to render the controls - one for each different way a control can be shown.

Alternatively, this logic can be contained all within one bottom-level control. Hence, for example, if we sometimes wanted to render a boolean input control as a drop-down combo box and sometimes as a set of radio buttons, we could have two templates named **BooleanInputControl_DropDown.vm** and **BooleanInputControl_Radio.vm**. Or, we could have just the one template called **BooleanInputControl.vm** which would determine for itself which way the control should be rendered.

This is a lot of scope for personal preference in the naming and structuring of the templates - there are no hard-and-fast rules that are enforced either in the application or the template logic governing this. This, however, may change. The only thing to keep in mind is that if the logic in one template is changed, the logic in all templates that it interacts with (either in its own hierarchy or related templates) is suspect. It is recommended that changes to naming schema, hierarchy, logic and so on, in the templates is either done wholesale or not at all.

See also:

[Velocity Templates Developer Guide](#)

Oracle Web Determinations Template Reference Guide

The purpose of this guide is to provide information about the Oracle Web Determinations templates in relation to how they are used to construct screens during an interview. This is intended to assist potential template developers in navigating the Oracle Web Determinations templates, explaining the design concepts behind the file structure and template architecture implementations, and the data available to templates from the Interview Engine.

The structure is as follows:

- [Section 1 - Oracle web Determinations Interview Screens](#) - a summary of the Oracle Web Determinations interview screens encountered.
- [Section 2 - Oracle Web Determinations File Structure](#) - an explanation of the template directory structure.
- [Section 3 - Oracle Web Determinations Template Architecture](#) - a description of how the templates are constructed to form a screen.
- [Section 4 - Interview Engine Data available in Templates](#) - the variables from property files, and objects/arrays/variables from the Interview Engine during runtime that are available to templates.

Sections 1 and 2 provide basic and core knowledge of the Oracle Web Determinations screens and templates.

Section 3 contains information on what templates are used for each screen. The information from the first two sections is useful to know, because it helps to understand what the screens are, and where the templates are located.

Section 4 is useful for an understanding of what data/information from the Interview Engine is available to Oracle Web Determinations templates

Note:

Oracle Web Determinations templates rely on the Apache Velocity template engine. It is important to be familiar with the Velocity templating language; see the [Velocity Templates Developer Guide](#).

Section 1 - Oracle Web Determinations interview screens

Commonly encountered interview screens

When a user goes through an Oracle Web Determinations interview, he/she will most likely encounter the screens below. These screens provide the core functionality of Oracle Web Determinations - conducting an investigation and allowing traceability for goals that have been determined successfully

- Summary screen
- Question screen - Automatic
- Question screen - Authored
- Data Review
- Decision Report

The screens below may also be encountered depending on the Oracle Web Determinations setup, rulebase complexity, and other factors:

- Entity Collect
- Relationship Reference

- Load and Save Screen
- Error Page

List of interview screens

The following is a list of all of the screens available in Oracle Web Determinations, and a description of each one:

Screen name	Description
Rulebase Select	Displayed - when there are multiple rulebases and the user enters the Oracle Web Determinations webapp base URL.
Locale Select	Displayed - when the user selects a rulebase, and the rulebase has more than one locale.
Summary Screen	Displayed - when the user finishes investigating a goal, or the user clicks on the Summary link in the menu bar, or when the user starts a rulebase interview.
Data Review	Displayed - when the user clicks on the Data Review link in the menu bar.
Decision Report	Displayed - when the user clicks on the [Why] link next to a Goal that has been successfully investigated.
Question Screen - Automatic	Displayed - during an investigation of a Goal, an Automatic screen is displayed if an attribute does not belong to an authored screen and needs to be asked for the determination of the current Goal.
Question Screen - Authored	Displayed - during an investigation of a Goal or a Flow, attributes that need to be asked and belongs to an authored screen is displayed. The key difference between Automatic and Authored is that Authored usually has multiple attributes per screen, Automatic screens can only ask one attribute per screen.
Entity Collect	Displayed - if the rulebase has Entities that need to be collected for the current Goal, the interview will present an Entity Collect screen to gather Entity Instances. Can be Automatic or Authored question screen.
Relationship Reference	Displayed - if the rulebase has a many-to-many relationship between two Entities and the two Entities need to be collected for the current goal Can be Automatic or Authored question screen.
Load Screen	Displayed - if the user clicks on the Load button.
Save Screen	Displayed - if the user clicks on Save As , or in certain cir-

Screen name	Description
	cumstances the Save screen.
Load and Save Success	Displayed - when the user successfully saves or loads a session.
Close/Restart Session	Displayed - when the user attempts to close or restart the current session.
Error/Fail Page	Displayed - when there is a general error, or failure to perform a certain action such as save, load, restart or close.
Commentary popup	Displayed - when the Oracle Web Determinations has been setup so the commentary opens in a new window instead of a frame. Clicking on a commentary link opens this popup.

Section 2 - Oracle Web Determinations template file structure

The Oracle Web Determinations templates are located in the following filepath (the **template parent folder**):

<OWD webapp>/WEB-INF/classes/templates

As an example, for a local Tomcat install it would be:

C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\web-determinations\WEB-INF\classes\templates

Base/Main templates

Base templates are templates that have the <html> tag. Therefore they are never included by any other template (unless that template is simply a 'proxy').

The base templates are most or all of the templates that are located directly under the template parent folder.

Template directory structure explained

templates folder - the templates folder contains all the base template files, and also contain sub-folders that categorize the templates.

controls sub-folder - contains all the templates to render each control type. The name of the template maps to the classname of the **Control** object that depends on it (since the Interview Engine renders a screen by rendering **Control** objects inside the current **Screen** object). Note that some controls are used in screens other than the *Question* screen.

includes sub-folder - as the name implies, contains templates that are 'included' by all base templates. It contains Javascript includes and also page components that is displayed in most screens such as the header, footer, and bars.

investigation sub-folder - contains templates that are mostly used during 'investigation'; that is, *Question* screens.

session_control sub-folder - contains templates that are used for saving and loading a session. Some of the templates are 'proxy' templates in that they simply include a base template such as the 'generic_screen.vm'.

Directory and File Listing

Below is a listing of the folder and file structure under the template parent folder. The underlined text are folders and the bold text are **important templates**

- **templates**

- confirm_end_session_screen.vm
- **generic_screen.vm**
- error_screen.vm
- **frameset.vm**
- data_review_screen.vm
- **main.vm.css**
- locale_selection_screen.vm
- **question_screen.vm**
- start_screen.vm
- **summary_screen.vm**
- controls
 - BaseLinkControl.vm
 - BooleanInputControl.vm
 - ButtonControl.vm
 - ButtonGroup.vm
 - CheckBoxControl.vm
 - ContainmentRelationshipControl.vm
 - ControlList.vm
 - CurrencyInputControl.vm
 - DataReviewControl.vm
 - DateInputControl.vm
 - DateTimeInputControl.vm
 - DecisionReportControl.vm
 - DocumentControl.vm
 - EntityInstanceCollectGroup.vm
 - GoalControl.vm
 - LabelControl.vm
 - NonAttributeTextInputControl.vm
 - NumberInputControl.vm
 - ReferenceRelationshipControl.vm
 - TextInputControl.vm
 - TimeOfDayInputControl.vm

- includes
 - header.vm
 - menu-bar.vm
 - status-bar.vm
 - stages.vm
 - footer.vm
 - javascript-init.vm
 - javascript-utilities.vm
 - javascript-browser-workarounds.vm
 - global-velocity.vm
- investigation
 - error.vm
 - warning.vm
 - Dropdown-selection.vm
 - Radiobutton-selection.vm
 - Listbox-selection.vm
 - input-style-overrides.vm
 - Link.vm
 - screen-messages.vm
 - screen-title.vm
 - **form.vm**
 - controlMessages.vm
 - control-text.vm
 - identifier-text.vm
- session_control
 - list.vm
 - load_failed.vm
 - save_as.vm
 - save_failed.vm
 - saved.vm

Template descriptions

Below are descriptions for some of the templates that are important or would benefit from further explanation:

Template name	Description
generic_screen.vm	Template used for generic form-based screens such as the load or save as screens
frameset.vm	Used when commentary is available for the rulebase, and commentary is displayed as frames.
main.vm.css	Contains the CSS for each screen displayed (added inside the HTML head tag), added as a VTL reference \$css-text in base templates.
question_screen.vm	Used as the base template for any question screen (both authored and automatic)
investigation/form.vm	Template included by any base screen that will display a 'form'.
confirm_end_session_screen.vm	Template used by the Restart and Close (session) functions.
includes/javascript-init.vm	Contains one Javascript function that is called by each base template for any 'Javascript initialisation' procedures.
includes/javascript-browser-work-arounds.vm	Contains Javascript functions that is solely for browser workarounds.
includes/global-velocity.vm	Included by every base template, useful for setting Velocity variables that should be available in all templates.
investigation/identifier-text.vm	Displays the attribute identifier next to the Control label of an attribute; for example, 'applicant_age'.
investigation/input-style-override.vm	Implements the CSS Class and CSS Style 'override' from the Oracle Policy Modeling Screen Authoring. Included by many control templates.
investigation/Link.vm	Used for various non-commentary links such as the link for documents (in the <i>Summary</i> screen) and the link in Load/Save Successful.
investigation/control-text.vm	Displays the label of an input control depending on commentary availability and setting (setting - to frame or popup window).
investigation/error.vm and investigation/warning.vm	Used by the screen-messages.vm and controlMessages.vm to display warning and error messages during an interview.
controls/ButtonGroup.vm	Used in screens that have groups of buttons, such as the Save and Cancel buttons in the <i>Save As</i> screen.

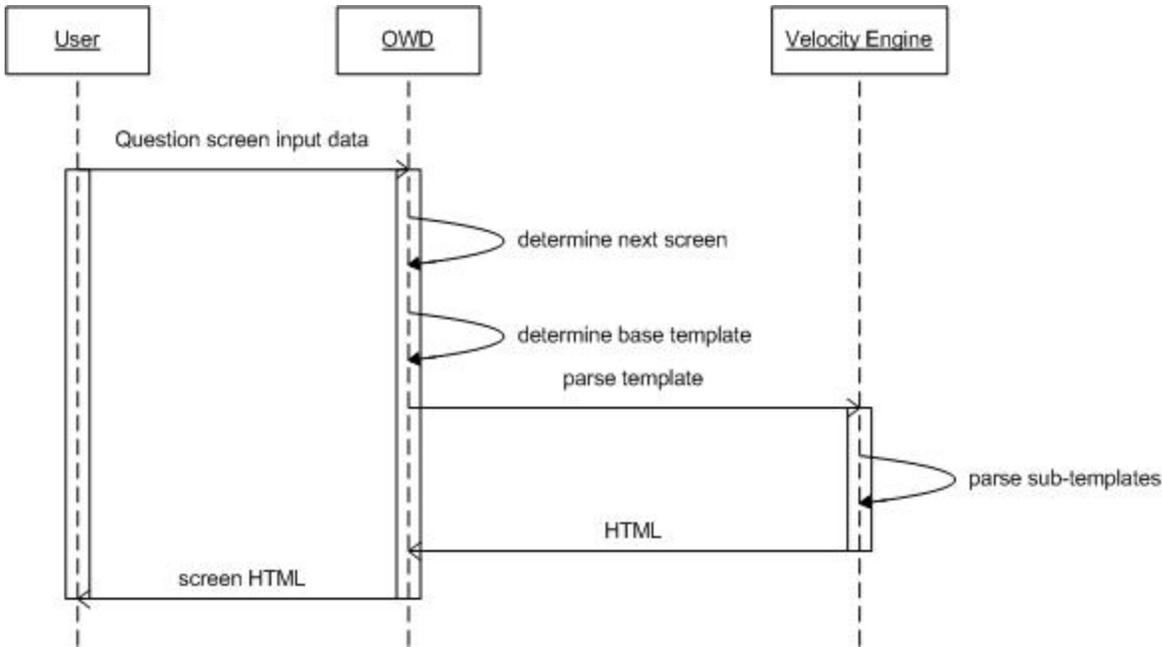
Section 3 - Oracle Web Determinations template architecture

This section describes how the template files are used by the Templating Engine to build an *Interview* screen.

The Oracle Web Determinations template architecture explained

Oracle Web Determinations templates rely on one of the base templates as the 'skeleton' of a screen; for example, the *Summary* screen relies on the **summary_screen.vm** template as it's skeleton/foundation.

The following is a sequence diagram that demonstrates at a high level how Oracle Web Determinations decides which 'base' template' to use for the next screen:



To get a better understanding of how Oracle Web Determinations generates screens using templates, here is a run-through of a sample scenario and the process that Oracle Web Determinations goes through to generate the HTML for the screen using templates. The sample scenario starts at the point where a user answers the last *Question* screen with input data that completes the goal investigation, thus triggering Oracle Web Determinations to return the *Summary* screen as the next screen:

1. The user provides the input for the current *Question* screen and submits.
2. Oracle Web Determinations receives the input data, and determines that the goal has been completed; as a result, the next screen to be displayed is the *Summary* screen.
3. The *Summary* screen (in Object form) inside Oracle Web Determinations, is mapped to the **summary_screen.vm** base template. Oracle Web Determinations uses the **summary_screen.vm** base template as the skeleton.
4. Oracle Web Determinations processes the **summary_screen.vm** template through the Velocity Template Engine to turn it into HTML; Velocity variables and (sub)templates to be parsed and attached are processed to HTML text.
5. The *Summary* screen HTML is generated, and Oracle Web Determinations returns it via HTTP response to the user.

From the above scenario, steps 3 and 4 are the key steps. In these two steps, Oracle Web Determinations has determined the base template file for the next screen, and processes it through the Velocity Template engine to parse it from the template format to a full HTML format.

It should also be noted that Oracle Web Determinations base templates are really made up of sub-templates (which themselves also use sub-templates). Having sub-templates promotes re-use, so that the same component for different screen elements are reused and can be changed from one sub-template file; for example, the header.

Base template and sub template - how it is processed to HTML

This section explains how a base template and its sub-templates hierarchy is parsed by the Velocity Engine to generate the HTML form.

For Oracle Web Determinations, the Velocity Engine provides the following functionality:

- Allows templates to add other templates into itself (using the `#parse` directive), thus promoting reuse and easy template readability.
- Allows access to configuration data from the properties files; for example, `appearance.properties`.
- Allows access to data in the Interview Engine such as session data, or the **Screen** object.

When the Velocity Engine goes through the base template, it processes the VTL directives of the template.

- References that are not with a directive are printed out as text; for example, `#{screen.getTitle}`.
- Some references (such as configuration values from 'properties' files) are used in VTL condition directives such as `#if` or `#for` loop to control logic.
- For a `#parse` directive, the Velocity Engine finds the sub-template indicated in the directive. That sub-template is then processed in the same manner. This can be recursive, in that if the sub-template also has `#parse` directives, its sub-templates are parsed too.

Example - start_screen.vm (Rulebase Selection screen)

It's much simpler to understand how a base template is processed to generate HTML by demonstrating it using a sample base template. This example provides a run-through of the **start_screen.vm** base template.

Sub-template parsing

Analyze the **start_screen.vm**, and its sub-templates for `#parse` statements; for example, `'#parse("includes/header.vm")'`

After analyzing the source for the **start_screen.vm** templates and all its other sub-templates, it is evident that the base template has one level of sub-templates; that is:

- start_screen.vm
 - includes/global-velocity.vm
 - includes/header.vm
 - includes/menu-bar.vm
 - includes/footer.vm

Velocity references as text

In the **start_screen.vm** template, the following code demonstrates Velocity references used as both HTML metadata (locale, pagedirString) and HTML display data (header-title):

```
<html lang="{locale}" dir="{pagedirString}">
  <head>
    <title>{header-title}</title>
```

Velocity references in conditional directives

From the **start_screen.vm** template, the following code demonstrates a Velocity reference (`$rulebases`) used in a **for** loop. The `$rulebases` reference is using a Java List that has been exposed by the Interview Engine to this template:

```
<div class="column">
  <ul>
    #foreach($rulebase in $rulebases)
      <li><a name="{rulebase.key}" href="{rulebase.value}">{rulebase.key}</a></li>
    #if($velocityCount % 23 == 0)
      </ul>
    </div>
    <div class="column">
      <ul>
    #end
  #end
</ul>
</div>
```

From the **includes/header.vm**, the code below demonstrates a Velocity reference (from **appearance.properties** file) being used in an **if** condition:

```
#if( ${show-application-name} == "true" )
  <p class="application-name">{application-name}</p>
#end
```

With the above explanations, let's follow a detailed step-through for generating HTML from **start_screen.vm**:

1. Oracle Web Determinations calls the Velocity Engine to transform the **start_screen.vm** to HTML.
2. The Velocity Engine starts to process **start_screen.vm**, by doing the following:
 - i. Parses '**includes/global-velocity.vm**' template.
 - ii. Processes some Velocity conditional statements.
 - iii. Starts the HTML page itself with the `<html>` and `<head>` start tags.
 - iv. Adds CSS text, then `<body>` start tag.
 - v. Parses the header - **includes/header.vm** into HTML; after parsing, the HTML is appended to the current HTML page (**start_screen.vm**) containing the parse directive.
 - vi. Parses the menu bar - **includes/menu-bar.vm** into HTML; after parsing, the HTML is appended to the current HTML page (**start_screen.vm**) containing the parse directive.
 - vii. HTML content area; process Velocity for loop to generate HTML code for the Rulebase links.
 - viii. Parses the footer - **includes/footer.vm** into HTML; after parsing, the HTML is appended to the current HTML page (**start_screen.vm**) containing the parse directive.
3. The Velocity Engine completes parsing the **start_screen.vm** to the completed HTML page.

Add new templates

Adding new templates is useful to ensure that template modifications are maintainable; this is particularly useful when making complex template modifications. Adding and using new template/s is done as follows:

1. Create the new template file (with file extension of **.vm**).
2. Add HTML/template code to the new template.
3. Use the new (sub)template file by referencing it with the **#parse** directive.
4. Restart Oracle Web Determinations and test to ensure that the new template is being referenced and used correctly.

Notes:

- When referencing the file, the root of the filepath is the parent template folder.
- Sub-templates usually belong to one of the subfolders under the parent template; create a new subfolder if necessary.

Important screens - templates used

The following table contains a useful reference to the important screens and their base template/sub-template hierarchies:

** = may or may not be parsed depending on runtime values*

Screen name	Screen template hierarchy
Summary Screen	<ul style="list-style-type: none">• summary_screen.vm<ul style="list-style-type: none">◦ includes/global-velocity.vm◦ includes/header.vm◦ includes/menu-bar.vm◦ includes/status-bar.vm◦ investigation/screen-title.vm◦ <i>for loop - controls</i><ul style="list-style-type: none">■ controls/[control type].vm◦ includes/footer.vm
Data Review	<ul style="list-style-type: none">• data_review_screen.vm<ul style="list-style-type: none">◦ includes/global-velocity.vm◦ includes/javascript-utilities.vm◦ includes/header.vm◦ includes/menu-bar.vm◦ includes/status-bar.vm◦ investigation/screen-messages.vm<ul style="list-style-type: none">■ includes/warning.vm*■ includes/error.vm*◦ investigation/screen-title.vm

Screen name	Screen template hierarchy
	<ul style="list-style-type: none"> ○ <i>for loop - controls</i> <ul style="list-style-type: none"> ■ controls/[control type].vm <ul style="list-style-type: none"> ○ uses DataReviewControl.vm ○ includes/footer.vm
Decision Report	<ul style="list-style-type: none"> ● question_screen.vm <ul style="list-style-type: none"> ○ includes/global-velocity.vm ○ includes/javascript-init.vm ○ includes/javascript-browser-workarounds.vm ○ includes/javascript-utilities.vm ○ includes/header.vm ○ includes/menu-bar.vm ○ includes/status-bar.vm ○ includes/stages.vm ○ investigation/screen-title.vm ○ investigation/screen-messages.vm <ul style="list-style-type: none"> ■ includes/warning.vm* ■ includes/error.vm* ○ investigation/form.vm <ul style="list-style-type: none"> ■ <i>for loop - controls</i> <ul style="list-style-type: none"> ○ uses DecisionReportControl.vm ○ includes/footer.vm
Question Screen - Automatic	<ul style="list-style-type: none"> ● question_screen.vm <ul style="list-style-type: none"> ○ includes/global-velocity.vm ○ includes/javascript-init.vm ○ includes/javascript-browser-workarounds.vm ○ includes/javascript-utilities.vm ○ includes/header.vm ○ includes/menu-bar.vm ○ includes/status-bar.vm ○ includes/stages.vm ○ investigation/screen-title.vm ○ investigation/screen-messages.vm <ul style="list-style-type: none"> ■ includes/warning.vm*

Screen name	Screen template hierarchy
	<ul style="list-style-type: none"> <ul style="list-style-type: none"> ■ includes/error.vm* ○ investigation/form.vm <ul style="list-style-type: none"> ■ <i>for loop - controls</i> <ul style="list-style-type: none"> ○ controls/[control type].vm ○ includes/footer.vm
Question Screen - Authored	<ul style="list-style-type: none"> • question_screen.vm <ul style="list-style-type: none"> ○ includes/global-velocity.vm ○ includes/javascript-init.vm ○ includes/javascript-browser-workarounds.vm ○ includes/javascript-utilities.vm ○ includes/header.vm ○ includes/menu-bar.vm ○ includes/status-bar.vm ○ includes/stages.vm ○ investigation/screen-title.vm ○ investigation/screen-messages.vm <ul style="list-style-type: none"> ■ includes/warning.vm* ■ includes/error.vm* ○ investigation/form.vm <ul style="list-style-type: none"> ■ <i>for loop - controls</i> <ul style="list-style-type: none"> ○ controls/[control type].vm ○ includes/footer.vm

Screens and Base Template mapping

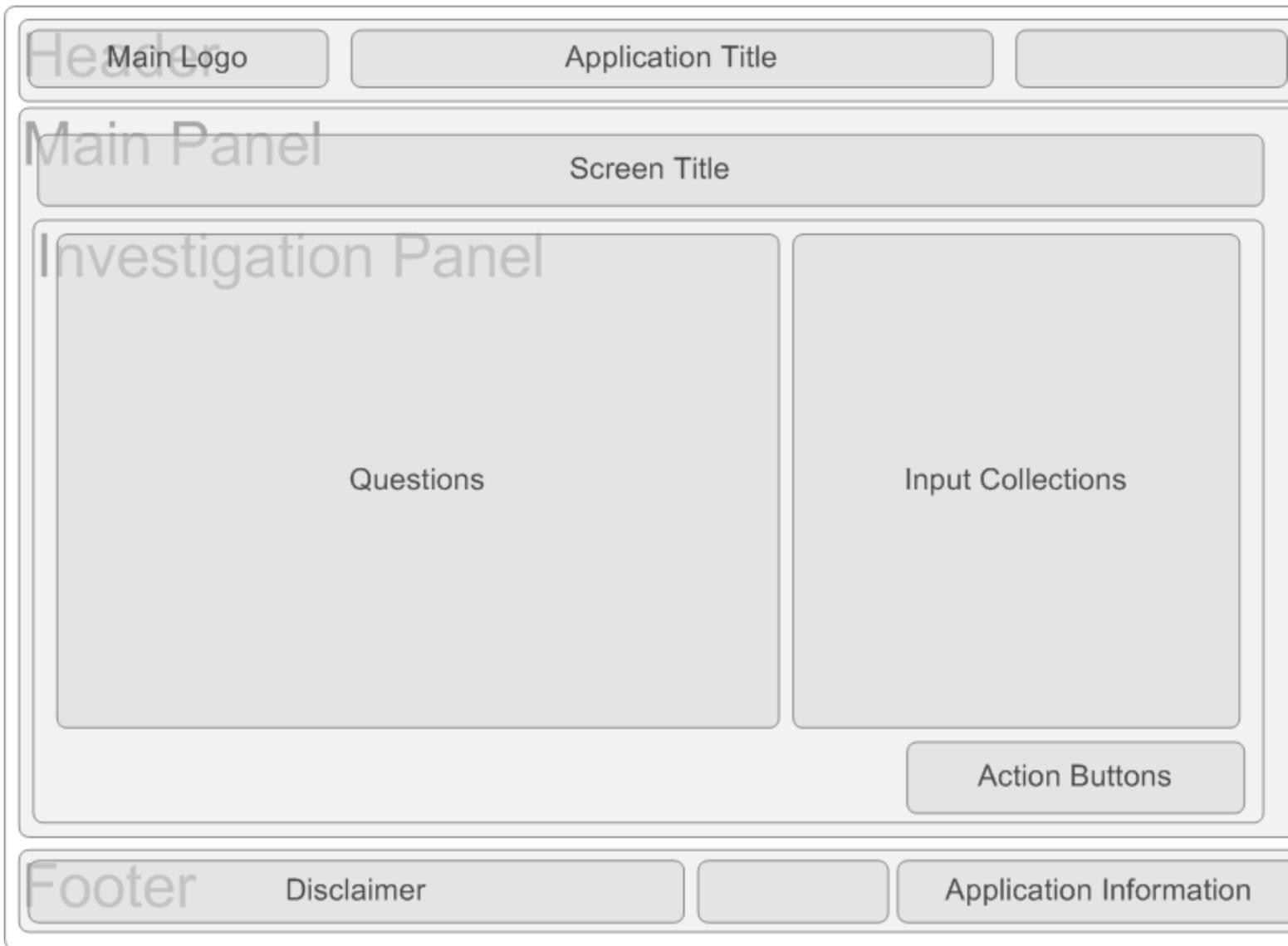
Maps all the screens to a base template as follows:

Screen name	Base template
Rulebase Select	start_screen.vm
Locale Select	locale_selection_screen.vm
Summary Screen	summary_screen.vm
Data Review	data_review_screen.vm
Decision Report	question_screen.vm

Screen name	Base template
Question Screen - Automatic	question_screen.vm
Question Screen - Authored	question_screen.vm
Load Screen	generic_screen.vm (via session_control/list.vm)
Save Screen	generic_screen.vm (via session_control/save_as.vm)
Load and Save Success	generic_screen.vm
Close/Restart Session	confirm_end_session.vm
Error/Fail Page	error_screen.vm
Question Screen with Commentary	frameset.vm

Default page structure and core layout for the Question screen

The following diagram illustrates the default page structure and core layout for the *Question* screen:



Section 4 - Interview Engine Data available in Templates

The Velocity templates in Oracle Web Determinations have access to variables from property files, and objects/arrays/variables from the Interview Engine during runtime.

Property files

Oracle Web Determinations templates have access to the values of property files that are loaded for the template. Since each property is a key/value pair, Oracle Web Determinations templates simply refer to the property key to access the template.

The following are the property files that each Oracle Web Determinations template has access to during runtime:

- **appearance.properties** - controls many of the display elements in Oracle Web Determinations, providing easily accessible configuration and localization.

- Show or hide sections or elements in Oracle Web Determinations, done by evaluating the property in an 'if condition' as true/false.
 - Provide easy customization to CSS and HTML attributes, done by directly outputting the property value on the Velocity variable tags.
- **message.xx-XX.properties** - controls localization of displayed text such as error messages and button text labels. The xx-XX refers to the locale value, so that the locale selection for an interview uses the correct localization property file

Extending Property files for template customizations

It is possible to add properties to the property files, so that configuration settings for new template customizations stay in the property files. It is very simple to add new properties as follows:

1. Add the property to the appropriate property files; each property file has a certain purpose, so it is a good idea to do this for maintainability.
2. Use the property in the Oracle Web Determinations templates.
3. Restart Oracle Web Determinations, and test to ensure that the new property or properties are working.

Interview Engine

The Interview Engine exposes many useful objects and variables to the Oracle Web Determinations templates; for example, it is possible to get the title of the current screen, or instance data of a certain entity's attribute, or the rulebase locale.

Application constants

The Interview Engine exposes various properties that are accessible directly. These properties are application constants that are handy to have direct access to from any template. The following table contains the application constants from the Interview Engine:

Property name	Description
user-id	<ul style="list-style-type: none"> • The user id of the current interview session. • Default is <i>guest</i>.
locale	The locale of the current interview session. If the user has not selected a rulebase or locale, this uses the default locale in <i>application.properties</i> .
image-resource-request	<ul style="list-style-type: none"> • Used as part of the URI to request image resources in Oracle Web Determinations, together with <i>context-root-path</i>; for example, <code>\${context-root-path}\${image-resource-request}/image_name.jpg</code>. <p>Note that this does not map to the folder name; for example, <code><OWD>/WEB-INF/classes/images</code>. The setting of where Oracle Web Determinations should pick up image resources to service image requests is in <i>application.properties</i>.</p> <ul style="list-style-type: none"> • Default is <i>/images</i>.

Property name	Description
context-root-path	The relative URI path to access the Oracle Web Determinations webapp from the webserver address; for example, if Oracle Web Determinations was set up in a Tomcat environment as 'web-determinations' webapp, the context-root-path is <code>/web-determinations</code> .
resource-request	<ul style="list-style-type: none"> Used as part of the URI to request non-image resources in Oracle Web Determinations, together with <code>context-root-path</code>; for example, <code>\${context-root-path}\${resource-request}/reset.css</code>. Similar to image-resource-request; note that the value of this property does not map to a folder name, but instead specifies the 'action' of the REST URL request. Default is <code>/resources</code>.
rulebase	Name of the rulebase.
rulebase-version	Version of the currently selected rulebase.
rulebase-build-time	Time the currently selected rulebase was built, expressed as an ISO 8601 UTC data time.
opm-version	Version of Oracle Policy Modeling with which the currently selected rulebase was built.
version	Version of the Oracle Web Determinations running; for example, 10.1.0.1.
case-id	Current case id attached to the session.
page-uri	URI of the current screen.
save-case-uri	URI to go to the <i>Save</i> screen. Used by the menu bar Save button.
save-case-as-uri	URI to go to the <i>Save As</i> screen. Used by the menu bar Save As button.
summary-screen-uri	URI to go to the <i>Summary</i> screen. Used by the menu bar Summary button.
data-review-screen-uri	URI to go to the <i>Data Review</i> screen. Used by the menu bar Data Review button.
close-session-uri	URI to go to the <i>Close Session</i> screen. Used by the menu bar Close button.
restart-session-uri	URI to go to the <i>Restart Session</i> screen. Used by the menu bar Restart button.
load-case-uri	URI to go to the <i>Load</i> screen. Used by the menu bar Load button.

Screen and interview objects

The Interview Engine exposes many useful objects via the **Screen** and **InterviewSession** objects. The **InterviewSession** object encapsulates other useful objects inside it. These objects need to be accessed via one of the Interview Session's methods and stored in a Velocity variable as follows:

```
#set $rulebase = $screen.getInterviewSession().getInterviewRulebase().getRulebase()
${rulebase.getLocale()}
```

The following table contains useful objects from the Interview Engine (or more accurately the Class), along with their description, and usage tips. For specific variables and methods of each Class, refer to Interview Engine's API:

Object name	Description	Usage
Screen	<ul style="list-style-type: none"> Contains objects and data pertinent to the current screen displayed. Also encapsulates InterviewSession and SessionContext objects. 	<ul style="list-style-type: none"> Useful for accessing interview objects such as InterviewSession and SessionContext. Useful for accessing screen-related objects and data such as current screen controls, id/title, commentary, warning/errors, screen messages. Accessible via the <code>#{screen}</code> Velocity variable.
InterviewSession	<ul style="list-style-type: none"> Encapsulates object containing user-provided instance data from the interview so far (the Instance Data). Encapsulates data about the interview's rulebase (the Rulebase Model Data). See Understand the InterviewSession. 	<ul style="list-style-type: none"> Access objects that contain rulebase model and instance data. Access Commentary service which can provide commentary functions such as getting commentary text of a target control. screen.getInterviewSession().
SessionContext	<ul style="list-style-type: none"> Responsible for storing the state of the interview session such as current screen, case ID, errors/warnings, last submit action/values, interview goal. Because this object stores the state of the interview session, it encapsulates InterviewSession as well. 	<ul style="list-style-type: none"> Access <i>administrative</i> data about the current interview session such as case ID, errors, last submit action object (SubmitAction). screen.getSessionContext().
Session	<ul style="list-style-type: none"> Contains user-provided instance data from the interview so far (the Instance Data). 	<ul style="list-style-type: none"> Access instance data of the current interview session, such as the value of a certain Global attribute to check if it is known or unknown, or to see if a particular entity has entity instances added. interviewSession.getRuleSession().
Rulebase	<ul style="list-style-type: none"> Contains data about the interview's current rulebase (the Rulebase Model Data). 	<ul style="list-style-type: none"> Access rulebase model data such as the entities of the rulebase, each entity's attributes, and relationships between entities. interviewSession.getRulebase().getRulebase().

Specify the user interface language for rule authoring on the command line

The user interface language setting controls what language is used for the user interface of all Oracle Policy Modeling components, such as dialog boxes and messages. This includes the user interface components inside Word and Excel.

The user interface language setting in Oracle Policy Modeling is specified from the command line using the following format:

```
Oracle.Policy.Modeling.exe --language= <language identifier>
```

Oracle Policy Modeling supports the following language identifiers:

Language Identifier	Language Name
ar	Arabic
zh-CN	Chinese (Simplified)
zh-HK	Chinese (Traditional)
cs	Czech
da	Danish
nl	Dutch
en	English (American)
en-GB	English (British)
fi	Finnish
fr	French
de	German
he	Hebrew
it	Italian
ja	Japanese
ko	Korean
nb-NO	Norwegian (Bokmål)
pl	Polish
pt-BR	Portuguese (Brazilian)
pt-PT	Portuguese (European)
ru	Russian

Language Identifier	Language Name
es	Spanish
sv	Swedish
th	Thai
tr	Turkish

When the user interface language is specified on the command line, the ability to change the currently selected user interface language interactively will be disabled (for information on how to select the user interface language interactively, see the topic "Select the user interface language for rule authoring" in the "Languages" section of the Oracle Policy Modeling User's Guide.)

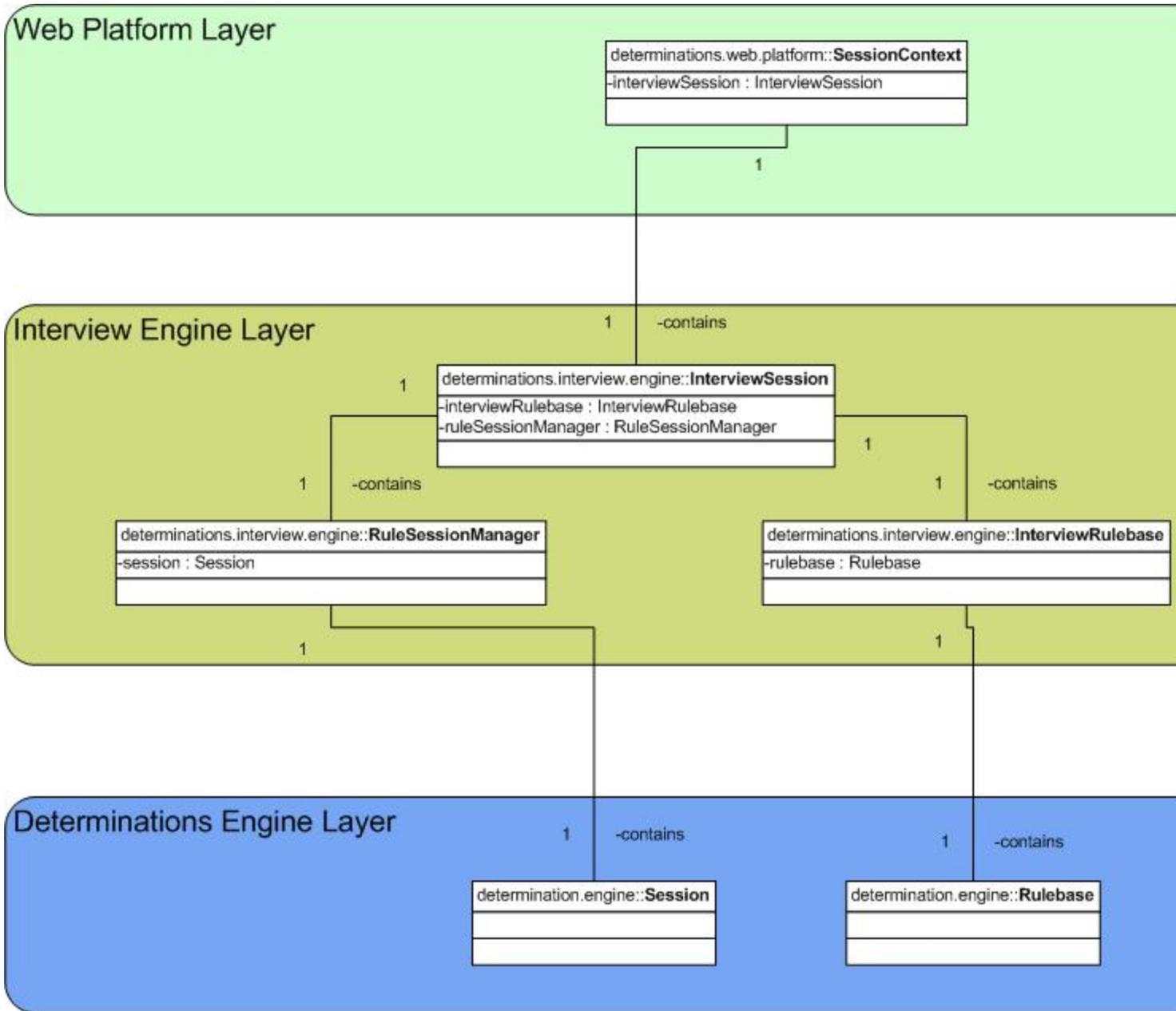
Understand the InterviewSession

The **InterviewSession** is the core component of an Interview Session. It is responsible for managing the data held within the rule session, creating screens and driving interview flows. As such, it is common to interact with this object when building extensions. This following information describes some of the common things you may wish to use the interview session for.

Most importantly, from a extension perspective, the **InterviewSession** object contains the current data of the session interview (the Instance Data) as well as data about the interview's rulebase (the Rulebase Model Data). The **InterviewSession** object exists in the Interview Engine Layer.

- The **InterviewSession** is a member of the **SessionContext** object.
- The **InterviewSession** encapsulates the **RuleSessionManager** and **InterviewRulebase** objects.
 - The **RuleSessionManager** provides access to the **Session** object (`com.oracle.determinations.engine.Session`) from the Interview Layer, another object in the Determinations Engine containing the *instance data*.
 - The **InterviewRulebase** provides access to the **Rulebase** object (`com.oracle.determinations.engine.Rulebase`) from the Interview Layer, another object in the Determinations Engine containing the *rulebase model data*.
 - In summary, the **InterviewSession** encapsulates the **Session** and **Rulebase** objects - objects in the Determinations Engine domain that contain the instance data and rulebase model data respectively.

The following diagram demonstrates the class hierarchy and domains:



The **InterviewSession** object has many other members - it encapsulates session state data in the Determinations Engine layer. But for this section we will focus on its Session member and indirectly the Rulebase object (via its InterviewRulebase member) because those two objects contain the instance data and rulebase model data of the current Web Determinations Interview. For more information on how the Interview Session, and the Interview Engine API works in general see [Create a custom interview user experience](#).

Which Web Determinations extensions can access the InterviewSession object

The **InterviewSession** object is accessible to some plugins and Event Handlers that require access to instance data and/or rulebase model data to provide functionality.

- Most plugins have access to the **InterviewSession** object, as input argument for one or more of their methods.
- Availability of the **InterviewSession** to each Event Handler varies, depending on the intended purpose of the Event. The **InterviewSession** is accessible to an Event Handler either via the passed **Event** object (as a member of the **Event** object), or the sender object itself. See [Events and Event Handlers](#) for more information about each event type

Accessing the Rulebase Model Data and Instance Data (via InterviewSession object)

It is important to understand the **InterviewSession** object as this object contains both Rulebase Model Data and Instance Data. Thus it is used by many plugins and Event Handlers.

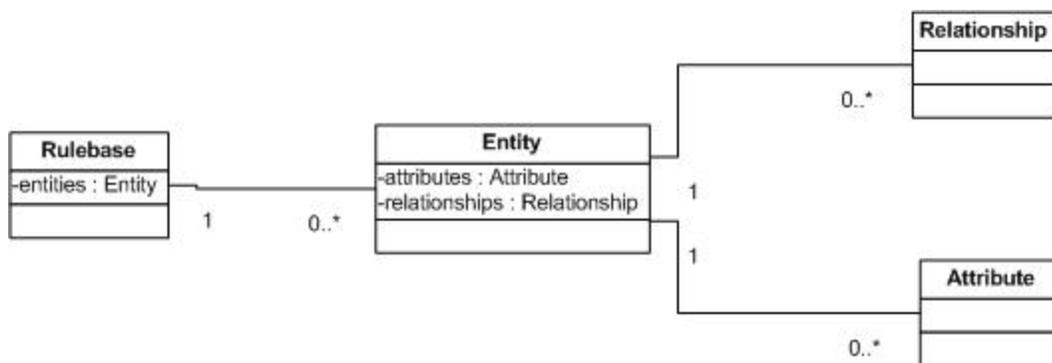
Definitions:

- **Rulebase Model Data** - this is commonly referred to as the *rulebase*. Model data covers the entities, their attributes, and possible relationships between two entities
- **Instance Data** - this is the actual instances of entities, actual values of entity attributes, and actual instances of relationships between two entities provided during Web Determinations Interview

Using the Rulebase Model Data

The Rulebase Model Data is encapsulated in the **Rulebase** object ([com.oracle.determinations.engine.Rulebase](#)). The Rulebase object is accessible if the Web Determinations extension has access to the **SessionContext** object, **InterviewSession** object, or **InterviewRulebase** object.

The basic hierarchy in the Rulebase is that the **Rulebase** object contains a list of **Entity** objects. Each **Entity** object contains all the **Attribute** and **Relationship** objects with which it is associated. This hierarchy matches the hierarchy of rule authoring in Oracle Policy Modeling - each attribute and relationship is always related to an entity.



With knowledge of the above hierarchy, the most basic way to access each entity and each attribute and relationship of an entity would be by looping through each entity, and retrieving and looping through attributes and relationships for each entity. See sample code below.

Loop through all Entities, their Attributes, and Relationships (Java)

```

public InterviewUserData load(SecurityToken token, String caseID,
                             InterviewRulebase rulebase)
{
    Rulebase rule = rulebase.getRulebase();
    System.out.println("Rulebase: " + rule.getBaseFileName());

    //Loop through all Entities of the current Rulebase
    List<Entity> entities = rule.getEntities();
    for(Entity entity : entities)
    {
        System.out.println("Entity: " + entity.getName());

        //Loop through all Attributes of the current Entity
        List<Attribute> entityAttributes = entity.getAttributes();
        for(Attribute attribute : entityAttributes)
        {
            System.out.println("Attribute Name: " + attribute.getName());

            //Loop through all Custom Properties of the current Attribute
            Map<String, String> attributeProperties = attribute.getProperties();
            Set attributePropKeys = attributeProperties.keySet();
            Iterator keyIterator = attributePropKeys.iterator();
            System.out.println("==Attribute Properties==");
            while(keyIterator.hasNext())
            {
                String key = (String)keyIterator.next();
                String value = (String)attributeProperties.get(key);
                System.out.println("Property key: " + key + ", Property value: " + value);
            }
        }
    }

    //Loop through all Relationships of the current Entity
    List<Relationship> entityRelationships = entity.getRelationships();
    for(Relationship relationship : entityRelationships)
    {
        System.out.println("Relationship Name: " + relationship.getName());
        if(relationship.getSymmetricRelationship() != null)
        {
            System.out.println("Reverse Relationship Name: " + relationship.getSymmetricRelationship
            ().getName());
        }

        //Loop through all Custom Properties of the current Relationship
        Map<String, String> relationshipProperties = relationship.getProperties();
        Set relationshipPropKeys = relationshipProperties.keySet();
    }
}

```

```

        Iterator keyIterator = relationshipPropKeys.iterator();
        System.out.println("==Relationship Properties==");
        while(keyIterator.hasNext())
        {
            String key = (String)keyIterator.next();
            String value = (String)relationshipProperties.get(key);
            System.out.println("Property key: " + key + ", Property value: " + value);
        }
    }
}
...
...

```

For more information about the **Rulebase**, **Entity**, **Attribute**, and **Relationship** objects, please consult its API documentation for Java or .NET.

Rulebase object: useful methods - the following is a list of Rulebase object members and also useful methods to access those members.

Note that an **Entity Attribute** or **Entity Relationship** object can be accessed directly from the **Rulebase** object via helper methods, to eliminate having to manually find a specific entity to retrieve an attribute or relationship.

```

|| Member || Description
|| Access Methods
||
| Global Entity
| This is the Entity object for the Global Entity of the current rulebase
| Entity getGlobalEntity()
|
| Entities | All the Entities of the current rulebase
| List getEntities()
Entity getEntity(String name)
|
| Entity Attribute
| Retrieve a specific Attribute object of an entity
| Attribute getAttribute(String attributeName, String entityName) |
|
| Entity Relationship
| Retrieve a specific Relationship object of an entity
| Relationship getRelationship(String relationshipName, String entityName)
|

```

Properties	Key/value pairs associated by the rulebase author to the rulebase for custom func-	Map getProperties()
------------	--	---------------------

tionality	String getPropertyValue(String key)
-----------	-------------------------------------

Using the Instance Data

The Instance Data is encapsulated in the **Session** object (`com.oracle.determinations.engine.Session`). The **Session** object is accessible if the Web Determinations extension has access to the **SessionContext** object, or the **InterviewSession** object.

The various Instance Data objects are as follows:

- **Entity Instances** - Entity Instance data is represented by the object **EntityInstance**.
- **Attribute value** - The value of an attribute is represented by various datatype objects corresponding to the various attribute types supported by Oracle Policy Automation, such as Boolean, Number, Date, Currency, Unknown, Uncertain.
- **Relationship instance** - A relationship instance works different to an entity or attribute in that relationship instances do not exist by themselves, but instead exist through their source and target **EntityInstance**; for example, a particular **EntityInstance** may have multiple relationships, and therefore may be linked to various source or target **EntityInstances**. Therefore the relationship instance exists by having a source and target **EntityInstance**.

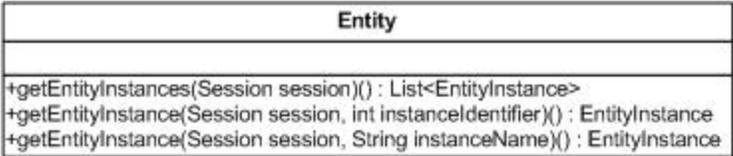
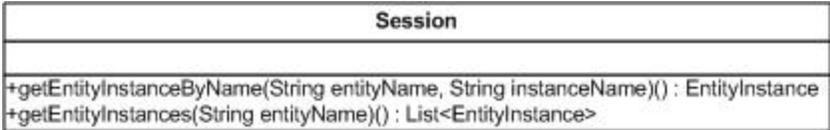
Accessing Instance Data is different to Rulebase Model Data - the Instance Data inside the session is not accessible directly. The most common way to access Instance Data is to use the Rulebase Model data object (Entity, Attribute, or Relationship) and pass session data to their methods (either **Session** or **EntityInstance**). This is explained in more detail in the subsections below.

Entity Instance

There are two ways to access Entity Instance Data.

1. Via the **Entity** object
2. Via the **Session** object

The following Class diagram shows methods in the **Entity** and **Session** object that retrieve **EntityInstance** objects; refer to the topic [Sample code - using Instance Data](#) for actual code usage:



Attribute Instance

There is only one way to access Attribute Instance Data (Attribute value), and it is by using the **Attribute** object, passing an **EntityInstance**, and retrieving the value of the attribute for the **EntityInstance** object passed. See the sample code below and also the section below, *Sample code - using Instance Data*.

Accessing Attribute Instance Data

```
//Get Entity Instance
Entity entity1 = rulebase.getEntity("entity1");
EntityInstance entity1Instance = entity1.getEntityInstance(session, 1);

//Get Attribute value
Attribute attribute1 = entity1.getAttribute("attribute1");
```

Relationship Instance (source and target EntityInstances)

As mentioned previously - Relationship Instance Data cannot be retrieved. Rather it is a matter of retrieving Target EntityInstance (s) by providing the Source EntityInstance - or vice versa, retrieving Source EntityInstance(s) by providing the TargetEntityInstance. See the sample code below and also the section below, *Sample code - Using Instance Data*.

Accessing Relationship source and target EntityInstance(s)

```
//Get Entity Instance
Entity entity1 = rulebase.getEntity("entity1");
EntityInstance entity1Instance = entity1.getEntityInstance(session, 1);

//Get Relationships for the Entity where the Entity is the Source, then get its Target EntityInstances
List<Relationship> sourceRelationships = entity1.getRelationships();
for(Relationship sourceRelationship : sourceRelationships)
{
    List<EntityInstance> targetEntityInstances = sourceRelationship.getAllTargets(entity1Instance);
}
}
```

Sample code - using Instance Data

Below is sample code that combines all the sample code for Entity, Attribute, and Relationship Instance Data and uses them together.

Sample code for using Instance Data objects (Java)

```
public String save(SecurityToken token, String requestCaseID,
                 InterviewSession session) {

    Session ruleEngineSession = session.getRuleSession();
    Rulebase rulebase = ruleEngineSession.getRulebase();
```

```

//Loop through all Entities of the current Rulebase
List<Entity> entities = rulebase.getEntities();
for(Entity entity : entities)
{
    //Get all EntityInstances of the current Entity, and loop through them
    List<EntityInstance> entityInstances = (List<EntityInstance> )entity.getEntityInstances
(ruleEngineSession);
    for(EntityInstance entityInstance : entityInstances)
    {

        //Get all Attributes and their values for the current EntityInstance
        List<Attribute> attributes = entityInstance.getAttributes();
        for(Attribute attribute : attributes )
        {
            Object attributeValue = attribute.getValue(entityInstance);
        }

        //Get all Relationships of the current Entity where the Entity is the Source, and loop through
        List<Relationship> sourceRelationships = entityInstance.getRelationships();
        for(Relationship relationship : sourceRelationships )
        {
            //For each relationship retrieve the Source or Target EntityInstances
            if(relationship.getRelationshipType().isSingleTarget())
            {
                EntityInstance targetInstance = relationship.getTarget(entityInstance);
            }
            else
            {
                List<EntityInstance> targetInstances = relationship.getAllTargets(entityIn-
stance);
            }
        }
    }
}
...

```

Modifying the Instance Data

As mentioned in the overview and in [Using the Instance Data](#), the Instance Data from a Web Determinations interview lives within the **Session** object in the Determinations Engine Layer.

Modification to the Instance Data inside the **Session** object is different to accessing/retrieving the Instance Data. It is done via transactions, and the changes to be made to the Instance Data are encapsulated inside an object.

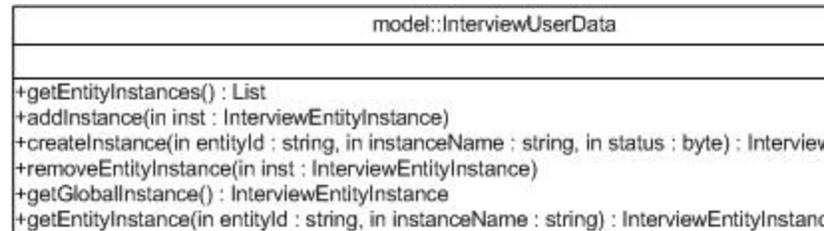
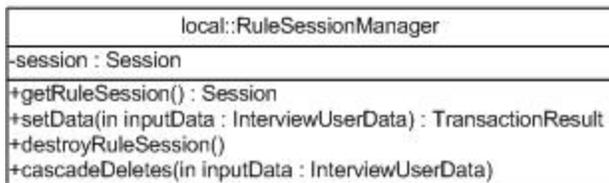
Modifying the Instance Data inside the **Session** object is done indirectly via the two objects below:

- **RuleSessionManager** - an object in the Interview Engine layer that is responsible for managing transactions (mainly modifications) to the session data in the Determinations Engine Layer (the **Session** object) from the Interview Engine

Layer.

- **InterviewUserData** - an object in the Interview Engine layer that represents Instance Data changes that need to be made to the **Session** object, and is read by the **RuleSessionManager** as part of a transaction for modifying the Instance Data inside the **Session** object.

The following are class diagrams for the **RuleSessionManager** and **InterviewUserData** objects. Only class members pertinent to modification of Instance Data is displayed:



The most common process for using **InterviewUserData** is as follows:

1. An Interview Engine object creates an **InterviewUserData** object.
2. The **InterviewUserData** object is populated with content - the changes to be made; for example, additions and deletions to instance/attribute/relationship instance data, modifications to attribute value of an instance.
3. The **InterviewUserData** object is passed by the **Interview Engine** object to the **RuleSessionManager.setData()** method.
4. The **RuleSessionManager** reads the **InterviewUserData** and performs the changes to the **Session** object.

An example of how to use the **InterviewUserData** object to modify Instance Data is demonstrated by the Web Determinations Data Adaptor extension and its **load()** method. The **load()** method allows Instance Data from a datasource to be loaded into the interview session before the user starts the interview (see [Data Adaptor Plugin](#)).

When Web Determinations calls the Data Adaptor **load()** method, it creates, populates, and returns an **InterviewUserData** object that represents the Instance Data to be loaded. Web Determinations then passes the **InterviewUserData** object to the **RuleSessionManager.setData()** so the Instance Data can be loaded into the **Session** object (which would have no Instance Data since it's newly started).

Note: Because **DataAdaptor load()** is always performed to a newly created **Session** object, modifications to the Instance Data (signified by the 'Perform Modifications to Session' message) will all be *additions* instead of additions/modifications/deletions since there is no Instance Data in the **Session** object to modify or delete.

For information on how to create an **InterviewUserData** object see [Construct the InterviewUserData](#).

Commentary - pseudo code

The code below is a pseudo code for the Commentary plugin, which incorporates some of the tips and recommendations from the [Commentary plugin](#) page.

Use the pseudo code below as a starting point for a custom Commentary plugin.

CommentaryPseudoCode

```
package com.oracle.determinations.interview.engine.userplugins.commentary;

import com.oracle.determinations.interview.engine.InterviewSession;
import com.oracle.determinations.interview.engine.exceptions.UnsupportedException;
import com.oracle.determinations.interview.engine.plugins.InterviewSessionPlugin;
import com.oracle.determinations.interview.engine.plugins.InterviewSessionRegisterArgs;
import com.oracle.determinations.interview.engine.plugins.commentary.CommentaryProviderPlugin;
import com.oracle.determinations.interview.util.TypedInputStream;

public class PseudoCommentary implements CommentaryProviderPlugin {

    //REQUIRED - for Plugin Architecture
    public PseudoCommentary()
    {

    }

    //If the target is not redirect - return the commentary content for the 'target'
    public TypedInputStream getCommentaryContent(InterviewSession session, String target) {
        TypedInputStream commentContent = null;
        String fulltarget = session.getLocale() + "/" + target;

        //Retrieve data using fulltarget e.g. from datasource
        //Format data to HTML if needed

        return commentContent;
    }

    //If the target is a redirect - return the URL commentary page for the 'target'
    public String getCommentaryURL(InterviewSession session, String target) {
        String fulltarget = session.getLocale() + "/" + target;

        //Retrieve URL or data to build URL using fulltarget e.g. from datasource
        //Build URL from data if needed
        //Return URL
    }
}
```

```
//Return true if the commentary for the 'target' is available
public boolean hasCommentary(InterviewSession session, String target) {
    String fulltarget = session.getLocale() + "/" + target;

    //Use fulltarget to check if commentary for target exists
    //return true or false
}

// Return true if the commentary for this Web Determinations Interview is available
public boolean isCommentaryEnabled(InterviewSession session) {

}

// Return true if the target is a redirect, i.e. uses URL for commentary. Otherwise return false
public boolean isCommentaryRedirect(InterviewSession session, String target) {
    String fulltarget = session.getLocale() + "/" + target;
    //Use fulltarget to determine if the target is a redirect or not
}

//REQUIRED - for Plugin interface
public InterviewSessionPlugin getInstance(InterviewSessionRegisterArgs args) {
    return new RedirectCommentary();
}
}
```

Data Adaptor - pseudo code

The following code is the pseudo code for the Data Adaptor plugin, which incorporates some of the tips and recommendations from the [Data Adaptor overview](#) topic (use this pseudo code as a starting point for a Data Adaptor plugin):

PseudoDataAdaptor

```
package com.oracle.determinations.interview.engine.userplugins;

import com.oracle.determinations.engine.Attribute;
import com.oracle.determinations.engine.Entity;
import com.oracle.determinations.engine.EntityInstance;
import com.oracle.determinations.engine.Relationship;
import com.oracle.determinations.engine.Rulebase;
import com.oracle.determinations.engine.Session;
import com.oracle.determinations.engine.local.LocalRulebase;
import com.oracle.determinations.engine.local.LocalSession;
import com.oracle.determinations.interview.engine.InterviewRulebase;
import com.oracle.determinations.interview.engine.InterviewSession;
import com.oracle.determinations.interview.engine.data.model.InterviewEntityInstance;
import com.oracle.determinations.interview.engine.data.model.InstanceStatus;
import com.oracle.determinations.interview.engine.data.model.InterviewUserData;
import com.oracle.determinations.interview.engine.plugins.DataAdaptorPlugin;
import com.oracle.determinations.interview.engine.plugins.InterviewSessionPlugin;
import com.oracle.determinations.interview.engine.plugins.InterviewSessionRegisterArgs;
import com.oracle.determinations.interview.engine.SecurityToken;
import com.oracle.determinations.interview.util.EngineConstants;
import com.oracle.util.plugins.Plugin;
import com.oracle.util.plugins.RegisterArgs;

public class PseudoDataAdaptor implements DataAdaptorPlugin {

    //REQUIRED for Plugin implementations - "constructor with no args"
    public PseudoDataAdaptor()
    {

    }

    //REQUIRED by Plugin interface
    public InterviewSessionPlugin getInstance(InterviewSessionRegisterArgs args) {
        //Inspect args if needed to determine if this DataAdaptor should be used for the current InterviewSession
        return null;
    }

    //REQUIRED by DataAdaptor interface
    public boolean dataAdaptorProvidesCaseID() {
        //Return true if this DataAdaptor will provide the Case ID (in save method) to use
    }
}
```

```

// this means that in the UI front-end, the user cannot 'Save As' and provide a Case ID, only 'Save' to
//save the current session
//Return false if the Case ID is not provided by the Data Adaptor - e.g. by the user or another Web
Determinations
    //Extension
    return false;
}

//REQUIRED by DataAdaptor interface
public String[] listCases(SecurityToken token, InterviewRulebase rulebase) {
    //Setup connection to datasource

    //Authenticate using the Security token

        //Access datasource
        //Retrieve list of Case ID's, format to String array
        //Return String array

    //If there are errors, raise an Error or unchecked Exception (e.g. implementations of RuntimeException)
    return new String[0];
}

//REQUIRED by DataAdaptor interface
public InterviewUserData load(SecurityToken token, String caseID,
    InterviewRulebase rulebase) {

    //Setup connection to datasource

    //Authenticate using the Security token

        //Validate Case ID on datasource

        //Create new InterviewUserData

        //Get Global instance from InterviewUserData, then get Global attribute values using rulebase
        //model data and datasource
        //Get all Entity instances and its attribute values using rulebase model data and datasource
        //Create relationships between the Entity instances retrieved using rulebase model data and
        //datasource

    //If there are errors, raise an Error or unchecked Exception (e.g. implementations of of RuntimeException)
    return null;
}

//REQUIRED by DataAdaptor interface
public String save(SecurityToken token, String requestCaseID,

```

```
        InterviewSession session) {  
  
        //Setup connection to datasource  
  
        //Authenticate using the Security token  
  
        //Start datasource transaction  
  
                //If the Case ID datasource-generated, create the record first for reference to the other  
                //entities/attributes/relationships  
                //Persist Entity instances and their attributes via rulebase model data  
                //Persist relationships between instances  
  
        //COMMIT datasource transaction if no errors  
  
        //If there are errors, ROLLBACK the transaction and raise an Error or unchecked Exception (e.g.  
        //implementations of of RuntimeException)  
        //Alternatively, an empty or null string can be returned to trigger the 'Save Failed' error  
        return null;  
    }  
}
```

Formatter - psuedo code

The code below is a pseudo code for the Formatter, which incorporates some of the tips and recommendations from the [Formatter plugin](#) page.

Use the pseudo code below as a starting point for a Formatter plugin.

FormatterPseudoCode

```
package com.oracle.determinations.interview.engine.userplugins;
import java.util.Iterator;
import java.util.Map;
import java.util.Set;

import com.oracle.determinations.engine.Attribute;
import com.oracle.determinations.web.platform.plugins.PlatformSessionPlugin;
import com.oracle.determinations.web.platform.plugins.PlatformSessionRegisterArgs;
import com.oracle.determinations.web.platform.plugins.formatting.DefaultFormatter;
import com.oracle.determinations.web.platform.plugins.formatting.WebDeterminationsFormatterPlugin;

public class LatitudeLongitudeFormatter implements WebDeterminationsFormatterPlugin {

    private DefaultFormatter defaultFormatter;

    //REQUIRED - for Plugin architecture
    public PseudoFormatter()
    {

    }

    public PseudoFormatter(DefaultFormatter defaultFormatter)
    {
        this.defaultFormatter = defaultFormatter;
    }

    public PlatformSessionPlugin getInstance(PlatformSessionRegisterArgs args) {
        new PseudoFormatter();
    }

    //return a String equivalent of the provided 'value' of type 'type'
    public String getFormattedValue(byte type, Object value) {
        // TODO Auto-generated method stub
        return null;
    }

    //return a String equivalent of the provided 'value' using 'type' and attribute
    public String getFormattedValue(byte type, Object value, Attribute attr) {
        // TODO Auto-generated method stub
    }
}
```

```
        return null;
    }

    //return the raw Object format of the provided 'value', using 'type' and attribute to parse
    public Object parse(byte type, String value, Attribute attr) {
        // TODO Auto-generated method stub
        return null;
    }

    //return the raw Object format of the provided 'value', using 'type' to parse
    public Object parse(byte type, String value) {
        // TODO Auto-generated method stub
        return null;
    }

    public boolean canFormatTemporal() {
        // TODO Auto-generated method stub
        return false;
    }

    public boolean canFormatType(byte type) {
        // TODO Auto-generated method stub
        return false;
    }

    public void initialize(Map properties) {
        // TODO Auto-generated method stub
    }
}
```

List Provider - pseudo code

The code below is a pseudo code for the List Provider, which incorporates some of the tips and recommendations from the [List Provider overview](#) topic.

Use the pseudo code below as a starting point for a List Provider plugin.

PseudoListProvider

```
package com.oracle.determinations.interview.engine.userplugins;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

import com.oracle.determinations.engine.Attribute;
import com.oracle.determinations.interview.engine.InterviewSession;
import com.oracle.determinations.interview.engine.plugins.InterviewSessionPlugin;
import com.oracle.determinations.interview.engine.plugins.InterviewSessionRegisterArgs;
import com.oracle.determinations.interview.engine.plugins.listprovider.ListProviderPlugin;
import com.oracle.determinations.interview.engine.screen.InputInterviewControl;
import com.oracle.determinations.interview.engine.screen.ListOption;

public class DerbyListProvider implements ListProviderPlugin {

    //REQUIRED by Plugin interface
    public DerbyListProvider()
    {

    }

    //REQUIRED by ListProviderPlugin interface
    public List getListOptions(InputInterviewControl controlInstance,
                             InterviewSession session) {

        //Setup connection to datasource

        //Access datasource

        //Loop through dataset
        //for each item, create a ListOption object
        //add ListOption object to the List object
    }
}
```

```
        //Return List object
    }
    //REQUIRED by Plugin interface
    public InterviewSessionPlugin getInstance(InterviewSessionRegisterArgs args) {
        //Inspect args if needed to determine if this ListProvider should be used for the current InterviewSession
        return null;
    }
}
```

Legacy document generation

With the updates to the document generation capabilities and framework in 10.3.0, previous rulebases designed to perform document generation will no longer function out of the box. In order to get these rulebases working two steps must be performed, outlined below.

Upgrade the rulebase

Previously, where all of the information required for the document generation process was stored against the *Summary* screen link, this changed in 10.3.0 with the introduction of **Document** objects to the *Screens* file.

When a rulebase authored pre-10.3.0 is loaded, the user will be prompted to upgrade in order to continue working with the rulebase. Part of this upgrade process moves the document generation configuration parameters from the **Document** link on the *Summary* screen to Custom Properties of newly created **Document** objects.

The most important custom property is **XSLT_File**, which defines where the XSLT file used to generate the document can be found. As with other template paths this is relative to the root of the rulebase archive.

Install the Legacy Document Generation plugin

Java

1. Locate the Legacy Document Generation plugin; Oracle Policy Modeling users can find this under `\templates\plugins\legacy-document-generation\java\web-determinations-engine-legacy-docgen.jar`. Those who've installed the Java Runtime will need to look in `\plugins\legacy-document-generation\web-determinations-engine-legacy-docgen.jar`.
2. Place the file in the `\WEB-INF\classes\plugins\` directory of your Web Determinations deployment (those debugging from Oracle Policy Modeling can also place this inside the *Release* folder of their Rulebase Project).

Note: if your deployment does not support automatic introspection and loading of plugins you may need to perform additional configuration steps.

This plugin alone is not enough to enable legacy document generation. The plugin itself relies on the Apache FOP engine, available from <http://xmlgraphics.apache.org/fop/>, and has been tested with version 0.95. Once the *fop.jar* file and its dependencies have been downloaded, place it in your `\WEB-INF\lib\` directory to ensure it is detected.

.Net

1. Locate the Legacy Document Generation plugin; Oracle Policy Modeling users can find this under `\templates\plugins\legacy-document-generation\dotnet\Legacy.Docgen.Plugin.dll`. Those who've installed the .Net Runtime will need to look in `\plugins\legacy-document-generation\Legacy.Docgen.Plugin.dll`.
2. Place the file in the `\bin\plugins\` directory of your Web Determinations deployment.

This plugin alone though is not enough to enable Legacy Document Generation. The plugin itself relies on the nFop engine, available from <http://sourceforge.net/projects/nfop/>, and has been tested with version 1.0.0. Once the *fop.net.dll* file has been downloaded, place it in your `\bin\` directory to ensure it is detected.

Miscellaneous notes

- Though there can only be one Document Generation plugin registered at once, it still hooks in to the new BI Publisher Document Generation plugin in order to generate documents which do not have the "XSLT_File" custom property specified with a value.
- Though generation of HTML documents in the Legacy Generator does not rely on the FOP engines mentioned previously, they must still be deployed in order for the plugin to load correctly.

Third party products and licenses

This topic lists the third-party products used in Oracle Policy Automation (OPA) v10.4 and [the licenses under which they are being used](#).

List of Third-Party Products

This is a list in alphabetical order of the third-party products used in OPA v10.4. Follow the links for more information about each product.

- [Java Web Services Developer Pack](#)
- [jQuery UI](#)
- [JRegex](#)
- [Log4j](#)
- [Log4Net](#)
- [NVelocity](#)
- [Tomcat](#)
- [Velocity](#)

Java Web Services Developer Pack

Version: 2.0

Vendor: Sun Microsystems, Inc (now Oracle)

Product(s): Policy Modeling, Web Determinations, Determinations Server

This product is licensed under the Apache 2.0 license agreement. See [Apache License Version 2.0](#).

jQuery UI

Version: 1.8.18

Vendor: John Resig

Copyright (c) 2011 John Resig, <http://jquery.com/>([opens in new window](#))

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

JRegex

Version: 1.2

Vendor: Sergey A. Samokhodkin

Product(s): Determinations Engine for Mobile Devices

Copyright (c) 2001, Sergey A. Samokhodkin

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of jregex nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Log4j

Version: 1.2.17

Vendor: Apache Software Foundation

Product(s): Web Determinations, Determinations Server

This product is licensed under the Apache 2.0 license agreement. See [Apache License Version 2.0](#).

Log4Net

Version: 1.2.13

Vendor: Apache Software Foundation

Product(s): Policy Modeling, Web Determinations, Determinations Server

This product is licensed under the Apache 2.0 license agreement. See [Apache License Version 2.0](#).

NVelocity

Version: 1.1.0

Vendor: Castle Project

Product(s): Web Determinations, Determinations Server

This product is licensed under the Apache 2.0 license agreement. See [Apache License Version 2.0](#).

Tomcat

Version: 5.5.35

Vendor: Apache Software Foundation

Product(s): Policy Modeling

This product is licensed under the Apache 2.0 license agreement. See [Apache License Version 2.0](#).

Velocity

Version: 1.7

Vendor: Apache Software Foundation

Product(s): Web Determinations, Determinations Server

This product is licensed under the Apache 2.0 license agreement. See [Apache License Version 2.0](#).

List of Third-Party Licenses

This is a list of the third-party licenses referred to by two or more products used in OPA. Click on the link to see details of the license.

- [Apache License Version 2.0](#)

Apache License Version 2.0

The following applies to all products licensed under the Apache 2.0 License:

You may not use the identified files except in compliance with the Apache License, Version 2.0 (the "License.")

You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>(opens in new window). A copy of the license is also reproduced [below](#).

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

Apache License

Version 2.0, January 2004

<http://www.apache.org/licenses/>(opens in new window)

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. **Definitions.**

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the [Appendix](#) below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original

work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. **Grant of Copyright License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. **Grant of Patent License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. **Redistribution.** You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - a. You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - b. You must cause any modified files to carry prominent notices stating that You changed the files; and
 - c. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - d. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. **Submission of Contributions.** Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. **Trademarks.** This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. **Disclaimer of Warranty.** Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. **Limitation of Liability.** In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

```
Copyright [yyyy] [name of copyright owner]
```

```
Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file  
except in compliance with the License. You may obtain a copy of the License at
```

```
http://www.apache.org/licenses/LICENSE-2.0
```

```
Unless required by applicable law or agreed to in writing, software distributed under the  
License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND,  
either express or implied. See the License for the specific language governing permissions  
and limitations under the License.
```

Tutorials and examples

Tutorials

Create and use a JAX-WS web service client for Oracle Determinations Server

Create and use a C# client for Oracle Determinations Server

Embed Web Determinations within my portal - a simple implementation

Examples

Batch Processor

Create test cases in the Batch Processor

Run the Batch Processor (InsuranceFraudScore)

Use the Batch Processor with a Database

Determinations Engine

Create a Custom Function Extension to capitalize text

Create a Custom Function Extension to default a value

Create a Rulebase Listener to preload reference data

Hide eligibility criteria from a decision report depending on benefit applied for

Retrieve a Decision Report

Rulebase Configuration File

Determinations Server

Assess Request xml

Assess Response xml

Determinations Server Custom Service Example

Implement Clients for the Assess Service

Implement Clients for the Interview Service

Investigation using the Determinations Server Interview Service

Interview Engine

Commentary Plugin - Derby Commentary

Commentary Plugin - Redirect Commentary

Data Adaptor Plugin - Derby Data Adaptor

Document Generator Plugin - TxtDocumentGenerator

List Provider Plugin

Rulebase Resolver Plugin - Derby Rulebase Service

Interview Portlet

Create a Custom Screen for the Interview Portlet

Encode the Interview Portlet's response

Extract the username in the Interview Portlet
Pass in parameters for an Interview Portlet across WSRP
Send external events to other portlets

Web Determinations

Add and remove buttons on Entity Collect screens
Custom Control - BenefitCode Walkthrough
Custom Control - CalendarDateControl Walkthrough
Custom Control - FilterListCode Walkthrough
Create a Custom Validator for control validation
Create a Custom Validator for screen validation
Configuration for Classpath-based loading (Java)
Configuration for File System-based loading (Java and .NET)
Data Adaptor Plugin - Autosave with Derby
Dynamically display an error message for a control
Formatter Plugin
Show or Hide an Attribute
Use the OnInterviewSessionCreatedEvent to pre-seed data into a newly created session

Tutorial: Create and use a JAX-WS web service client for Oracle Determinations Server

The advantage of using a rigorous and well defined interface such as a WS-I compliant web service, is the ability to integrate it with other applications, using tools to aid that integration. Using HTTP as the communication layer and XML as the request and response formats, gives a systems integrator an industry standard way of communicating, with the choice of many tools to build that integration.

This tutorial is a quick walkthrough of the direct integration of Oracle Determinations Server running a rulebase with a simple Java application. The generated JAX-WS client will perform the job of creating the Request as a web service call, making the call, and interpreting the response.



A JAX-WS client allows you to build a web service call with Plain Old Java Objects (POJOs). The client performs the conversion to XML and a SOAP Request. The response is likewise converted from XML back to POJOs for ease of integration.

This tutorial uses the JAX-WS Reference Implementation project available from <http://jax-ws.dev.java.net>.

Requirements to follow the tutorial

As this tutorial discusses programming in Java, you should be familiar with java, and be able to set up and run a program in an IDE of your choice (Eclipse or Netbeans for example). The following Software is required to follow this tutorial:

- Oracle Policy Modeling 10.2 or later
- The SimpleBenefits rulebase (located at [examples\rulebases\compiled\SimpleBenefits.zip](#)) in the Oracle Policy Automation Runtime package.
- JAX-WS Reference Implementation Project (<http://jax-ws.dev.java.net/>)
- A Java development environment/IDE with the ability to compile and run Java 1.5 (or later) code

You should follow the instructions for installing JAX-WS correctly on your computer. This tutorial was written using version 2.1.7 of the JAX-WS RI, however older and new versions should work.

1. Compile and run the SimpleBenefits rulebase

Compile and run the SimpleBenefits rulebase

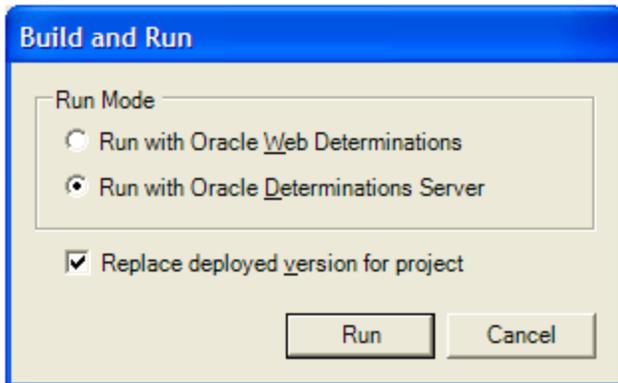
The first thing to do is to have a look at the SimpleBenefits rulebase in Oracle Policy Modeling. This rulebase is a very simple example. It determines 3 goals.

- Is the claimant eligible for the low income allowance?
- What is the claimant's low income allowance amount?
- Is the claimant eligible for the teenage child allowance?

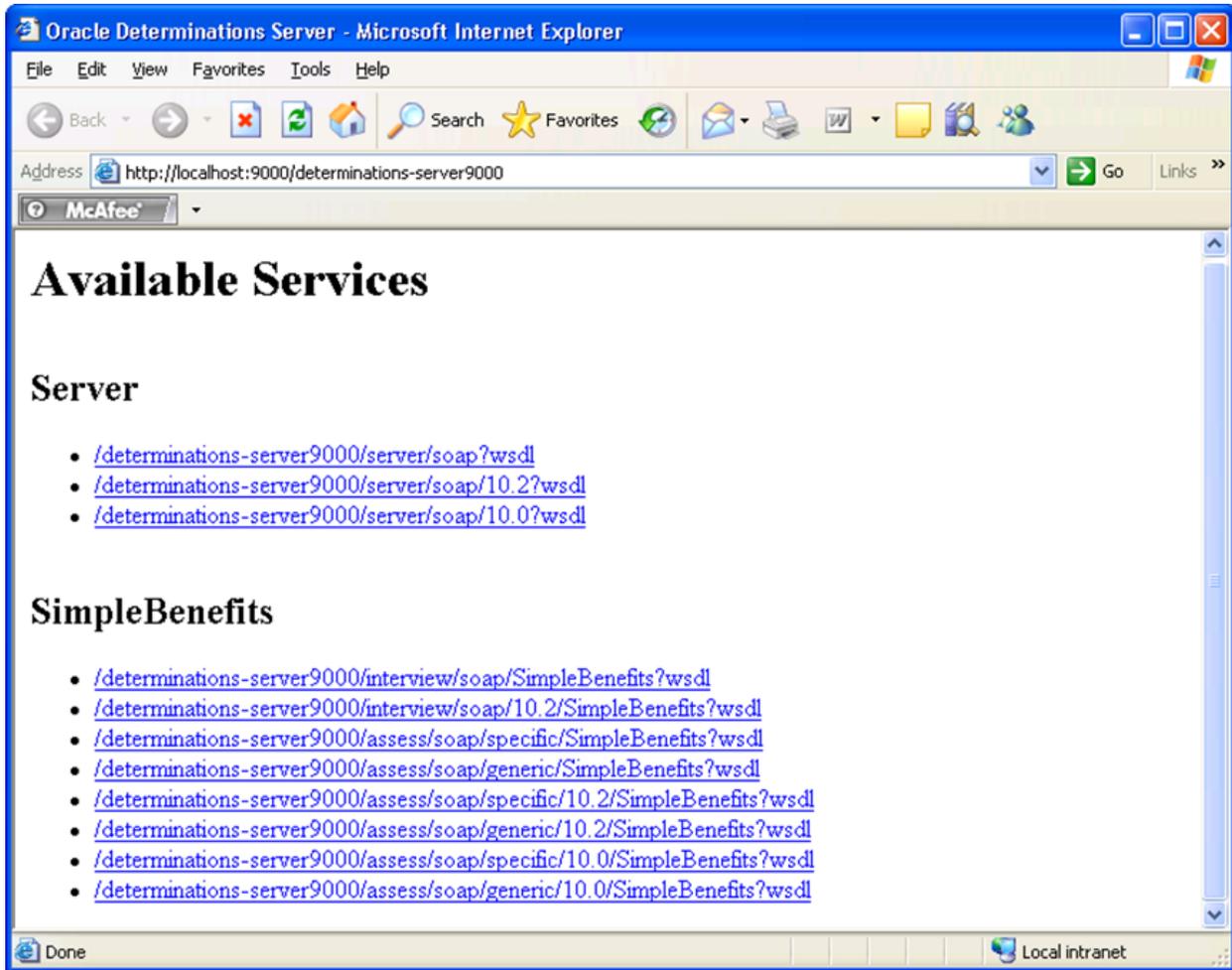
These goals are attributes of the global entity, and there are also child entities - the claimant's children. Eligibility for the low income allowance is based on the information on the global attribute only. Eligibility for the teenage child allowance is based on the claimant's children and their age.

You can run this rulebase in the Determinations Server from Oracle Policy Modeling by the following method:

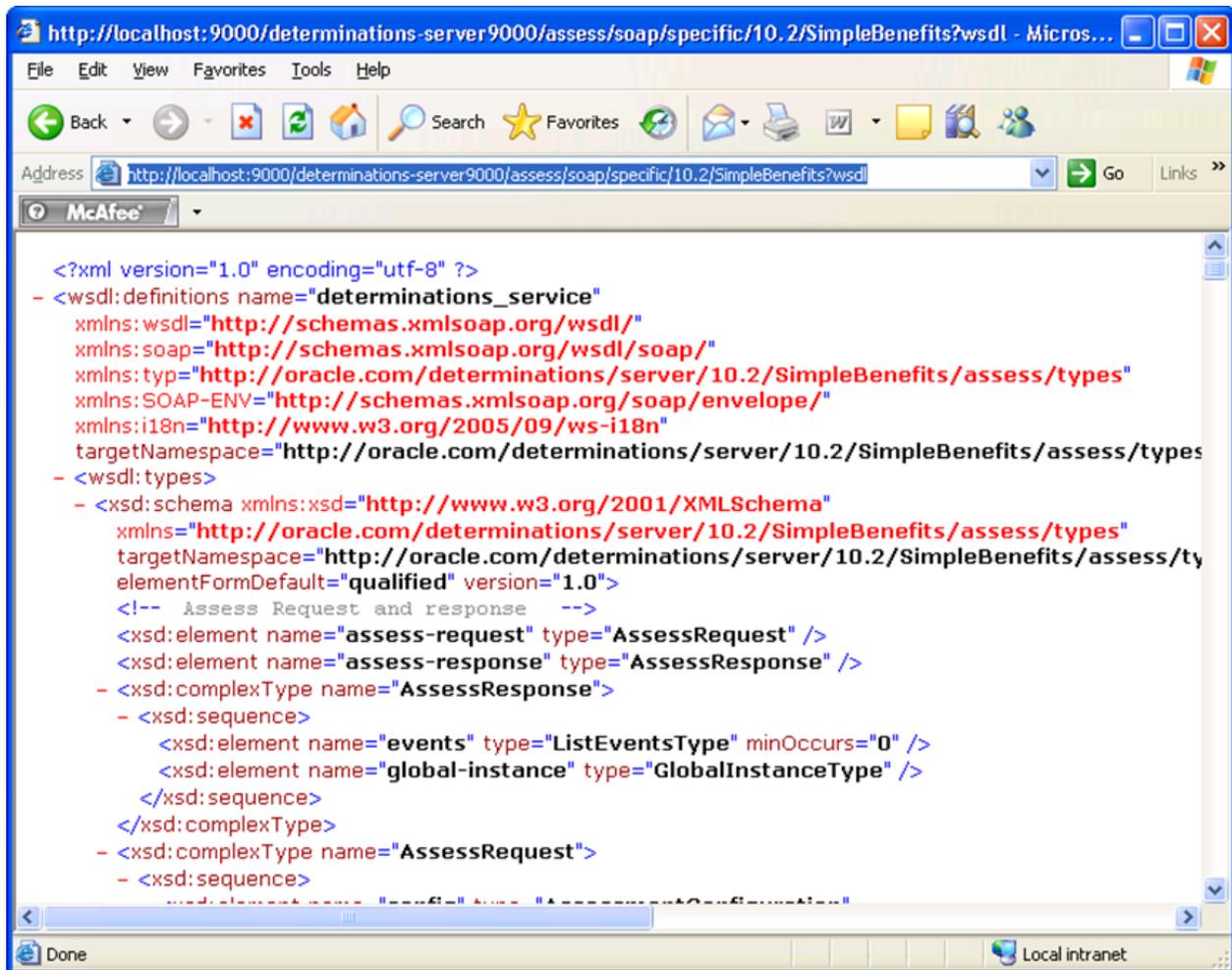
1. From the Build menu, choose Build and Run
2. When the Build and Run Dialog appears, choose Run with Oracle Determinations Server. Also select the Replace deployed version for project



3. After a brief pause, Oracle Policy Modeling should start up your default web browser with the default page for the Determinations Server.



4. You can view the WSDLs for the rulebase by clicking the 10.2 specific WSDL link on the page. In the case of Oracle Policy Modeling running SimpleBenefits, the URL should be: <http://localhost:9000/determinations-server-9000/assess/soap/specific/10.2/SimpleBenefits?wsdl>.
If the WSDL appears as XML in the browser, then the rulebase has successfully deployed to the Determinations Server:



5. If the WSDL appears as XML in the browser, then the rulebase has successfully deployed to the Determinations Server:
6. The WSDL is used by the JAX-WS tool, wsimport to generate a Java client. You can save this page as an xml file so that the Determinations Server does not have to be running when you want to generate your client. Save the current page as **SimpleBenefits_Specific.wsdl**.

2. Compile the client

Compile the client

The JAX-WS wsimport tool can be found in the bin directory of the JAX-WS install. It is called (wsimport.bat for Windows and wsimport.sh for Unix). For detailed instructions on running this tool, see <https://jax-ws.dev.java.net/jax-ws-21-ea2/docs/wsimport.html> Using **wsimport.bat** to create a Java client on Windows.

1. From a command prompt go to the jaxws-ri\bin directory.
2. Run the following command (all one line):
`wsimport -s C:\SimpleBenefits\wsdls\specific\jaxws\javaclient\src -d C:\SimpleBenefits\wsdls\specific\jaxws\javaclient\classes C:\SimpleBenefits\wsdls\specific\SimpleBenefits_specific.wsdl`

The arguments passed to the wsimport program are as follows:

- "-s C:\SimpleBenefits\wsdls\specific\jaxws\javaclient\src" - specifies the output directory for the generated source files. It is very useful to have the .java files that are compiled to create the web service client.
- "-d C:\SimpleBenefits\wsdls\specific\jaxws\javaclient\classes" - specifies the output location for the compiled classes.
- "C:\SimpleBenefits\wsdls\specific\SimpleBenefits_specific.wsdl " - the last argument is the WSDL for which the client is to be generated.

3. Write a program to use the Client

Write a program to use the Client

Now that we have a Java client that runs against the specific wsdl for SimpleBenefits, we can create a program that uses it. The program in this tutorial is a simple command line application that creates an assess request, runs it against a Determinations Server and prints the results.

The program will take in a single optional argument, the end point to send the request to. If no end point is provided, it will default to: <http://localhost:9000/determinations-server9000/assess/soap/specific/10.2/SimpleBenefits>

The entire class SimpleBenefitsSpecificAssessJaxWs is provided in [Appendix 2](#) at the end of this document.

There are several steps to creating a correct assess request using the JAX-WS generated client

Step 1 - Add the JAX-WS libraries to the classpath.

In order to compile and run the program, you will need to add all the jars found in the `jaxws-ri\lib` directory

Step 2 - Import the generated java client code

In order to use the JAX-WS generated code, we need to import it into our class:

```
import com.oracle.determinations.server._10_2.simplebenefits.assess.types.*;
```

Step 3 - Create the assess and the entities that you will need for the request needs

In the code below, we create the assess request, the session and the entities that we intend to use (Session, Global, ListChild – for holding children entities). All these objects exist within the SimpleBenefits rulebase and also exist as XML elements within the service request that we will send. The generated objects are named for the entity and attribute names, which makes them easy to write code for.

If this were a real application, it would probably collect information on the claimant and the children from a user interface, or perhaps load them from a database. To keep things simple, we will just hard-code the values for the claimant and his/her two children.

Each entity instance needs a unique id to identify it so we will set them to "global", "child1" and "child2" respectively. If the data was coming from a database, we would probably use primary keys for the ids of the entities.

```
AssessRequest request = new AssessRequest();  
GlobalInstanceType global = new GlobalInstanceType();
```

```
ChildEntityListType children = new ChildEntityListType();  
global.setListChild(children);
```

```
ChildInstanceType child1 = new ChildInstanceType();  
child1.setId("child1");  
ChildInstanceType child2 = new ChildInstanceType();  
child2.setId("child2");
```

```
children.getChild().add(child1);  
children.getChild().add(child2);
```

Step 4 - Specify the outcomes (answers) that we want the Determinations Server to answer

Before we send the request we need to add the outcomes that we want the Determinations Server to return. In this case, there are three outcomes that we want: is the claimant eligible for the low income allowance, their low income allowance payment and is the claimant eligible for the teenage child allowance.

To request outcomes for these 3 attributes, we add each attribute, and, instead of providing a value, we set the outcome style. This indicates that instead of providing a value we are asking for the Determinations Server to return the value.

In this case we are only interested in the answer, so the outcome style will be value-only.

```
global.setEligibleLowIncomeAllowance(new BooleanAttributeType());
global.getEligibleLowIncomeAllowance()
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);
```

```
global.setLowIncomeAllowancePayment(new NumberAttributeType());
global.getLowIncomeAllowancePayment()
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);
```

```
global.setEligibleTeenageAllowance(new BooleanAttributeType());
global.getEligibleTeenageAllowance()
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);
```

Step 5 - Set attributes and relationships of the entities

Now we will set the values that we know: the claimant's income, whether the claimant a public housing client, and the ages of the children. These are attributes in the rulebase, and also in the generated web client - attributes of the Global entity (for the claimant) and the child entity (the child's age).

```
System.out.println("Setting claimant_income to 13000.00");
global.setClaimantIncome(new NumberAttributeType());
global.getClaimantIncome().setNumberVal(new BigDecimal(13000.00));
```

```
System.out.println("Setting claimant_public_housing_client to true");
global.setClaimantPublicHousingClient(new BooleanAttributeType());
global.getClaimantPublicHousingClient().setBooleanVal(true);
```

```
System.out.println("Setting child_age on child1 to 16");
child1.setChildAge(new NumberAttributeType());
child1.getChildAge().setNumberVal(new BigDecimal(16));
```

```
System.out.println("Setting child_age on child2 to 8");
child2.setChildAge(new NumberAttributeType());
child2.getChildAge().setNumberVal(new BigDecimal(8));
```

Next, there is no need to associate the children with the claimant, via the relationship *claimantschildren* as *claimantschildren* is a containment relationship. We simply need to set the global instance to the request:

```
request.setGlobalInstance(global);
```

Step 6 - Call the assess method

The request is complete and ready to send. We create a specific service instance and call the assess method. This method will return an *AssessResponseDocument*, which should have the outcomes we asked for.

```
OdsAssessServiceSpecific102SimpleBenefits service
```

```
    = new OdsAssessServiceSpecific102SimpleBenefits(new URL(endPoint),  
        SERVICE_QNAME);
```

```
System.out.println("\n--- Request Sent to Determinations Server ---");
```

```
AssessResponse response =
```

```
    service.getOdsAssessServiceSpecific102SimpleBenefitsSOAP().assess(request);
```

```
System.out.println("\n--- Response from Determinations Server ---");
```

The *endPoint* is the one specified when the program is run (supplied as an argument, or default and the *SERVICE_QNAME* is defined statically, at the top of the class as it never changes.

```
public static QName SERVICE_QNAME = new QName(  
    "http://oracle.com/determinations/server/10.2/SimpleBenefits/assess/types",  
    "odsAssessServiceSpecific102SimpleBenefits");
```

```
    "http://oracle.com/determinations/server/10.2/SimpleBenefits/assess/types",  
    "odsAssessServiceSpecific102SimpleBenefits");
```

The Determinations Server must be running, with the SimpleBenefits rulebase deployed at the expected end point for the request to succeed.

Step 7 - Process the response

When the Response document is returned, we can now process it for the outcomes that were in the request. If this were a real application, the outcomes would probably be displayed to the user, or persisted to the database. In this case we will simply print them out.

```
// look for the outcomes
```

```
BooleanAttributeType lowIncomeAllowance = response.getGlobalInstance()  
    .getEligibleLowIncomeAllowance();
```

```
NumberAttributeType lowIncomeAllowancePayment = response.getGlobalInstance()  
    .getLowIncomeAllowancePayment();
```

```
BooleanAttributeType childAllowance = response.getGlobalInstance()  
    .getEligibleTeenageAllowance();
```

```
// print out the results
```

```
System.out.println("\n--- Results ----");
```

```

if (lowIncomeAllowance.isBooleanVal() != null) {
    System.out.println("eligible_low_income_allowance = "
        +lowIncomeAllowance.isBooleanVal());
}
else if (lowIncomeAllowance.getUnknownVal() != null) {
    System.out.println("eligible_low_income_allowance is unknown");
}
else if (lowIncomeAllowance.getUncertainVal() != null) {
    System.out.println("eligible_low_income_allowance is uncertain");
}

if (lowIncomeAllowancePayment.getNumberVal() != null) {
    System.out.println("low_income_allowance_payment = "
        +lowIncomeAllowancePayment.getNumberVal());
}
else if (lowIncomeAllowancePayment.getUnknownVal() != null) {
    System.out.println("low_income_allowance_payment is unknown");
}
else if (lowIncomeAllowancePayment.getUncertainVal() != null) {
    System.out.println("low_income_allowance_payment is uncertain");
}

if (childAllowance.isBooleanVal() != null) {
    System.out.println("eligible_teenage_allowance = "
        +lowIncomeAllowance.isBooleanVal());
}
else if (childAllowance.getUnknownVal() != null) {
    System.out.println("eligible_teenage_allowance is unknown");
}
else if (childAllowance.getUncertainVal() != null) {
    System.out.println("eligible_teenage_allowance is uncertain");
}
}

```

Step 8 - Test the program

When you run the program, you should get the following output printed to standard out. From the response, we can see that all outcomes are known and that the claimant is eligible for both allowances and that the low income allowance payment is 70.0.

```

--- Starting new SimpleBenefitsAssess ---
Creating new Assess request
Setting attribute outcomes for 'eligible_low_income_allowance',
'low_income_allowance_payment' and 'eligible_teenage_allowance'
Setting claimant_income to 13000.00
Setting claimant_public_housing_client to true
Setting child_age on child1 to 16
Setting child_age on child2 to 8

```

--- Request Sent to Determinations Server ----

--- Response from Determinations Server ----

--- Results ----

eligible_low_income_allowance = true
low_income_allowance_payment = 70.0
eligible_teenage_allowance = true

4. Using a Java Client for the Generic WSDL

Using a Java Client for the Generic WSDL

In the tutorial above, we created and used a Java client compiled against the Specific wsdl of the SimpleBenefits rulebase. The procedure for creating a client against the Generic wsdl is an identical, although the Java client will be different, and require different code to achieve the same effect.

To complete this tutorial against the Generic wsdl, you should follow the steps above but with the following differences.

1 - Compile and run SimpleBenefits

Follow the steps outlined in *1 Compile and run the SimpleBenefits rulebase*, but save the Generic wsdl instead of the specific.

Save this wsdl as the file *SimpleBenefits_generic.wsdl*

2 - Compile the client

Follow the steps outlined in *2 Compile the client*, but for the saved generic wsdl instead of the specific. So, the wsimport command will look like:

```
wsimport -s C:\SimpleBenefits\wsdls\generic\jaxws\javaclient\src -d  
C:\SimpleBenefits\wsdls\generic\jaxws\javaclient\classes  
C:\SimpleBenefits\wsdls\generic\SimpleBenefits_generic.wsdl
```

3 - Write a program to use the Client

This is the significantly different part for the Java client generated against the generic wsdl. Although the steps are the same, the code needed to get the same result will be different for the generic java client

1. Add the JAX-WS libraries to the classpath.

As with the specific exampl, you will need to add all the jars found in the jaxws-ri\lib directory. The list of jars is as follows:

activation.jar
FastInfoset.jar
http.jar
jaxb-api.jar

```
jaxb-impl.jar
jaxb-xjc.jar
jaxws-api.jar
jaxws-rt.jar
jaxws-tools.jar
jsr173_api.jar
jsr181-api.jar
jsr250-api.jar
mimepull.jar
resolver.jar
saaj-api.jar
saaj-impl.jar
stax-ex.jar
streambuffer.jar
woodstox.jar
```

2. Import the generated java client code

The generic namespace will be different from the specific namespace. In order to use the JAX-WS generated code, we need to import it into our class:

```
import
com.oracle.determinations.server._10_2.rule-
base.assess.types.OdsAssessServiceGeneric102SimpleBenefits;
```

3. Create the assess and the entities that you will need for the request needs

The code for creating entities is a little different for the generic client. All entities must be put inside their containing entity (except global entity). All the entities must have the proper entity public name as the attribute "id".

From the code below, you can see that you need a few more lines to create the entity instances for the generic Java client.

```
AssessRequest request = new AssessRequest();
GlobalInstanceType global = new GlobalInstanceType();
```

```
EntityType children = new EntityType();
children.setId("child");
global.getEntity().add(children);
```

```
EntityTypeInstance child1 = new EntityTypeInstance();
child1.setId("child1");
EntityTypeInstance child2 = new EntityTypeInstance();
```

```
child2.setId("child2");
```

```
children.getInstance().add(child1);  
children.getInstance().add(child2);
```

4. Specify the outcomes (answers) that we want the Determinations Server to answer

Creating outcomes for the generic client also requires a few more lines of code. We create an AttributeType for each outcome, set the attribute public name as "id" and set its outcome style.

```
AttributeType lowIncomeAllowance = new AttributeType();  
lowIncomeAllowance.setId("eligible_low_income_allowance");  
lowIncomeAllowance  
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);
```

```
AttributeType lowIncomeAllowancePayment = new AttributeType();  
lowIncomeAllowancePayment.setId("low_income_allowance_payment");  
lowIncomeAllowancePayment  
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);
```

```
AttributeType teenageAllowance = new AttributeType();  
teenageAllowance.setId("eligible_teenage_allowance");  
teenageAllowance  
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);
```

```
global.getAttribute().add(lowIncomeAllowance);  
global.getAttribute().add(lowIncomeAllowancePayment);  
global.getAttribute().add(teenageAllowance);
```

5. Set attributes and relationships of the entities

When we create attributes for the generic client we create "AttributeType" objects and set the id for the Attribute to the public name of the rulebase attribute.

```
System.out.println("Setting claimant_income to 13000.00");  
AttributeType claimantIncome = new AttributeType();  
claimantIncome.setId("claimant_income");  
claimantIncome.setNumberVal(new BigDecimal(13000.00));
```

```
System.out.println("Setting claimant_public_housing_client to true");  
AttributeType claimantPHClient = new AttributeType();  
claimantPHClient.setId("claimant_public_housing_client");  
claimantPHClient.setBooleanVal(true);
```

```
global.getAttribute().add(claimantIncome);  
global.getAttribute().add(claimantPHClient);
```

```
System.out.println("Setting child_age on child1 to 16");
AttributeType child1Age = new AttributeType();
child1Age.setId("child_age");
child1Age.setNumberVal(new BigDecimal(16));
child1.getAttribute().add(child1Age);
```

```
System.out.println("Setting child_age on child2 to 8");
AttributeType child2Age = new AttributeType();
child2Age.setId("child_age");
child2Age.setNumberVal(new BigDecimal(8));
child2.getAttribute().add(child2Age);
```

Reference relationships of entities have to be identified in the same way. When we create a Relationship, set the name to the public name of the relationship. In this example, the relationship "climantschildren" is a containment relationship, thus it no longer needs to be declared and set to the request. After all attributes and relationships have been populated, the global entity instance needs to be set to the request.

```
request.setGlobalInstance(global);
```

6. Call the assess method

Calling the assess method in the generic client is almost identical to the specific method, although the names are different.

```
OdsAssessServiceGeneric102SimpleBenefits service = new
OdsAssessServiceGeneric102SimpleBenefits
(new URL(endPoint), SERVICE_QNAME);
```

```
System.out.println("\n--- Request Sent to Determinations Server ---");
AssessResponse response = service.getOdsAssessServiceGeneric102SimpleBenefitsSOAP()
.assess(request);
```

7. Process the response

Once the response has been returned by the rulebase, we need to process the response to get the answers to the questions that we asked. This requires a little more code in the generic format because the generic XML does not distinguish between different types of entities, and it does not have specific names for the attributes we need to get.

However, by adding a simple method to look for the attributes that we need, we can simplify the code. For details on the very simple methods `getAttribute`, see the full listing of the code in the Appendix below.

```

System.out.println("\n--- Results ----");
if (lowIncomeAllowanceResp.isBooleanVal() != null) {
    System.out.println("eligible_low_income_allowance = "
        + lowIncomeAllowanceResp.isBooleanVal());
}
else if (lowIncomeAllowanceResp.getUnknownVal() != null) {
    System.out.println("eligible_low_income_allowance is unknown");
}
else if (lowIncomeAllowanceResp.getUncertainVal() != null) {
    System.out.println("eligible_low_income_allowance is uncertain");
}
if (teenageAllowanceResp.isBooleanVal() != null) {
    System.out.println("eligible_teenage_allowance = "
        + teenageAllowanceResp.isBooleanVal());
}
else if (teenageAllowanceResp.getUnknownVal() != null) {
    System.out.println("eligible_teenage_allowance is unknown");
}
else if (teenageAllowanceResp.getUncertainVal() != null) {
    System.out.println("eligible_teenage_allowance is uncertain");
}
if (lowIncomeAllowancePaymentResp.getNumberVal() != null) {
    System.out.println("low_income_allowance_payment = "
        + lowIncomeAllowancePaymentResp.getNumberVal());
}
else if (lowIncomeAllowancePaymentResp.getUnknownVal() != null) {
    System.out.println("low_income_allowance_payment is unknown");
}
else if (lowIncomeAllowancePaymentResp.getUncertainVal() != null) {
    System.out.println("low_income_allowance_payment is uncertain");
}
}

```

Step 4 - Test the program

When you run the generic version of this simple program, you should get the following output printed to standard out. From the response, the results are exactly the same as the specific program.

```

--- Starting new SimpleBenefitsAssess (Generic) ---
Creating new Assess request
Setting attribute outcomes for 'eligible_low_income_allowance',
'low_income_allowance_payment' and 'eligible_teenage_allowance'
Setting claimant_income to 13000.00
Setting claimant_public_housing_client to true
Setting child_age on child1 to 16

```

Setting child_age on child2 to 8

--- Request Sent to Determinations Server ----

--- Response from Determinations Server ----

--- Results ----

eligible_low_income_allowance = true

eligible_teenage_allowance = true

low_income_allowance_payment = 70.0

5. Generic vs. Specific WSDL

Generic vs. Specific WSDL

As you can see for both examples you can follow the same steps to generate and use an JAX-WS Java client for a rulebase deployed on the Determinations Server. You can use either client to achieve the desired operation.

The major difference between the specific and the generic client is ease of use versus maintainability. The specific format is easier to use and much less prone to error, because attributes and relationships have specific names within the rulebase. You cannot accidentally misname an attribute or a relationship using the specific format. The disadvantage with the Specific format is that adding, removing or renaming attributes and relationships will require you to regenerate the Java client using the JAX-WS wsimport tool.

The interface generated for the generic client however, can be used for any rulebase, and never needs to be recompiled. Its disadvantage is that it is easier to make mistakes with the names of attributes and relationships and the code is somewhat more cumbersome.

6. Appendix 1 – Glossary of terms

Appendix 1 – Glossary of terms

End Point – An address that can be used to communicate with a web service. For this Tutorial the end point is the location of the SimpleBenefits rulebase when deployed to the Oracle Determinations Server.

JAX-WS – Java API for XML-Based Web Services. A standard defined by JSR 224 (<http://jcp.org/en/jsr/detail?id=224>). This tutorial uses the JAX-WS Reference implementation to generate a Web Service client.

Oracle Determinations Server – A web application which provides Oracle Policy Automation services as a web service.

Rulebase – A compiled rule project authored in Oracle Policy Modeling.

URL – Uniform Resource Locator. A global address for documents and services on the World Wide Web

Web Service – A service provided over the Web. Typically using XML and SOAP.

WSDL – Web Service Description Language. A standard way of describing Web Services that use XML and SOAP

7. Appendix 2 - SimpleBenefitsSpecificAssessJaxWS Class

Appendix 2 - SimpleBenefitsSpecificAssessJaxWS Class

```
import java.math.BigDecimal;
```

```
import java.net.URL;
```

```
import javax.xml.namespace.QName;
```

```

import com.oracle.determinations.server._10_2.simplebenefits.assess.types.AssessRequest;
import com.oracle.determinations.server._10_2.simplebenefits.assess.types.AssessResponse;
import com.oracle.determinations.server._10_2.simplebenefits.assess.types.BooleanAttributeType;
import com.oracle.determinations.server._10_2.simplebenefits.assess.types.ChildEntityListType;
import com.oracle.determinations.server._10_2.simplebenefits.assess.types.ChildInstanceType;
import com.oracle.determinations.server._10_2.simplebenefits.assess.types.GlobalInstanceType;
import com.oracle.determinations.server._10_2.simplebenefits.assess.types.NumberAttributeType;
import
    com.oracle.determinations.server._10_2.simplebenefits.assess.types.OdsAssessServiceSpecific102SimpleBenefits;
import com.oracle.determinations.server._10_2.simplebenefits.assess.types.OutcomeStyleEnum;

public class SimpleBenefitsSpecificAssessJaxWs
{
    public static String DEFAULT_ENDPOINT
        = "http://localhost:9000/determinations-server9000/assess/soap/specific/10.2/SimpleBenefits";

    public static QName SERVICE_QNAME = new QName(
        "http://oracle.com/determinations/server/10.2/SimpleBenefits/assess/types",
        "odsAssessServiceSpecific102_SimpleBenefits");

    public static void main(String[] args) {
        String endPoint = args.length > 0 ? args[0] : DEFAULT_ENDPOINT;

        try {
            System.out.println("--- Starting new SimpleBenefitsAssess ---");

            System.out.println("Creating new Assess request");

            AssessRequest request = new AssessRequest();
            GlobalInstanceType global = new GlobalInstanceType();

            ChildEntityListType children = new ChildEntityListType();
            global.setListChild(children);

            ChildInstanceType child1 = new ChildInstanceType();
            child1.setId("child1");
            ChildInstanceType child2 = new ChildInstanceType();
            child2.setId("child2");

            children.getChild().add(child1);
            children.getChild().add(child2);

            System.out.println("Setting attribute outcomes for "
                + "'eligible_low_income_allowance',"
                + "'low_income_allowance_payment'"

```

```

        +" and 'eligible_teenage_allowance'");

global.setEligibleLowIncomeAllowance(new BooleanAttributeType());
global.getEligibleLowIncomeAllowance()
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);

global.setLowIncomeAllowancePayment(new NumberAttributeType());
global.getLowIncomeAllowancePayment()
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);

global.setEligibleTeenageAllowance(new BooleanAttributeType());
global.getEligibleTeenageAllowance()
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);

System.out.println("Setting claimant_income to 13000.00");
global.setClaimantIncome(new NumberAttributeType());
global.getClaimantIncome().setNumberVal(new BigDecimal(13000.00));

System.out.println("Setting claimant_public_housing_client to true");
global.setClaimantPublicHousingClient(new BooleanAttributeType());
global.getClaimantPublicHousingClient().setBooleanVal(true);

System.out.println("Setting child_age on child1 to 16");
child1.setChildAge(new NumberAttributeType());
child1.getChildAge().setNumberVal(new BigDecimal(16));

System.out.println("Setting child_age on child2 to 8");
child2.setChildAge(new NumberAttributeType());
child2.getChildAge().setNumberVal(new BigDecimal(8));

request.setGlobalInstance(global);

OdsAssessServiceSpecific102SimpleBenefits service
    = new OdsAssessServiceSpecific102SimpleBenefits(new URL(endPoint),SERVICE_QNAME);

System.out.println("\n--- Request Sent to Determinations Server ----");
AssessResponse response = service.getOdsAssessServiceSpecific102SimpleBenefitsSOAP().assess(request);

System.out.println("\n--- Response from Determinations Server ----");

// look for the outcomes
BooleanAttributeType lowIncomeAllowance = response.getGlobalInstance()
    .getEligibleLowIncomeAllowance();

```

```

NumberAttributeType lowIncomeAllowancePayment = response.getGlobalInstance()
    .getLowIncomeAllowancePayment();

BooleanAttributeType childAllowance = response.getGlobalInstance().getEligibleTeenageAllowance();

// print out the results
System.out.println("\n--- Results ----");
if (lowIncomeAllowance.isBooleanVal() != null) {
    System.out.println("eligible_low_income_allowance = "
        +lowIncomeAllowance.isBooleanVal());
}
else if (lowIncomeAllowance.getUnknownVal() != null) {
    System.out.println("eligible_low_income_allowance is unknown");
}
else if (lowIncomeAllowance.getUncertainVal() != null) {
    System.out.println("eligible_low_income_allowance is uncertain");
}

if (lowIncomeAllowancePayment.getNumberVal() != null) {
    System.out.println("low_income_allowance_payment = "
        +lowIncomeAllowancePayment.getNumberVal());
}
else if (lowIncomeAllowancePayment.getUnknownVal() != null) {
    System.out.println("low_income_allowance_payment is unknown");
}
else if (lowIncomeAllowancePayment.getUncertainVal() != null) {
    System.out.println("low_income_allowance_payment is uncertain");
}

if (childAllowance.isBooleanVal() != null) {
    System.out.println("eligible_teenage_allowance = "
        +lowIncomeAllowance.isBooleanVal());
}
else if (childAllowance.getUnknownVal() != null) {
    System.out.println("eligible_teenage_allowance is unknown");
}
else if (childAllowance.getUncertainVal() != null) {
    System.out.println("eligible_teenage_allowance is uncertain");
}
}
catch(Exception e) {
    System.out.println("An error occurred"+ e.getMessage());
    e.printStackTrace();
}
}

```

```
}  
}
```

8. Appendix 3 - SimpleBenefitsGenericAssessJaxWS Class

Appendix 3 - SimpleBenefitsGenericAssessJaxWS Class

```
import java.math.BigDecimal;  
import java.net.URL;  
  
import javax.xml.namespace.QName;  
  
import com.oracle.determinations.server._10_2.rulebase.assess.types.AssessRequest;  
import com.oracle.determinations.server._10_2.rulebase.assess.types.AssessResponse;  
import com.oracle.determinations.server._10_2.rulebase.assess.types.AttributeType;  
import com.oracle.determinations.server._10_2.rulebase.assess.types.EntityInstanceType;  
import com.oracle.determinations.server._10_2.rulebase.assess.types.EntityType;  
import com.oracle.determinations.server._10_2.rulebase.assess.types.GlobalInstanceType;  
import com.oracle.determinations.server._10_2.rulebase.assess.types.OdsAssessServiceGeneric102SimpleBenefits;  
import com.oracle.determinations.server._10_2.rulebase.assess.types.OutcomeStyleEnum;  
  
public class SimpleBenefitsGenericAssessJaxWs  
{  
  
    public static String DEFAULT_ENDPOINT  
        = "http://localhost:9000/determinations-server9000/assess/soap/generic/10.2/SimpleBenefits";  
    public static QName SERVICE_QNAME = new QName(  
        "http://oracle.com/determinations/server/10.2/rulebase/assess/types",  
        "odsAssessServiceGeneric102_SimpleBenefits");  
  
    public static void main(String[] args) {  
        String endPoint = args.length > 0 ? args[0] : DEFAULT_ENDPOINT;  
  
        try {  
            System.out.println("--- Starting new SimpleBenefitsAssess (Generic) ---");  
  
            System.out.println("Creating new Assess request");  
  
            AssessRequest request = new AssessRequest();  
            GlobalInstanceType global = new GlobalInstanceType();  
  
            EntityType children = new EntityType();  
            children.setId("child");  
            global.getEntity().add(children);  
  
            EntityInstanceType child1 = new EntityInstanceType();
```

```
child1.setId("child1");
EntityInstanceType child2 = new EntityInstanceType();
child2.setId("child2");

children.getInstance().add(child1);
children.getInstance().add(child2);

System.out.println("Setting attribute outcomes for "
    + "'eligible_low_income_allowance',"
    + "'low_income_allowance_payment'"
    + " and 'eligible_teenage_allowance'");

AttributeType lowIncomeAllowance = new AttributeType();
lowIncomeAllowance.setId("eligible_low_income_allowance");
lowIncomeAllowance
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);

AttributeType lowIncomeAllowancePayment = new AttributeType();
lowIncomeAllowancePayment.setId("low_income_allowance_payment");
lowIncomeAllowancePayment
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);

AttributeType teenageAllowance = new AttributeType();
teenageAllowance.setId("eligible_teenage_allowance");
teenageAllowance
    .setOutcomeStyle(OutcomeStyleEnum.VALUE_ONLY);

global.getAttribute().add(lowIncomeAllowance);
global.getAttribute().add(lowIncomeAllowancePayment);
global.getAttribute().add(teenageAllowance);

System.out.println("Setting claimant_income to 13000.00");
AttributeType claimantIncome = new AttributeType();
claimantIncome.setId("claimant_income");
claimantIncome.setNumberVal(new BigDecimal(13000.00));

System.out.println("Setting claimant_public_housing_client to true");
AttributeType claimantPHClient = new AttributeType();
claimantPHClient.setId("claimant_public_housing_client");
claimantPHClient.setBooleanVal(true);

global.getAttribute().add(claimantIncome);
global.getAttribute().add(claimantPHClient);

System.out.println("Setting child_age on child1 to 16");
AttributeType child1Age = new AttributeType();
```

```

child1Age.setId("child_age");
child1Age.setNumberVal(new BigDecimal(16));
child1.getAttribute().add(child1Age);

System.out.println("Setting child_age on child2 to 8");
AttributeType child2Age = new AttributeType();
child2Age.setId("child_age");
child2Age.setNumberVal(new BigDecimal(8));
child2.getAttribute().add(child2Age);

request.setGlobalInstance(global);

OdsAssessServiceGeneric102SimpleBenefits service = new OdsAssessServiceGeneric102SimpleBenefits
(new URL(endPoint), SERVICE_QNAME);

System.out.println("\n--- Request Sent to Determinations Server ----");
AssessResponse response = service.getOdsAssessServiceGeneric102SimpleBenefitsSOAP()
.assess(request);

System.out.println("\n--- Response from Determinations Server ----");

// look for the outcomes
GlobalInstanceType globalResp = response.getGlobalInstance();

AttributeType lowIncomeAllowanceResp
    = getAttribute(globalResp, "eligible_low_income_allowance");

AttributeType lowIncomeAllowancePaymentResp
    = getAttribute(globalResp, "low_income_allowance_payment");

AttributeType teenageAllowanceResp
    = getAttribute(globalResp, "eligible_teenage_allowance");

// print out the results
System.out.println("\n--- Results ----");
if (lowIncomeAllowanceResp.isBooleanVal() != null) {
    System.out.println("eligible_low_income_allowance = "
        + lowIncomeAllowanceResp.isBooleanVal());
}
else if (lowIncomeAllowanceResp.getUnknownVal() != null) {
    System.out.println("eligible_low_income_allowance is unknown");
}
else if (lowIncomeAllowanceResp.getUncertainVal() != null) {
    System.out.println("eligible_low_income_allowance is uncertain");
}
if (teenageAllowanceResp.isBooleanVal() != null) {

```

```

        System.out.println("eligible_teenage_allowance = "
            +teenageAllowanceResp.isBooleanVal());
    }
    else if (teenageAllowanceResp.getUnknownVal() != null) {
        System.out.println("eligible_teenage_allowance is unknown");
    }
    else if (teenageAllowanceResp.getUncertainVal() != null) {
        System.out.println("eligible_teenage_allowance is uncertain");
    }
    if (lowIncomeAllowancePaymentResp.getNumberVal() != null) {
        System.out.println("low_income_allowance_payment = "
            +lowIncomeAllowancePaymentResp.getNumberVal());
    }
    else if (lowIncomeAllowancePaymentResp.getUnknownVal() != null) {
        System.out.println("low_income_allowance_payment is unknown");
    }
    else if (lowIncomeAllowancePaymentResp.getUncertainVal() != null) {
        System.out.println("low_income_allowance_payment is uncertain");
    }
}
catch(Exception e) {
    System.out.println("An error occurred"+ e.getMessage());
    e.printStackTrace();
}
}

// Go through the list of attributes for the global entity and return the
// attribute that matches the attributeId.
private static AttributeType getAttribute(GlobalInstanceType globalResp, String attributeId) {
    AttributeType attribute = null;

    for (int i=0; i < globalResp.getAttribute().size(); i++) {
        if (attributeId.equals(globalResp.getAttribute().get(i).getId())) {
            attribute = globalResp.getAttribute().get(i);
            break;
        }
    }
    return attribute;
}
}

```

Tutorial: Create and use a C# client for Oracle Determinations Server

The advantage of using a rigorous and well defined interface, such as a WS-I compliant web service, is the ability to integrate it with other applications using tools to aid that integration. Using HTTP as the communication layer and XML as the request and response formats gives a systems integrator an industry standard way of communicating with the choice of many tools to build that integration.

This tutorial is a quick walkthrough of the direct integration of the Determinations Server running a rulebase with a simple .NET application (written in C Sharp). Integration between the application and the Web Service is handled by a client, automatically generated by Visual Studio 2008. The generated web service client will perform the job of creating the request as a web service call, making the call, and interpreting the response.



Requirements to follow the tutorial

This tutorial discusses programming in C#. You should be familiar with C#, and Visual Studio.

The following software is required to follow this tutorial.

- Oracle Policy Modeling 10.2 or above
- The SimpleBenefits rulebase (located at [examples\rulebases\compiled\SimpleBenefits.zip](#)) in the Oracle Policy Automation Runtime package.
- Visual Studio (this tutorial uses VS 2008, but other versions should also work).
- .NET SDK (which should be installed with Visual Studio). .NET versions 2.0 or later will work with this tutorial.

This tutorial uses Visual Studio to generate a Service Reference. For more information on how to add a reference to a web service in Visual Studio, see <http://msdn.microsoft.com/en-us/library/bb628649.aspx>

1. Compile and run the SimpleBenefits rulebase

Compile and run the SimpleBenefits rulebase

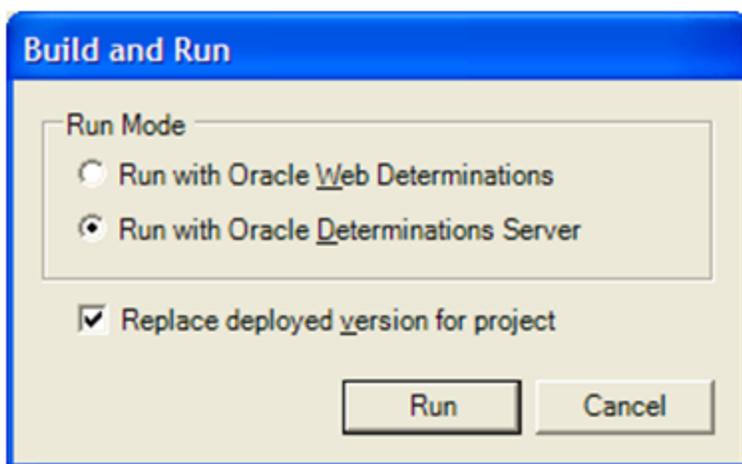
The first thing to do is to have a look at the SimpleBenefits rulebase in Oracle Policy Modeling. This rulebase is a very simple example. It determines the following 3 goals:

Is the claimant eligible for the low income allowance?
What is the claimant's low income allowance amount?
Is the claimant eligible for the teenage child allowance?

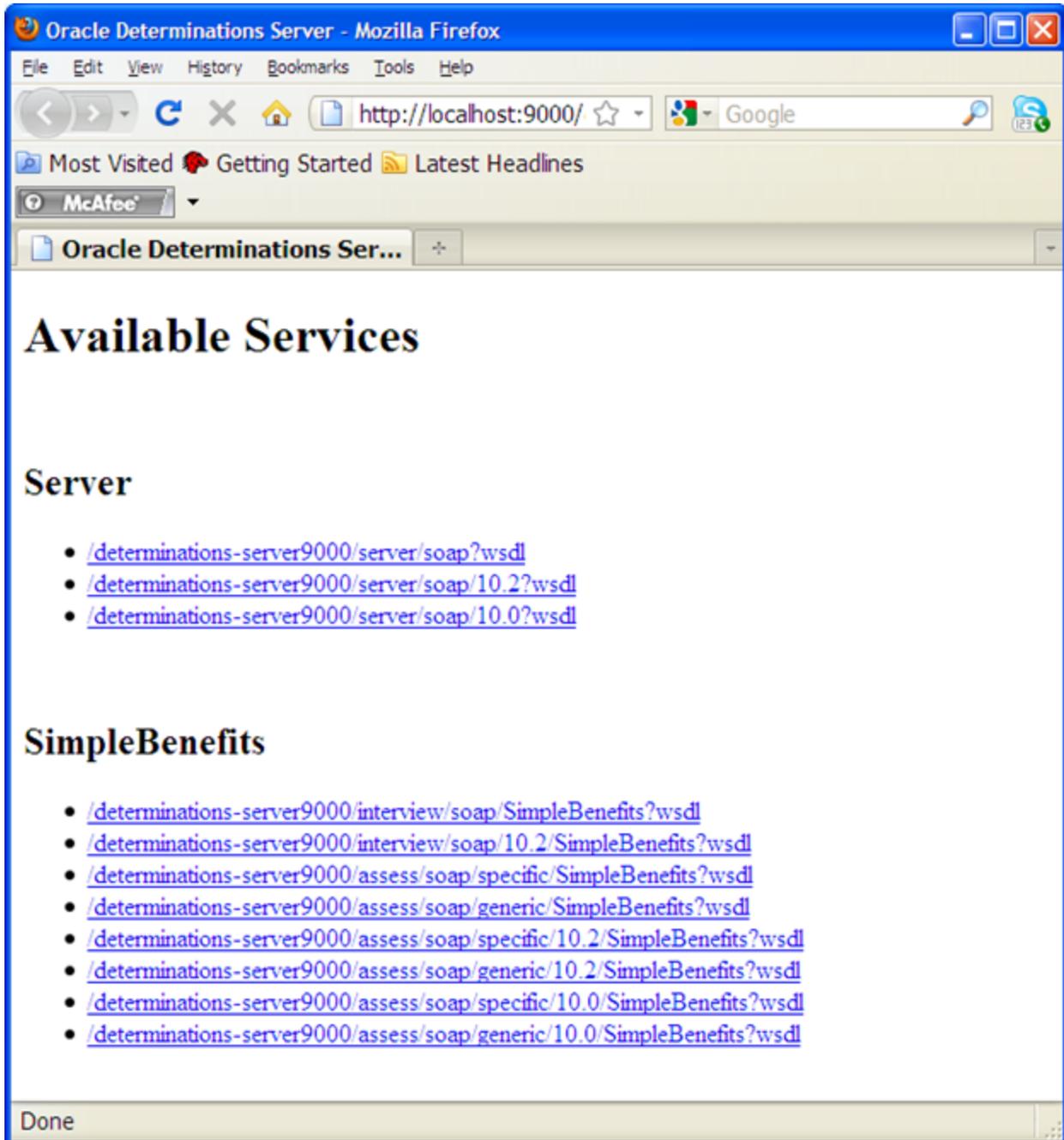
You can compile and run this rulebase in the Determinations Server from Oracle Policy Modeling by the following method:

1. From the **Build** menu, choose **Build and Run**
2. When the *Build and Run Dialog* appears, choose *Run with Oracle Determinations Server*; also select *Replace deployed version for project*

Note: Oracle Policy Modeling may require you to recompile the rule documents of the project and if it is the case, proceed by selecting "Compile and Continue".



3. After a brief pause, Oracle Policy Modeling should start up your default web browser with the default page for the Determinations Server.

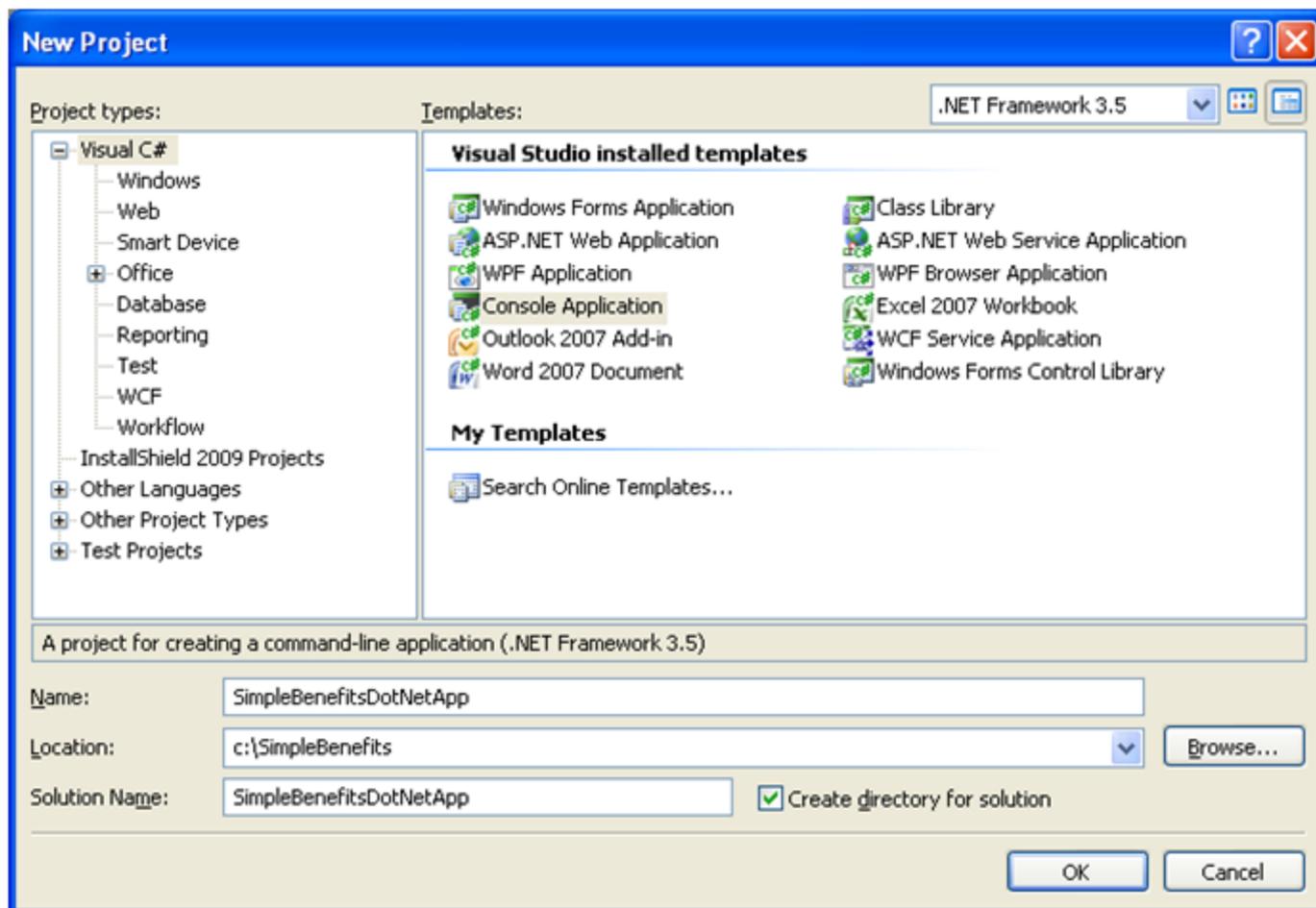


4. At this point, Oracle Determinations Server will display links for the server's WSDLs and links for the rulebase's WSDLs. For this tutorial, we are interested with the specific WSDL of the "SimpleBenefits" rulebase. You can view the specific rulebase WSDL by clicking on the link:

</determinations-server9000/assess/soap/specific/SimpleBenefits?wsdl>

2. Open Visual Studio and create a new C# Console Application

Open Visual Studio and create a new C# Console Application



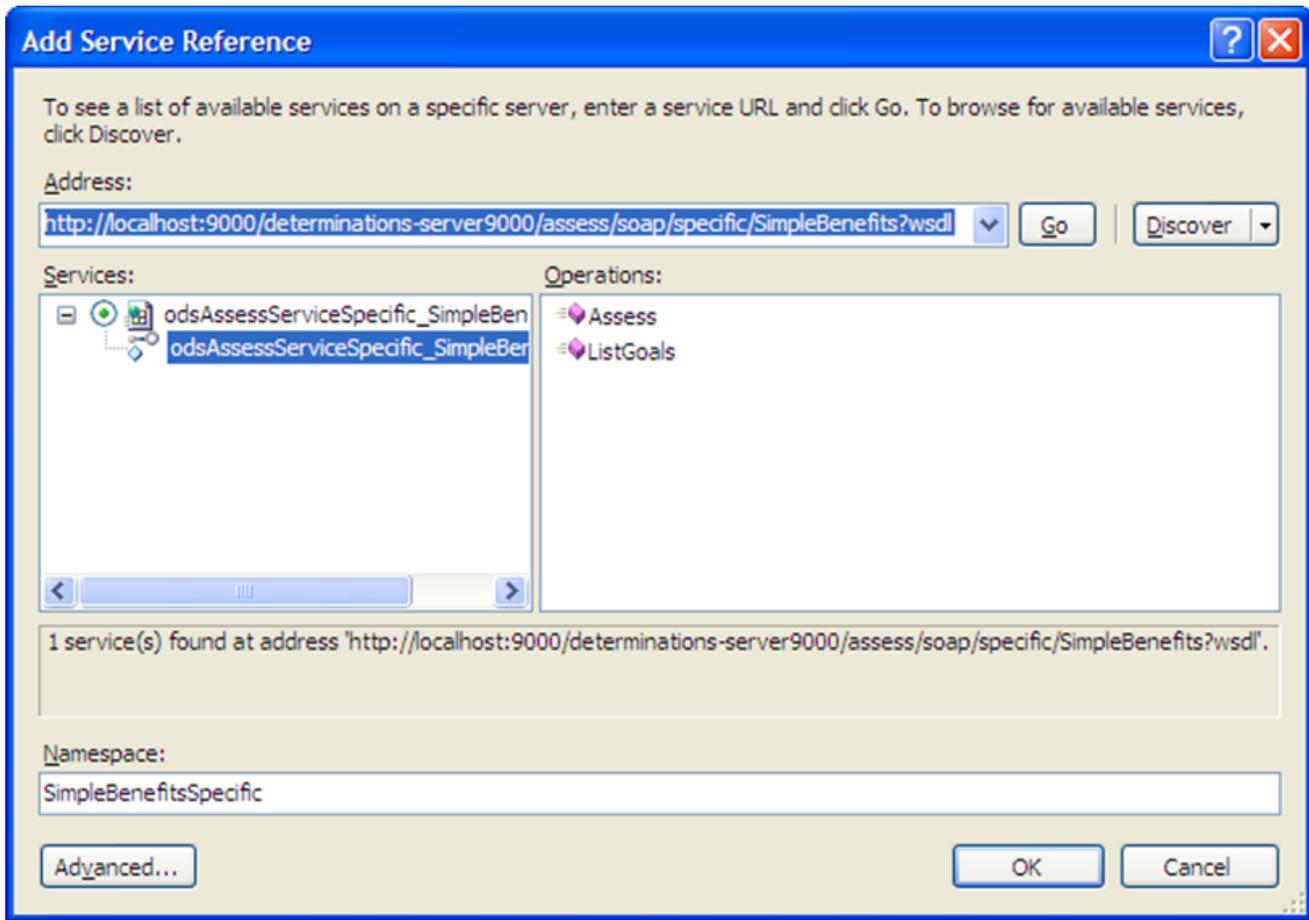
3. Add a Service Reference for the Determinations Server Specific Client

Add a Service Reference for the Determinations Server Specific Client

Visual Studio will create Web Service Client generated from a WSDL. The best way to do this is to do the following:

1. Make sure that the service is running. In this case, make sure that Oracle Policy Modeling is running the Determinations Server.
2. From the **Project** Menu, choose **Add Service Reference...** You can also add a Service Reference by Right Clicking on the Project in the Solution Explorer and choosing Add Service Reference...
3. In the Add Service Reference dialog box, type in the URL for the Simple Benefits specific Web Service. If it is running from Oracle Policy Modeling, this should be:

<http://localhost:9000/determinations-server9000/assess/soap/specific/SimpleBenefits?wsdl>



4. Use the Service Reference

Use the Service Reference

Now that we have a Reference to the SimpleBenefits rulebase, we can use it in the program. The program in this tutorial will create an assess request, run it against a Determinations Server and prints the results.

The entire class SimpleBenefitsComandLine is provided in Appendix 1 at the end of this document.

There are several steps in writing the program.

Step 1 - Import the Service Reference classes

In order to use the SimpleBenefits Service Reference client generated code, we need to import it into our class.
using `SimpleBenefitsDotNetApp.SimpleBenefitsSpecific;`

Step 2 - Create the assess and the entities that you will need for the request needs

In the code below, we create the assess request, the session and the entities that we intend to use (session, global, listchild – for holding child entities). All these objects exist within the SimpleBenefits rulebase and also exist as XML elements within the service request that we will send. We will use objects generated as part of the service reference,

which match the XML of the request that we need to send./

If this were a real application, it would probably collect information on the claimant and the children from a user interface, or perhaps load them from a database. To keep things simple, we will just hard-code the values for the claimant and his/her two children.

Each entity instance needs a unique id to identify it. If the data was coming from a database, we would probably use primary keys for the ids of the entities, but, again, to keep things simple, we will set them to "global", "child1" and "child2" respectively.

```
AssessRequest request = new AssessRequest();

request.globalinstance = new GlobalInstanceType();

request.globalinstance.listchild = new childEntityListType();

childInstanceType child1 = new childInstanceType();
child1.id = "child1";

childInstanceType child2 = new childInstanceType();
child2.id = "child2";

request.globalinstance.listchild.child = new childInstanceType[] { child1, child2 };
```

Step 3 - Specify the outcomes (answers) that we want the Determinations Server to answer

Before we send the request we need to add the outcomes that we want the Determinations Server to return. In this case, there are three outcomes that we want:

- Is the claimant eligible for the low income allowance?
- The claimants low income allowance payment
- Is the claimant eligible for the teenage child allowance?

To request outcomes for these 3 attributes, we add each attribute, and, instead of providing a value, we set the outcome style. This indicates that instead of providing a value we are asking for the Determinations Server to return the value.

In this case we are only interested in the answer, so the outcome style will be value-only.

```
GlobalInstanceType g = request.globalinstance;

g.eligible_low_income_allowance = new booleanAttributeType();
g.eligible_low_income_allowance.outcomestyle = OutcomeStyleEnum.valueonly;
g.eligible_low_income_allowance.outcomestyleSpecified = true;

g.low_income_allowance_payment = new numberAttributeType();
g.low_income_allowance_payment.outcomestyle = OutcomeStyleEnum.valueonly;
g.low_income_allowance_payment.outcomestyleSpecified = true;

g.eligible_teenage_allowance = new booleanAttributeType();
```

```
g.eligible_teenage_allowance.outcomestyle = OutcomeStyleEnum.valueonly;
g.eligible_teenage_allowance.outcomestyleSpecified = true;
```

Note:

because the attribute "outcome-style" is an optional attribute, when we set this attribute, we also need to indicate that we have specified the attribute by setting the property outcomestyleSpecified to true.

Step 4 - Set attributes of the entities

Now we will set the values that we know: the claimant's income, whether the claimant is a public housing client and the ages of the children. These are attributes in the Rulebase, and also in the generated service reference - attributes of the Global entity (for the claimant) and the child entity (the child's age).

```
g.claimant_income = new numberAttributeType();
g.claimant_income.Item = new Decimal(13000.00);
```

```
g.claimant_public_housing_client = new booleanAttributeType();
g.claimant_public_housing_client.Item = true;
```

```
child1.child_age = new numberAttributeType();
child1.child_age.Item = new Decimal(16);
```

```
child2.child_age = new numberAttributeType();
child2.child_age.Item = new Decimal(8);
```

Step 5 - Call the assess method

The request is complete and ready to send. We create a specific service instance and call the assess method.

```
odsAssessServiceSpecific_SimpleBenefits_typeClient service
= new odsAssessServiceSpecific_SimpleBenefits_typeClient();
AssessResponse response = service.Assess(request);
```

Step 6 - Process the response

When the Response document is returned, we can now process it for the outcomes that were in the request. If this were a real application, the outcomes would probably be displayed to the user, or persisted to the database. In this case we will simply print them out.

```
booleanAttributeType lowIncomeAllowance = response.globalinstance.eligible_low_income_allowance;

numberAttributeType lowIncomeAllowancePayment = response.globalinstance.low_income_allowance_payment;

booleanAttributeType teenageAllowance = response.globalinstance.eligible_teenage_allowance;

Console.WriteLine("\n--- Results ----");
```

```

if (lowIncomeAllowance.Item is Boolean)
{
    Console.WriteLine("low_income_allowance_payment = "
        + lowIncomeAllowance.Item);
}
else if (lowIncomeAllowance.Item is UnknownValue)
{
    Console.WriteLine("low_income_allowance_payment is unknown");
}
else if (lowIncomeAllowance.Item is UncertainValue)
{
    Console.WriteLine("low_income_allowance_payment is uncertain");
}
if (lowIncomeAllowancePayment.Item is Decimal)
{
    Console.WriteLine("eligible_low_income_allowance = "
        + lowIncomeAllowancePayment.Item);
}
else if (lowIncomeAllowancePayment.Item is UnknownValue)
{
    Console.WriteLine("eligible_low_income_allowance is unknown");
}
else if (lowIncomeAllowancePayment.Item is UncertainValue)
{
    Console.WriteLine("eligible_low_income_allowance is uncertain");
}
if (teenageAllowance.Item is Boolean)
{
    Console.WriteLine("eligible_teenage_allowance = "
        + teenageAllowance.Item);
}
else if (teenageAllowance.Item is UnknownValue)
{
    Console.WriteLine("eligible_teenage_allowance is unknown");
}
else if (teenageAllowance.Item is UncertainValue)
{
    Console.WriteLine("eligible_teenage_allowance is uncertain");
}
}

```

Step 7 - Test the program

When you run the program you should get the following output printed to standard out.

From the response, we can see that all outcomes are known and that the claimant is eligible for both allowances and that the low income allowance payment is 70.0.

```
---- Starting new SimpleBenefitsAssess (Specific) ----
Creating new Assess request
Setting attribute outcomes for 'eligible_low_income_allowance',
' low_income_allowance_payment', and 'eligible_teenage_allowance'
Setting claimant_income to 13000.00
Setting claimant_public_housing_client to true
Setting child_age on child1 to 16
Setting child_age on child2 to 8
```

```
---- Request Sent to Oracle Determinations Server ----
```

```
---- Response from Oracle Determinations Server ----
```

```
--- Results ----
low_income_allowance_payment = True
eligible_low_income_allowance = 70.0
eligible_teenage_allowance = True
```

5. Running against a different endpoint

Running against a different endpoint

The endpoint of your rulebase running in the Oracle Determination Server will change when you move it to a different server, or you change some other aspect of the deployment like the port the Determinations Server runs on. For example, you may write your integration code against the Determinations Server deployed in a test environment, and, of course, you want your code to run against the production deployment when it is released.

The endpoint for a Service Reference is stored in the app.config file for the program. In Visual Studio, you should be able to see a file in your project called app.config. If you open that file, you should be able to a section starting with the client element.

In the endpoint element, you should be able to see details of the service endpoint, including the URL to run against. You can change this XML to match the Determinations Server that you want to run against.

You should only need to change the address attribute of the endpoint element

```
<client>
  <endpoint address="http://localhost:9000/determinations-server9000/assess/soap/generic/SimpleBenefits"
    binding="basicHttpBinding"
    bindingConfiguration="odsAssessServiceGeneric_SimpleBenefits"
    contract="SimpleBenefitsGeneric.odsAssessServiceGeneric_SimpleBenefits_type"
    name="odsAssessServiceGeneric_SimpleBenefits_SOAP" />
</client>
```

6. Using a Service Reference for the Generic WSDL

Using a Service Reference for the Generic WSDL

In the tutorial above, we created and used a Web Service Reference client compiled against the Specific wsdl of the SimpleBenefits rulebase. The procedure for creating a client against the Generic wsdl is an identical one, although the Web Service Reference client will be different, and require different code to achieve the same effect.

To complete this tutorial against the Generic wsdl, you should follow the steps above but with the following differences.

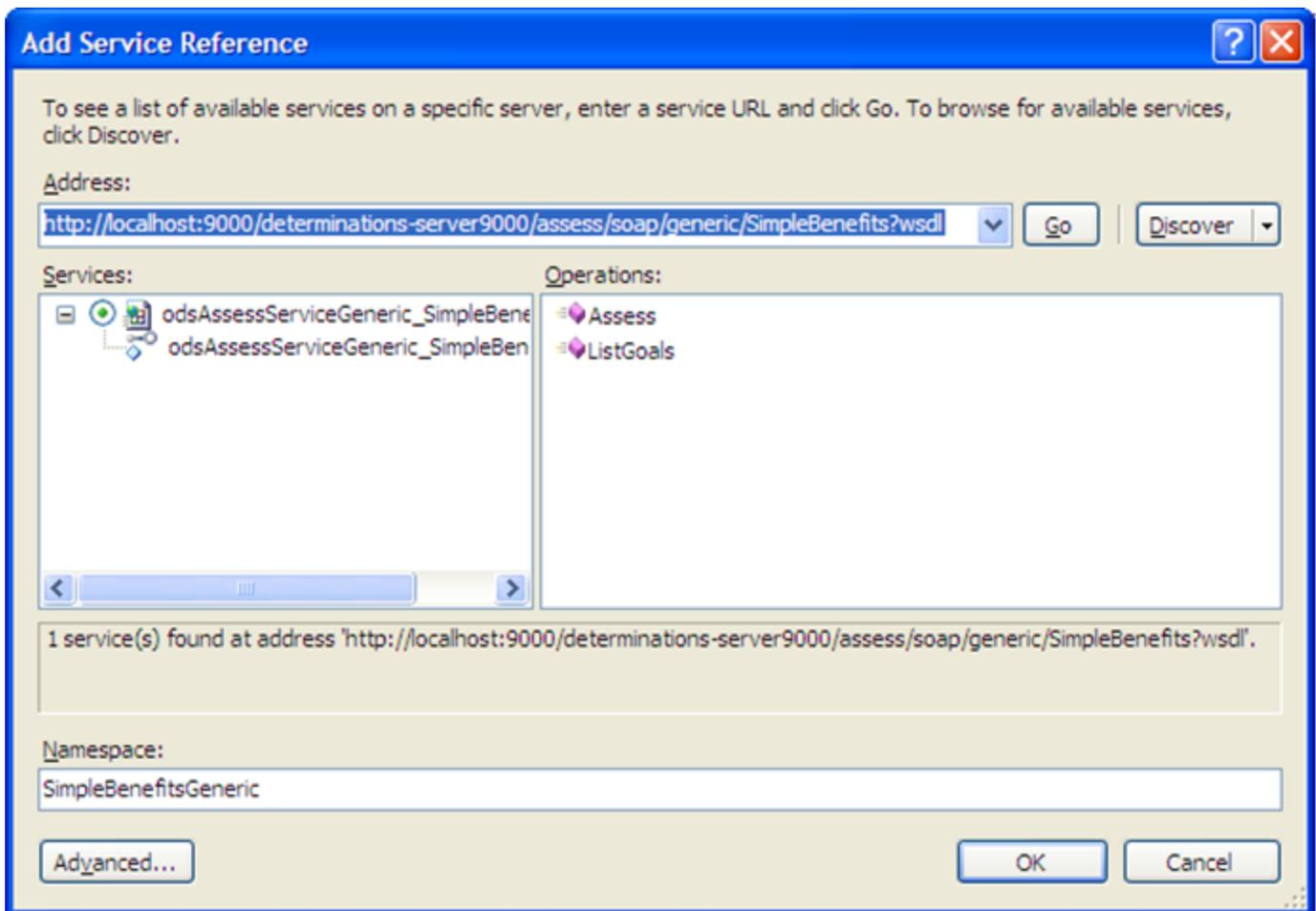
1. Compile and run SimpleBenefits

Follow the steps outlined in 1 Compile and run SimpleBenefits, but instead of clicking the specific WSDL, click the generic WSDL, which has the form `/<determinations-server-url><port>/assess/soap/specific/<rulebase name>?wsdl`, or, in the case of this example:

`/determinations-server9000/assess/soap/generic/SimpleBenefits?wsdl`

2. Create a Service Reference to the Generic

Follow the steps outlined in 3 Add a Service Reference for the Determinations Server Specific Client, but for the generic wsdl instead of the specific.



3. Write a program to use the Service Reference

This is the significantly different part for the Reference generated against the generic service. Although the steps are the same, the code needed to get the same result will be different.

1. Import the generated java client code

The generic namespace will be different from the specific namespace. In order to use the JAX-WS generated code, we need to import it into our class

```
using SimpleBenefitsDotNetApp.SimpleBenefitsGeneric;  
using Attribute=SimpleBenefitsDotNetApp.SimpleBenefitsGeneric.AttributeType;
```

2. Create the assess and the entities that you will need for the request needs

The code for creating entity instances is a little different for the generic service. All entity instances must be contained into the entity's instance array. And the entity id should always be populated.

From the code below, you can see that you need a few more lines to create the entity instances for the generic service.

```
AssessRequest request = new AssessRequest();  
request.globalinstance = new GlobalInstanceType();  
  
EntityType childEntity = new EntityType();  
childEntity.id = "child";  
  
request.globalinstance.entity = new EntityType[] { childEntity };  
  
EntityTypeInstance child1 = new EntityTypeInstance();  
child1.id = "child1";  
  
EntityTypeInstance child2 = new EntityTypeInstance();  
child2.id = "child2";  
  
childEntity.instance = new EntityTypeInstance[] { child1, child2 };
```

3. Specify the outcomes (answers) that we want the Determinations Server to answer

Creating outcomes for the generic client also requires a few more lines of code. It requires creating new instances of Attribute and setting the id, outcomestyle and outcomestyleSpecified properties.

```
Attribute eligible_low_income_allowance = new Attribute();  
eligible_low_income_allowance.id = "eligible_low_income_allowance";  
eligible_low_income_allowance.outcomestyle = OutcomeStyleEnum.valueonly;  
eligible_low_income_allowance.outcomestyleSpecified = true;  
  
Attribute low_income_allowance_payment = new Attribute();  
low_income_allowance_payment.id = "low_income_allowance_payment";  
low_income_allowance_payment.outcomestyle = OutcomeStyleEnum.valueonly;  
low_income_allowance_payment.outcomestyleSpecified = true;  
  
Attribute eligible_teenage_allowance = new Attribute();
```

```
eligible_teenage_allowance.id = "eligible_teenage_allowance";
eligible_teenage_allowance.outcomestyle = OutcomeStyleEnum.valueonly;
eligible_teenage_allowance.outcomestyleSpecified = true;
```

4. Set attributes of the entities

When we create attributes for the generic client we create "Attribute" objects and set the id for the Attribute to the public name of the rulebase attribute. We also have to specify for each attribute, what value type it will use (ItemChoiceType)

```
Attribute claimant_income = new Attribute();
claimant_income.id = "claimant_income";
claimant_income.Item = new Decimal(13000);
claimant_income.ItemElementName = ItemChoiceType.numberval;
```

```
Attribute claimant_public_housing_client = new Attribute();
claimant_public_housing_client.id = "claimant_public_housing_client";
claimant_public_housing_client.Item = true;
claimant_public_housing_client.ItemElementName = ItemChoiceType.booleanval;
```

```
g.attribute = new Attribute[] { eligible_low_income_allowance,
                               low_income_allowance_payment,
                               eligible_teenage_allowance,
                               claimant_income,
                               claimant_public_housing_client
                             };
```

```
Attribute child1_age = new Attribute();
child1_age.id = "child_age";
child1_age.Item = new Decimal(16);
child1_age.ItemElementName = ItemChoiceType.numberval;
child1.attribute = new Attribute[] { child1_age };
```

```
Attribute child2_age = new Attribute();
child2_age.id = "child_age";
child2_age.Item = new Decimal(8);
child2_age.ItemElementName = ItemChoiceType.numberval;
```

```
child2.attribute = new Attribute[] { child2_age };
```

5. Call the assess method

Calling the assess method in the generic service is almost identical to the specific method, although the

names are different.

```
odsAssessServiceGeneric_SimpleBenefits_typeClient service = new  
odsAssessServiceGeneric_SimpleBenefits_typeClient();
```

```
AssessResponse response = service.Assess(request);
```

6. Process the response

Once the response has been returned by the rulebase, we need to process the response to get the answers to the questions that we asked. This requires a little more code in the generic format because the generic XML does not distinguish between different types of entities, and it does not have specific names for the attributes we need to get.

However, by adding some simple methods to look for the attributes and entities that we need, we can simplify the code.

For details on the very simple method `GetAttribute`, see the full listing of the code in the Appendix below

```
Attribute eligibleLowIncomeAllowance = GetAttribute(response, "eligible_low_income_allowance");
```

```
Attribute lowIncomeAllowancePayment = GetAttribute(response, "low_income_allowance_payment");
```

```
Attribute eligibleTeenageAllowance = GetAttribute(response, "eligible_teenage_allowance");
```

```
Console.WriteLine("\n---- Results ----");
```

```
if (eligibleLowIncomeAllowance.Item is Boolean)
```

```
{
```

```
    Console.WriteLine("eligible_low_income_allowance = " +  
        eligibleLowIncomeAllowance.Item);
```

```
}
```

```
else if (eligibleLowIncomeAllowance.Item is UnknownValue)
```

```
{
```

```
    Console.WriteLine("eligible_low_income_allowance is unknown");
```

```
}
```

```
else if (eligibleLowIncomeAllowance.Item is UncertainValue)
```

```
{
```

```
    Console.WriteLine("eligible_low_income_allowance is uncertain");
```

```
}
```

```
if (lowIncomeAllowancePayment.Item is Decimal)
```

```
{
```

```
    Console.WriteLine("low_income_allowance_payment = " +  
        lowIncomeAllowancePayment.Item);
```

```
}
```

```
else if (lowIncomeAllowancePayment.Item is UnknownValue)
```

```
{
```

```

        Console.WriteLine("low_income_allowance_payment is unknown");
    }
    else if (lowIncomeAllowancePayment.Item is UncertainValue)
    {
        Console.WriteLine("low_income_allowance_payment is uncertain");
    }

    if (eligibleTeenageAllowance.Item is Boolean)
    {
        Console.WriteLine("eligible_teenage_allowance = " +
            eligibleTeenageAllowance.Item);
    }
    else if (eligibleTeenageAllowance.Item is UnknownValue)
    {
        Console.WriteLine("eligible_teenage_allowance is unknown");
    }
    else if (eligibleTeenageAllowance.Item is UncertainValue)
    {
        Console.WriteLine("eligible_teenage_allowance is uncertain");
    }
}

```

4. Test the program

When you run the generic version of this simple program, you should get the following output printed to standard out. From the response, the results are exactly the same as the specific program.

```

---- Starting new SimpleBenefitsAssess (Generic) ----
Creating new Assess request
Setting attribute outcomes for 'eligible_low_income_allowance', 'low_income_allowance_payment', and 'eligible_teenage_
allowance'
Setting claimant_income to 13000.00
Setting claimant_public_housing_client to true
Setting child_age on child1 to 16
Setting child_age on child2 to 8

---- Request Sent to Oracle Determinations Server ----

---- Response from Oracle Determinations Server ----

---- Results ----
eligible_low_income_allowance = True
low_income_allowance_payment = 70.0
eligible_teenage_allowance = True

```

Generic vs. Specific WSDL

Generic vs. Specific WSDL

As you can see for both examples you can follow the same steps to generate and use an JAX-WS Java client for a rulebase deployed on the Determinations Server. You can use either client to achieve the desired operation.

The major difference between the specific and the generic client is ease of use versus maintainability. The specific format is easier to use and much less prone to error, because attributes and relationships have specific names within the rulebase. You cannot accidentally misname an attribute or a relationship using the specific format. The disadvantage with the Specific format is that adding, removing or renaming attributes and relationships will require you to regenerate the Java client using the JAX-WS wsimport tool.

The interface generated for the generic client however, can be used for any rulebase, and never needs to be recompiled. Its disadvantage is that it is easier to make mistakes with the names of attributes and relationships and the code is somewhat more cumbersome.

Appendix 1 – Glossary of terms

Appendix 1 – Glossary of terms

.NET – A framework provided by Microsoft for building applications. For more information on .NET see <http://www.microsoft.com/net/>

C# (Sharp) – A programming language created by Microsoft. See: <http://msdn.microsoft.com/en-us/vcsharp/aa336809.aspx>

End Point – An address that can be used to communicate with a web service. For this tutorial the end point is the location of the SimpleBenefits rulebase when deployed to the Oracle Determinations Server.

Oracle Determinations Server – A web application which provides Oracle Policy Automation services as a web service.

Rulebase – A compiled rule project authored in Oracle Policy Modeling.

Service Reference – In Visual Studio, this is a reference to an external Service. In the context of this tutorial the service is a Web Service.

URL – Uniform Resource Locator. A global address for documents and services on the World Wide Web

Web Service – A service provided over the Web. Typically using XML and SOAP.

WSDL – Web Service Description Language. A standard way of describing Web Services that use XML and SOAP

Appendix 2 - SimpleBenefitsSpecificComandLine Class

Appendix 2 - SimpleBenefitsSpecificComandLine Class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using SimpleBenefitsDotNetApp.SimpleBenefitsSpecific;

namespace SimpleBenefitsDotNetApp
{
    /// <summary>
    /// Tutorials and Examples - Create and use a C# client for Oracle Determinations Server
    /// using a specific WSDL
    /// </summary>
    class SimpleBenefitsSpecificCommandLine
```

```

{
    /// <summary>
    /// Assess request processing for the SimpleBenefits rulebase using a specific WSDL
    /// </summary>
    static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("---- Starting new SimpleBenefitsAssess (Specific) ----");

            Console.WriteLine("Creating new Assess request");
            // Create the assess and the entities that you will need for the request needs
            AssessRequest request = new AssessRequest();

            request.globalinstance = new GlobalInstanceType();

            request.globalinstance.listchild = new childEntityListType();

            childInstanceType child1 = new childInstanceType();
            child1.id = "child1";

            childInstanceType child2 = new childInstanceType();
            child2.id = "child2";

            request.globalinstance.listchild.child
                = new childInstanceType[] { child1, child2 };

            Console.WriteLine("Setting attribute outcomes for "
                + "'eligible_low_income_allowance', "
                + "'low_income_allowance_payment', "
                + "and 'eligible_teenage_allowance'");

            // Specify the outcomes (answers) that we want the Determinations Server to answer
            GlobalInstanceType g = request.globalinstance;

            g.eligible_low_income_allowance = new booleanAttributeType();
            g.eligible_low_income_allowance.outcomestyle = OutcomeStyleEnum.valueonly;
            g.eligible_low_income_allowance.outcomestyleSpecified = true;

            g.low_income_allowance_payment = new numberAttributeType();
            g.low_income_allowance_payment.outcomestyle = OutcomeStyleEnum.valueonly;
            g.low_income_allowance_payment.outcomestyleSpecified = true;

            g.eligible_teenage_allowance = new booleanAttributeType();
            g.eligible_teenage_allowance.outcomestyle = OutcomeStyleEnum.valueonly;
            g.eligible_teenage_allowance.outcomestyleSpecified = true;
        }
    }
}

```

```

// Set attributes and relationships of the entities

Console.WriteLine("Setting claimant_income to 13000.00");
g.claimant_income = new numberAttributeType();
g.claimant_income.Item = new Decimal(13000.00);

Console.WriteLine("Setting claimant_public_housing_client to true");
g.claimant_public_housing_client = new booleanAttributeType();
g.claimant_public_housing_client.Item = true;

Console.WriteLine("Setting child_age on child1 to 16");
child1.child_age = new numberAttributeType();
child1.child_age.Item = new Decimal(16);

Console.WriteLine("Setting child_age on child2 to 8");
child2.child_age = new numberAttributeType();
child2.child_age.Item = new Decimal(8);

odsAssessServiceSpecific_SimpleBenefits_typeClient service = new odsAssessServiceSpecific_
    SimpleBenefits_typeClient();
// Call the assess method
AssessResponse response = service.Assess(request);
Console.WriteLine("\n---- Request Sent to Oracle Determinations Server ----");

// Process the reponse
Console.WriteLine("\n---- Response from Oracle Determinations Server ----");
// look for the outcomes
booleanAttributeType lowIncomeAllowance = response.globalinstance.eligible_low_income_
    allowance;

numberAttributeType lowIncomeAllowancePayment = response.globalinstance.low_income_
    allowance_payment;

booleanAttributeType teenageAllowance = response.globalinstance.eligible_teenage_allowance;

// print out the results
Console.WriteLine("\n---- Results ----");

if (lowIncomeAllowance.Item is Boolean)
{
    Console.WriteLine("low_income_allowance_payment = " + lowIncomeAllowance.Item);
}
else if (lowIncomeAllowance.Item is UnknownValue)
{
    Console.WriteLine("low_income_allowance_payment is unknown");
}

```



```

using System.Linq;
using System.Text;
using SimpleBenefitsDotNetApp.SimpleBenefitsGeneric;
using Attribute = SimpleBenefitsDotNetApp.SimpleBenefitsGeneric.AttributeType;

namespace SimpleBenefitsDotNetApp
{
    /// <summary>
    /// Tutorials and Examples - Create and use a C# client for Oracle Determinations Server
    /// using a generic WSDL
    /// </summary>
    class SimpleBenefitsGenericCommandLine
    {
        /// <summary>
        /// Retrieves a specific attribute from the global instance of the response
        /// </summary>
        /// <param name="response">the assess response</param>
        /// <param name="id">the id of the attribute to retrieve</param>
        /// <returns>the attribute</returns>
        private static Attribute GetAttribute(AssessResponse response, string id)
        {
            foreach (Attribute attribute in response.globalinstance.attribute)
            {
                if (attribute.id.Equals(id))
                {
                    return attribute;
                }
            }
            return null;
        }

        /// <summary>
        /// Assess request processing for the SimpleBenefits rulebase using a generic WSDL
        /// </summary>
        /// <param name="args">arguments</param>
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("---- Starting new SimpleBenefitsAssess (Generic) ----");

                // Create the assess and the entities that you will need for the request needs
                Console.WriteLine("Creating new Assess request");

                AssessRequest request = new AssessRequest();
                request.globalinstance = new GlobalInstanceType();
            }
            catch { }
        }
    }
}

```

```

EntityType childEntity = new EntityType();
childEntity.id = "child";

request.globalinstance.entity = new EntityType[] { childEntity };

EntityTypeInstanceType child1 = new EntityTypeInstanceType();
child1.id = "child1";

EntityTypeInstanceType child2 = new EntityTypeInstanceType();
child2.id = "child2";

childEntity.instance = new EntityTypeInstanceType[] { child1, child2 };

// Specify the outcomes(answers) that we want the Oracle Determinations Server to answer
GlobalInstanceType g = request.globalinstance;

Console.WriteLine("Setting attribute outcomes for "
    + "'eligible_low_income_allowance', "
    + "'low_income_allowance_payment', "
    + "and 'eligible_teenage_allowance'");

Attribute eligible_low_income_allowance = new Attribute();
eligible_low_income_allowance.id = "eligible_low_income_allowance";
eligible_low_income_allowance.outcomestyle = OutcomeStyleEnum.valueonly;
eligible_low_income_allowance.outcomestyleSpecified = true;

Attribute low_income_allowance_payment = new Attribute();
low_income_allowance_payment.id = "low_income_allowance_payment";
low_income_allowance_payment.outcomestyle = OutcomeStyleEnum.valueonly;
low_income_allowance_payment.outcomestyleSpecified = true;

Attribute eligible_teenage_allowance = new Attribute();
eligible_teenage_allowance.id = "eligible_teenage_allowance";
eligible_teenage_allowance.outcomestyle = OutcomeStyleEnum.valueonly;
eligible_teenage_allowance.outcomestyleSpecified = true;

// Set attributes of the entities
Console.WriteLine("Setting claimant_income to 13000.00");
Attribute claimant_income = new Attribute();
claimant_income.id = "claimant_income";
claimant_income.Item = new Decimal(13000);
claimant_income.ItemElementName = ItemChoiceType.numberval;

Console.WriteLine("Setting claimant_public_housing_client to true");
Attribute claimant_public_housing_client = new Attribute();

```

```

claimant_public_housing_client.id = "claimant_public_housing_client";
claimant_public_housing_client.Item = true;
claimant_public_housing_client.ItemElementName = ItemChoiceType.booleanval;

g.attribute = new Attribute[] { eligible_low_income_allowance,
                                low_income_allowance_payment,
                                eligible_teenage_allowance,
                                claimant_income,
                                claimant_public_housing_client
                                };

Console.WriteLine("Setting child_age on child1 to 16");
Attribute child1_age = new Attribute();
child1_age.id = "child_age";
child1_age.Item = new Decimal(16);
child1_age.ItemElementName = ItemChoiceType.numberval;

child1.attribute = new Attribute[] { child1_age };

Console.WriteLine("Setting child_age on child2 to 8");
Attribute child2_age = new Attribute();
child2_age.id = "child_age";
child2_age.Item = new Decimal(8);
child2_age.ItemElementName = ItemChoiceType.numberval;

child2.attribute = new Attribute[] { child2_age };

// Call the assess method
odsAssessServiceGeneric_SimpleBenefits_typeClient service = new odsAssessServiceGeneric_
    SimpleBenefits_typeClient();
AssessResponse response = service.Assess(request);
Console.WriteLine("\n---- Request Sent to Oracle Determinations Server ----");

// Process the reponse
Console.WriteLine("\n---- Response from Oracle Determinations Server ----");

Attribute eligibleLowIncomeAllowance = GetAttribute(response, "eligible_low_income_allowance");

Attribute lowIncomeAllowancePayment = GetAttribute(response, "low_income_allowance_
    payment");

Attribute eligibleTeenageAllowance = GetAttribute(response, "eligible_teenage_allowance");

Console.WriteLine("\n---- Results ----");

```

```

if (eligibleLowIncomeAllowance.Item is Boolean)
{
    Console.WriteLine("eligible_low_income_allowance = " + eligibleLowIncomeAllowance.Item);
}
else if (eligibleLowIncomeAllowance.Item is UnknownValue)
{
    Console.WriteLine("eligible_low_income_allowance is unknown");
}
else if (eligibleLowIncomeAllowance.Item is UncertainValue)
{
    Console.WriteLine("eligible_low_income_allowance is uncertain");
}

if (lowIncomeAllowancePayment.Item is Decimal)
{
    Console.WriteLine("low_income_allowance_payment = "
        + lowIncomeAllowancePayment.Item);
}
else if (lowIncomeAllowancePayment.Item is UnknownValue)
{
    Console.WriteLine("low_income_allowance_payment is unknown");
}
else if (lowIncomeAllowancePayment.Item is UncertainValue)
{
    Console.WriteLine("low_income_allowance_payment is uncertain");
}

if (eligibleTeenageAllowance.Item is Boolean)
{
    Console.WriteLine("eligible_teenage_allowance = " + eligibleTeenageAllowance.Item);
}
else if (eligibleTeenageAllowance.Item is UnknownValue)
{
    Console.WriteLine("eligible_teenage_allowance is unknown");
}
else if (eligibleTeenageAllowance.Item is UncertainValue)
{
    Console.WriteLine("eligible_teenage_allowance is uncertain");
}
}
catch (Exception e)
{
    Console.WriteLine("Error occurred " + e.Message);
}
}
}

```

}

Tutorial: Embed Web Determinations within my portal - a simple implementation

Oracle Policy Automation now provides an Interview Portlet that allows you to embed a Web Determinations interview in a portal. For more information, see [Customize the Interview Portlet](#).

If you do not wish to use the out-of-the-box portlet, you can use this tutorial to teach you how to produce a very simple Interview Portlet, designed to provide the end user with a basic set of screens that will allow the completion of a simple Goal Investigation.

Some more complex features such as access control, inter-portlet communication and more advanced Oracle Policy Automation functionality may be touched on, but for the purposes of this tutorial they will not be gone into in any detail. It is assumed that readers of this tutorial are already familiar with their chosen portal and the intricacies of its Portlet API.

In summary, this tutorial should produce a portlet that utilizes the Interview Service to provide the user with the following:

- A list of available rulebases.
- A list of available goals and their state, for a chosen rulebase.
- A set of screens that allow for the collection of simple text and boolean attributes.

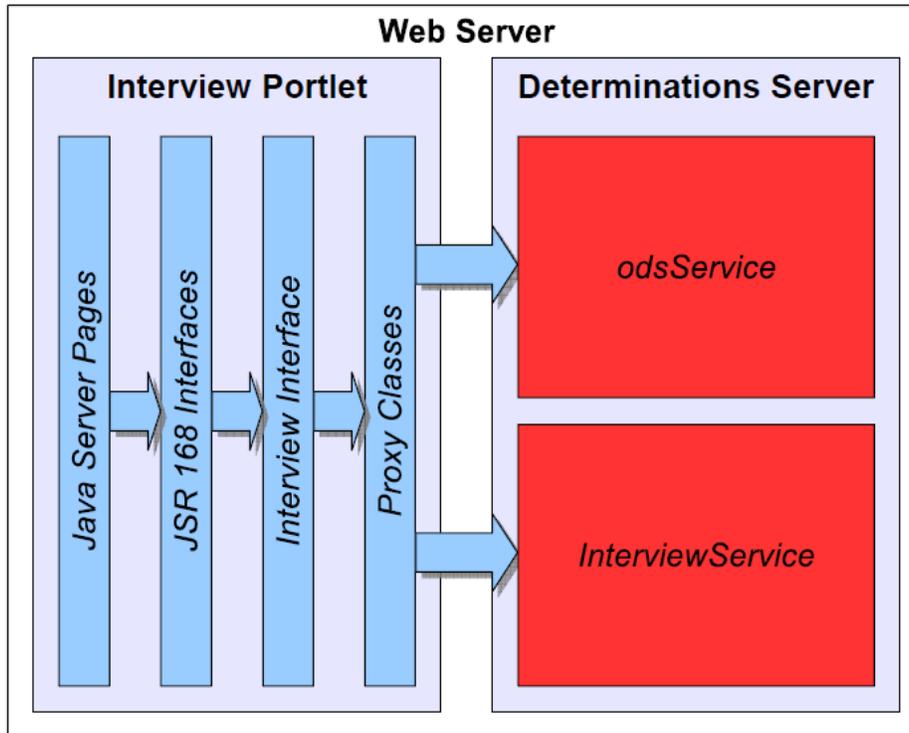
To complete this tutorial, you will need the following:

- A Portal Deployment - one for which you have Administrator access so that you are able to add new portlets (for the purposes of this tutorial we will be using Portal 11g).
- A Web Server - one to which you have Administrator access so that you are able to deploy the portlet to it (for the purposes of this tutorial we will be using Weblogic 11g).
- An IDE - not essential, but highly recommended (for the purposes of this tutorial we will be using JDeveloper 11g).

Throughout this tutorial the aim is to keep things as simple as possible, providing a basic overview of the way the parts of the portlet being created will hang together without going into excessive detail; for example, in the [Render the Summary screen](#) section below, we look at a simple Web Service call using proxy classes to retrieve a new Session Token. Unfortunately no web service call is ever guaranteed to succeed and there are numerous errors that may occur when attempting to call the **openSession()** method.

Between networking errors, web server issues and problems with the requests themselves being malformed or invalid, there are numerous problems that a portlet would be expected to contend with. For brevity, this tutorial will not perform any error handling, but note that for every web service call made there should always be some attempt to at least check that the call was successful (**isSuccessful()**), and potentially parse the **errorList**. No return is ever guaranteed.

Implementation Plan



As shown in the diagram above, there are four main sections (those shown in blue) that this tutorial will focus on implementing. Taking a ground up approach, starting with the proxy classes required to interact with the Oracle Determinations Server, these are as follows:

- *Proxy Classes* - auto-generated based on the WSDL interface provided by the Determinations server, these classes and the corresponding Java libraries allow the building and sending of requests to the Determinations Server and process the corresponding response.
- *Interview Interface* - this provides a layer in-between the Action/Render logic of the portlet and the proxy classes themselves.
- *JSR 168 Interfaces* - this is the portlet implementation itself, including all Action and Render methods; for example, **doView**, **processAction** and so on.
- *Java Server Pages* - though it is possible to construct everything required for the Render response by using pure Java code, it is not a particularly clean or manageable option. For the purposes of this tutorial we have chosen to use JSP, but there are numerous other options you may prefer, including template engines such as Velocity which is used in Web Determinations.

Deploy Determinations Server

Apart from a working development environment and a portal instance, one of the most important things to set-up before commencing work is a deployment of the Determinations Server containing your target rulebase.

The Installation Guides that ship with the Runtime Installers do a better job of explaining the process of deploying the Determinations Server to your target Web Server than could ever be accomplished in this tutorial, so it is recommended you refer to them for the relevant information. Remember to make a note of the Determinations Server's URL once it has been deployed, as you will need this in the next few steps to pull out the required WSDL.

Produce a set of Proxy Classes

In order to facilitate communication with the Interview Service from Java, we are going to be the JAX. To do this, firstly a set of

Proxy Classes that represent our web service in Java must be produced, allowing requests to be built and responses processed using Java Objects.

Proxy class generation is something that is almost always done automatically, and a number of different tools exist to produce the required classes from a source WSDL. Depending on the library that you decided to use, the process you go through to generate the proxy classes will be slightly different, but the general theory is the same. For this example JDeveloper and its *Create Web Service Proxy* wizard will be used to create the Proxy Classes.

Generate the Proxy Classes

The first step is to pick out the URL that will be used to access the Interview Service for the deployed rulebase. The Determinations Server itself has a base URL which, when provided with no arguments, will provide a list of all available WSDLs, similar to the one shown below. As you can see, the Determinations Server itself has three WSDLs available, and each deployed rulebase has eight.

Available Services

Server

- </determinations-server/server/soap?wsdl>
- </determinations-server/server/soap/10.2?wsdl>
- </determinations-server/server/soap/10.0?wsdl>

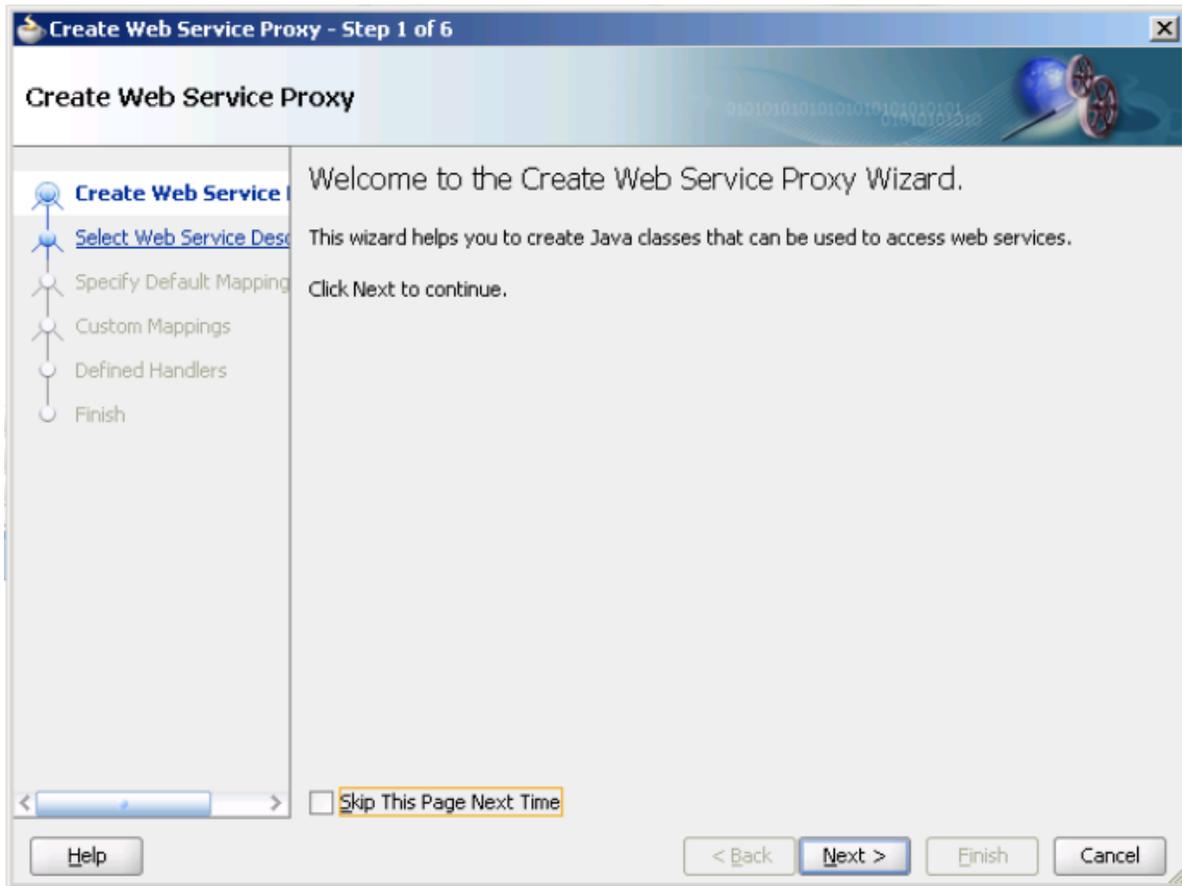
InterviewServiceTest

- </determinations-server/interview/soap/InterviewServiceTest?wsdl>
- </determinations-server/interview/soap/10.2/InterviewServiceTest?wsdl>
- </determinations-server/assess/soap/specific/InterviewServiceTest?wsdl>
- </determinations-server/assess/soap/generic/InterviewServiceTest?wsdl>
- </determinations-server/assess/soap/specific/10.2/InterviewServiceTest?wsdl>
- </determinations-server/assess/soap/generic/10.2/InterviewServiceTest?wsdl>
- </determinations-server/assess/soap/specific/10.0/InterviewServiceTest?wsdl>
- </determinations-server/assess/soap/generic/10.0/InterviewServiceTest?wsdl>

From the list of available WSDLs, the WSDL from which the Proxy Classes will be generated, and the one that will be used for the end point, is the standard Interview WSDL. Note that while the specific and generic WSDLs have legacy 10.0 endpoints available, supporting a non-containment based structure, the Interview Service does not; this is due to the Interview Service only having been introduced in the 10.2 release.

Having selected the relevant WSDL (in this case: `/determinations-server/interview/soap/ InterviewServiceTest?wsdl`), continue with the process of generating the proxy classes through JDeveloper.

The *Create Web Service Proxy* wizard can be accessed by right clicking on a *Project* in JDeveloper, selecting **New...** and navigating to the *Web Services* category under *Business Tier*. You may need to go to the *All Technologies* tab depending on the type of project you are presently developing, but after selecting *Web Service Proxy* you should be presented with the following wizard:



Like other proxy class generation tools, the wizard expects to be provided with the location of a valid WSDL for the Web Service you are looking to communicate with. In subsequent steps you will need to provide the Interview Service URL discovered above.

JDeveloper will also request that you provide a root package name in which the classes should be generated. Take care at this point to ensure the package can be referenced easily but does not clutter up any existing source trees.

Further steps may be required depending on the nature of your development, with security policies and handler classes being defined in the last two steps of the wizard. For the purposes of this tutorial however, we can go ahead and click **Finish** once we have specified a package name, leaving JDeveloper to generate the proxy classes and add them to the project ready for consumption.

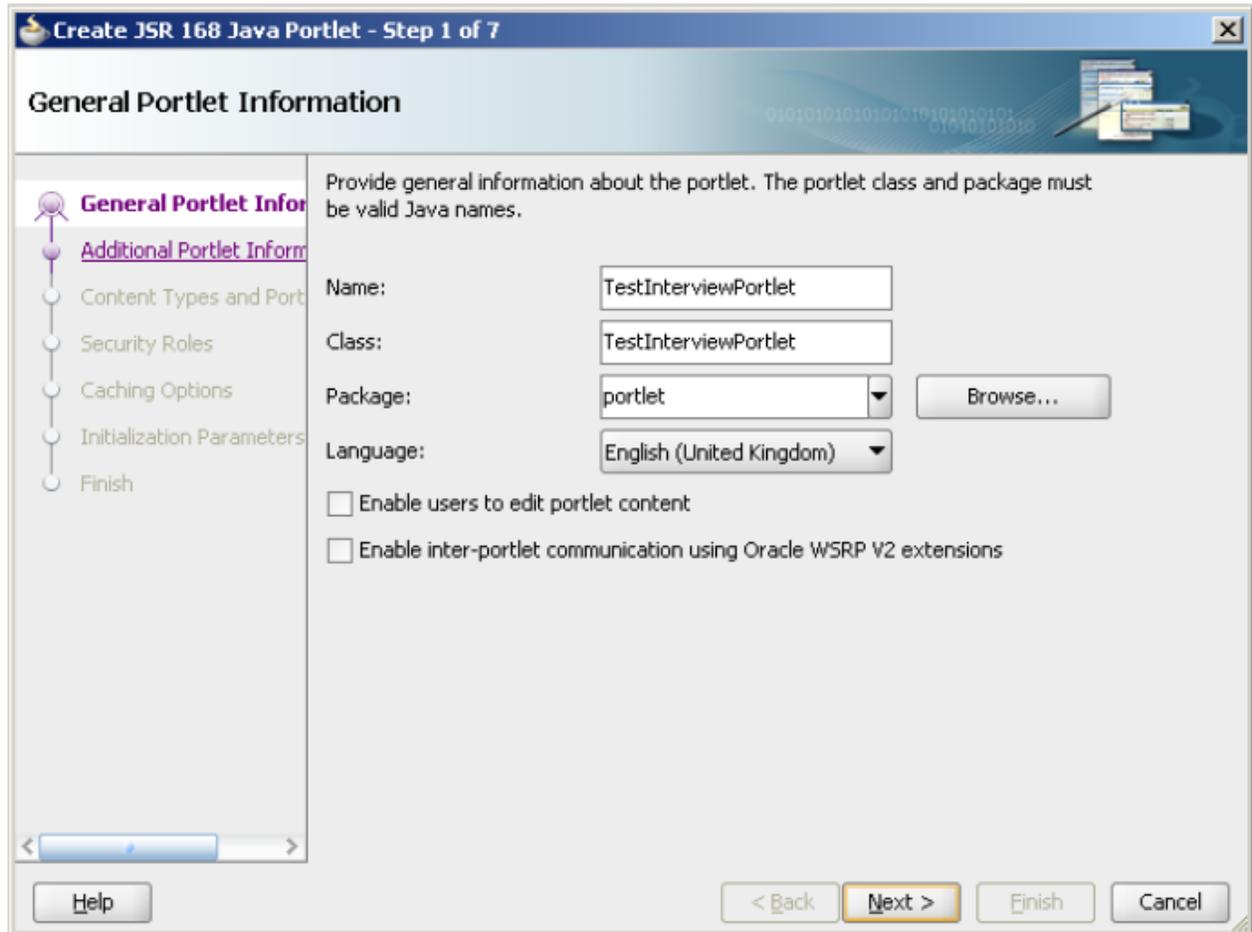
Get the Portlet up and running

In order to get a simple JSR 168 Portlet up and running, the only step we really need to take is extending the **GenericPortlet** abstract class and overriding the **Action** and **Render** methods. Generally there is no need to override the **Render** or

doDispatch methods, since ultimately these methods handle the setting of the portlet title and the dispatching of the request to one of the available **doView**, **doEdit** or **doHelp** methods. Overriding the specific methods is faster and neater.

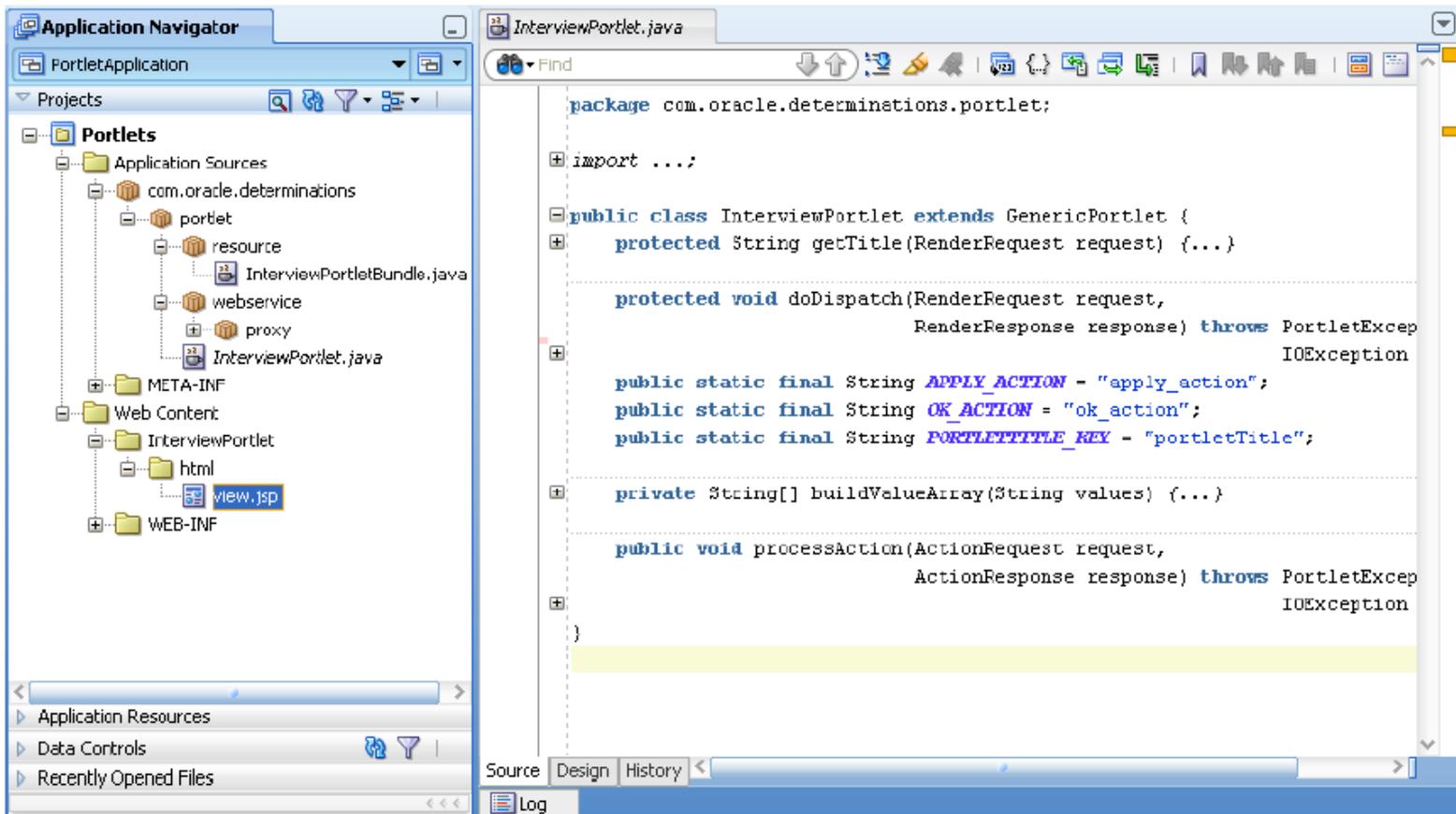
JDeveloper is being used for this tutorial and so we will be running through the Standards-based *Create JSR 168 Java Portlet* wizard in order to get up and running.

With this in mind, the first step is right click on the project and select **New...**, navigating to the *Portlets* category under *Web Tier* in the subsequent dialog box. Here you should see the option to kick off the Standards-based *Create JSR 168 Java Portlet* wizard, the first step of which is as follows:



Since we're making a simple View only portlet, all we need is an appropriate *Name*, *Class* and *Package* in which it will be created. Unchecking the *Enable users to edit portlet content* flag will ensure that there are no additional *Edit* pages and code snippets created. Clicking **Next** will auto-fill the *Additional Portlet Information* page as shown below, filling in details such as the *Portlet Title* and *Display Name* and allowing us to go ahead and finish the tutorial.

Whether you choose to use the *Create JSR 168 Java Portlet* wizard, or another environment's wizard, or hand craft the required code, the end result should be a very basic JSR 168 Portlet that can be deployed and added to a portal server. In the case of the JDeveloper wizard, a JSP page is also created to which the *View* requests are dispatched. Using JSP or even a more advanced template engine is preferable to attempting to hand craft the portlet's return each time.



Open a Session

The first step in any communication with the Interview Service is the opening of a Session. Without the token provided by **OpenSession** it won't be possible to make any successful calls to other methods, so the first task upon having a user load a Portlet is to assign them a session.

To do this, create a simple method in the Interview Interface which takes no arguments and returns a String, the Session Token.

openSession Example Code

```
public static String openSession() {
    OpenSessionResponse response = odsInterviewService.openSession(
        new OpenSessionRequest());
    return response.getInterviewSessionId();
}
```

As you can see in the simple method above, accessing the Web Service now the proxy classes have been created, is simply a matter of calling the appropriate method and then sorting through the Java Object returned. In this case, a simple **OpenSession** method call is all that's required and the **InterviewSessionId** can be extracted from the response immediately.

Now this token has been obtained, it needs a place to be stored so that it can be used by each subsequent request made by the user. By storing the returned token in the **PortletSession** we have a simple way of tracking sessions on a per-user basis. A simple way to do this is to check for the presence of the token in the **PortletSession** each time a **Render** or **Action** request is received, returning it or making a call to the newly created **openSession** method if it is not present.

Retrieve a Summary screen

The *Summary* screen acts as a landing page for all rulebase interactions. Its main purpose is generally to hold a list of *Goals* which can be investigated and to report on the state of those goals. However, by using the *Screens Editor* inside Oracle Policy Automation it can be extended to contain additional HTML, conditional links to Documents, and numerous other elements. Due to the potential complexities involved in rendering summary screens, a simple *Goal List* could be considered an alternative. This tutorial though will instead focus on providing the capability to render basic summary screens, instead of forming its own.

What is needed first then are a number of additional methods in the Interview Interface, capable of supporting the retrieval of the *Summary* screen. What we are looking to do is send a **GetScreen** request to the Interview Service, but to do this we first need to locate the *Identifier* of the summary screen. The **ListScreens** method returns a list of all the available screens in a rulebase, similar to the following response:

ListScreens Response

```
<typ:list-screens-response>
  <typ:interview-session-id>454c7bab-a293-44ec-a948-
25282491405b</typ:interview-session-id>
  <typ:entity id="global">
    <typ:instance id="global">
      <typ:screen id="qs$summary$global$global" name="summary"
title="Assessment Summary" context-entity-id="global" context-instance-
id="global" type="summary" is-automatic="false"/>
      <typ:screen id="qs$s4@Screens_xint$global$global" name="s4@Screens_xint"
title="Person Collection" context-entity-id="global" context-instance-
id="global" type="question" is-automatic="false"/>
      <typ:screen id="dr$DSO$global$global" name="Default Data Review"
title="Data Review" context-entity-id="global" context-instance-id="global"
type="data-review" is-automatic="false"/>
    </typ:instance>
  </typ:entity>
</typ:list-screens-response>
```

First it is necessary to have a method that will search the above response for a screen where the type attribute equals summary and pull out the id attribute to obtain the *Identifier* required for the **GetScreen** request. Following this, a method that accepts an *Identifier* and returns the screen contained in the response, is all that is required before the rendering phase.

Method Signature	Description
<code>String getSummaryScreenId(sessionToken)</code>	Makes a ListScreens request and searches the response for the Id of the Summary screen
<code>String getScreen(sessionToken, screenId)</code>	Makes a getScreen request for the given screenId – It's important to note that any screens returned will be populated with the appropriate data from the given session

Since we are dispatching to JSP in order to produce the rendered mark-up, it is necessary to make two additions. The first is to attach the **Screen** object returned, in this case, by the **getScreen** method to the **RenderRequest**; this gives the page access to the **Screen** object and allows it to be parsed in the JSP code that is written.

Render the Summary screen

So far we have:

- Had a user land the portlet and opened a new session for them.
- Assigned their Session Token to their **PortletSession**.
- Established that as there is no interview in progress, the *Summary* screen should be displayed.
- Searched the list of screens in the rulebase and established the id of the *Summary* screen.
- Requested and had returned, a definition of that *Summary* screen.
- Attached the returned screen definition to the **RenderRequest**.

Up to this point, the focus has mainly been on the in's and out's of working with the Interview Service and associated proxy classes, pausing only briefly to set-up a basic JSR 168 Portlet. In the last section though, the Summary Screen for the rulebase was retrieved, so it now needs to be rendered to the user. The last stage therefore, is parsing this Screen Definition in JSP to produce the desired HTML, XHTML, WML, and so on.

Interview Screens themselves are potentially very complex structures, containing not only 15 different Control types but errors, warnings, property and potentially commentary as well. For the simple interview portlet though, we are only interested in a small subset of those controls, with discussion of the remaining controls in the [Advanced Implementation Considerations](#) section.

All controls inherit from a base control type known as the **ControlBase**, and as such have a set of consistent attributes; of interest to our *Simple Portlet* are:

- *is-visible* – this needs to be checked on every control to decide whether or not it is rendered to the user. Where this is false, the control shouldn't be included in the screen.
- *caption* – particularly important for items such as goal-controls, which link off to other pages; this is the caption that should be displayed with the control.
- *is-html* – defines whether or not the contents of the control should be rendered as HTML. Generally only used on label-controls, it is nevertheless still important to check this so a decision can be made as to whether the contents should be escaped before rendering.
- *css-class* – the *class* that should be assigned to the element containing this control, for styling by CSS.

Given the above list of Attributes, the following is a list of controls that our Simple Portlet will be capable of rendering:

- *label-control* – a simple extension of the *Base Control* type, designed for displaying text, and potentially HTML to the user.
- *goal-control* – restricted to the *Summary* screen, goal-controls extend the base control to include not only the identifier of one of the goals available for investigation in the rulebase, but also potentially their value if they have been investigated (this can potentially be unknown or uncertain).
- *boolean-control*, *text-control*, *number-control* – a subset of the seven different attribute controls that have been chosen for our simple portlet. These allow the user not only to input the values of the associated attributes, providing the attribute Id and question text, but also potentially provide us with a value to pre-populate into the input control (potentially either a default or previously entered value).

We now have a list of elements to look for in the screen definition, and since that has been attached to the **RenderRequest** it is also possible to access that definition in the JSP page. The *Summary* screen in the example that was retrieved in the last section contains both *label* and *goal* controls, so it is important that the dispatched JSP page is able to handle both.

JSP Skeleton

```
<%@ page contentType="text/html" import="javax.portlet.*,java.util.*,
    com.oracle.determinations.portlet.WebService.*"%>

<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>

<portlet:defineObjects/>

<%
InterviewScreen screen = renderRequest.getAttribute("screen-definition");
%>
```

The *JSP Skeleton* shown above is simply the start of a JSP page, including the appropriate *import* and *taglib* statements to give access to the portlet tags. One of these portlet tags is `<portlet:defineObjects/>`, which establishes three objects for use in portlet JSP pages; **renderRequest**, **renderResponse** and **portletConfig**. You can see use of one of these objects, the **renderRequest**, at the end of the JSP stub above, where we pull the screen definition out of the **renderRequest** attributes.

With that screen definition now in scope and available to be used in the JSP page, all that is left to do is to pull out any required screen attributes:

JSP Screen Attributes

```
<div id="header">Title: <%= screen.getTitle() %></div>

<div id="entity_header">Entity: <%= screen.getContextEntityId() %> - <%=
    screen.getContextInstanceId() %>

...

<div id="footer">Automatic Screen: <%= screen.getIsAutomatic() %></div>
```

and loop through the list of controls attached to the screen:

JSP Screen Controls

```
<% for (ControlBase control : screen.getTextControlOr...Control()) {  
    // this would be better served by overridden method calls  
    if (control instanceof LabelControl) { %>  
        <div class="control"><%= control.getCaption() %></div>  
  
<% } else if (control instanceof GoalControl) { %>  
        <div class="control">  
            <a href="<portlet:renderURL> <portlet:param name="_goalId"  
value="<%= ((GoalControl)control).getGoalId() %>"> </portlet:param>  
</portlet:renderURL>"><%= control.getCaption() %></a>  
  
        </div>  
  
<% }  
} %>
```

By bringing scriptlets into the mix, it is possible to effectively loop through the *Control* list, checking the type of each control before applying a suitable rendering method. In the above code stub you can also see that we've used the `portlet:renderURL` tag to create a URL with which to start our interview. At present, clicking that link will merely result in the same screen being displayed again, since our **doView** or **doDispatch** methods have no logic to deal with the presence of a goal parameter. In the next section we'll look at what we need to add in order to go through an interview in our portlet.

Process basic Question screens

In the previous section Portlet JSP tags were used to construct a new **renderURL** with a *goal* parameter. What this means is that now, when a user clicks on that link, a new **doView** or **doDispatch** (depending on code structure) request will be made, with the goal parameter passed to it in the **RenderRequest**.

By checking for the presence of this goal parameter, it is possible to make a decision as to whether or not the current **getScreen** request to retrieve the summary screen is enough, or whether it is now necessary to shift into "Interview" mode.

When first called, the *Investigate* method takes a session token and goal identifier and returns the first screen definition in that goal's investigation. Subsequent calls to the investigate method then, contain the session token, goal identifier and a populated version of the last screen returned. The interview service then takes the data from this populated screen, adding to the rulebase session and returning the next required screen. Once the goal is calculated as known, unknown or uncertain, the summary screen is then returned with the goal value updated.

RenderRequest Parameter Stub

```
public void doView(RenderRequest request, RenderResponse response)
    throws PortletException, IOException {

    // Initial doView code (setContentType, etc.)
    // Check for, and if necessary open, an Interview Session

    String goalId = request.getParameter("goal");

    if (goalId == null) {
        // Retrieve and render the summary screen
    } else {
        // Pass the goalId to the Investigate method and render the returned
screen
    }
}
```

The above stub shows what would be required to differentiate between requests where a goal was being investigated, and those where just the *Summary* screen needed to be displayed. By modifying the JSP page we looked at previously, we can also add code to handle rendering a form for data collection. In fact it would be possible in simple cases to use a single JSP page to handle rendering of both the *Summary* and *Interview* screens, though this is not generally recommended.

When creating the data collection form though, the question is raised as to where to target the form's action. Where previously we have used JSP to create a **renderURL**, passing it the **goalId** as a parameter, and it would be possible to continue this theme in the form's target. The alternative though is to target the form using JSP to create a new **actionURL**.

A strong argument for this is that by targeting the form at an **actionURL** we're able to move the form processing code into the **processAction** method of the portlet, something which is more in line with the intentions of the portlet specification. This would also allow this code to be reused where the portlet container handled other action requests.

Since there is no easy way to pass the screen definition returned by the investigate method from the **processAction** to the appropriate render method (**doView**, **doDispatch**, **render**) through the Request and Response objects, we instead go back to the Portlet Session, storing the **InterviewScreen** there alongside our **sessionId** that we placed previously.

Considering the above issue and with performance in mind, we choose to place our form processing code in the **processAction** method of our example, targeting the rendered forms using the same actionURL.

JSP Interview Form

```
<form method="POST" action="<portlet:actionURL/>">

<input type="hidden" name="_goalId" value="<%=
    renderRequest.getAttribute("goal") %>"/>

<input type="hidden" name="_screenName" value="<%= screen.getId() %>"/>

<% for (ControlBase control : screen.getTextControlOr...Control()) {

    // ... render each control in turn

} %>
<input type="submit"/>

</form>
```

Store the screen in the portlet session

With the investigate method expecting a populated screen definition to be returned in order to continue the interview, we are presented with the following choices:

- Attempt to recreate the screen from scratch, building it up from the base **InterviewScreen** class.
- Query the Interview Service and investigate method again, using the screen definition returned.
- Store the screen-definition in memory, generally using the **PortletSession**.

For our example, we are going to use the final option, choosing to store the screen in the **PortletSession** before the code dispatches to the JSP page, retrieving it from the session when the subsequent form is then submitted. While we need to ensure that the submitted form is actually related to the screen in memory, we'll be using a request to the **getScreen** method of the interview service to handle retrieving of a screen which is not currently stored in memory.

Method Signature	Description
<code>InterviewScreen investigate(goalId, sessionId, InterviewScreen)</code>	Passes the populated InterviewScreen to the Investigate web service method and returns the screen contained in the response
<code>InterviewScreen investigate(goalId, sessionId)</code>	Makes an Investigate request to the Interview Service for the given goal and session token, returning the screen contained in the response

When the form is submitted to an Action URL the **processAction** method is called with a list of parameters from the form itself. These parameters contain not only hidden fields we've inserting detailing the name of the screen and particular **goalId** being investigate but also the value of all fields on the form.

Since our form fields are named to correspond with particular attributes on the screen it represents we can confidently use these parameters in the Request object to update to the Screen definition with new **CurrentValues**. Looping through each control on the screen in turn, we attempt to pull out the value from the parameter list, format it appropriately where required (Dates, Numbers, and so on) and insert it back into the Screen definition.

The method signature above describes the new method in our Interview Interface, designed to handle part of the form processing requirements. When this screen definition has been updated, the method signature shown above that accepts an **InterviewScreen** parameter can be used to update the session on the Interview Service.

While the example does not handle entity collection screens where multiple instances may be collecting the same attribute, extending the Attribute Id to contain the Instance Id it relates to would be possible.

With this in mind then, the next thing to do is extend our example's **processAction** method to handle the possibility of a form submission.

Extend the portlet to support multiple Entity instances

Very few rulebases will ever contain a global entity only. Indeed one of the key advantages of using the Determinations Engine is the ability to define, populate and reason with multiple instances of different entities.

To support multiple entity instances then, our portlet must be extended. Whilst the attribute collection is handled automatically, since the Interview Engine itself takes care of producing multiple "instances" of attribute collection screens for entities, the entity instantiation or collection is not. The next thing to do then is extend our portlet to handle *Entity Collect* screens.

In order to do this it is necessary to first extend the core JSP page to be able to render containment relationship elements. The elements appear on a screen designed to collect entities and contain zero or more entity-instance elements. These entity-instance elements may themselves contain controls which then need to be rendered (allowing the user to collect attribute data at the same time as entering entity instances), but for the purpose of our basic tutorial we will merely render a single label-control for each entity instance.

With the entity collect screen rendered then, the user must be presented with links to add another instance of the entity, or finish collecting instances and continue to the next part of the interview.

These two links, in our basic portlet, will target the **actionURL**, with parameters inserted into the URL containing instructions for the **processAction** code to either add another entity instance to the screen or continue with the interview.

Finally then, our **processAction** code must be updated to handle the creation and production of an entity collect screen. When an entity collect screen is first returned by the Interview Service it is inserted into the Portlet User Session. Further requests by the user to add an additional instance result in this in session screen definition being updated until, when the user clicks the continue link this screen definition is passed to the **Investigate** method.

In this way, a basic entity collect screen can be produced.

Advanced Implementation Considerations

While this tutorial is meant to give an overview of portlet construction there are some more complicated topics that were not covered as part of the basic Portlet we produced.

Load and Save

One of the advantages of using the Interview Service is the plug-in architecture that is shared with Web Determinations. Data adaptors written for the Interview Engine can be deployed to both the Determinations Server and Web Determinations, giving the Interview Service the ability to load and save common sessions.

For a Portlet this means that we don't have to worry about serializing and saving sessions and can instead make simple **LoadSession**, **SaveSession** and **ListSession** requests to the Interview Service itself, in order to facilitate session storage. What is import-

ant though is that some way of linking a session's identifier to a particular User or Group within the portal be produced, so it may be later loaded from the session store.

Interview API

Though the portlet given in this tutorial is designed around the Interview Service it is entirely possible that the API will be chosen instead. Though the Interview Interface concept used in the tutorial means that the changeover of calls to the Interview API from the Service need only be done in one location, it is unfortunately not the only change required. The JSP and Portlet all reference the proxy classes in order to read and write the request and response objects from the Interview Service and these references will all need to be changed to utilize the API Classes. Consider the importance then, of deciding your chosen architecture up front to avoid the need for a switch later in the development cycle.

Where a change may be needed or the possibility of needing to use both the Interview Service and API is present, consider extending the functionality of the Interview Interface to completely wrap all the required classes, instead of directly referencing either the API or Proxy classes. In this way only the Interview Interface need be swapped, though the development effort required is increased.

Data Review

One of the features of Web Determinations not covered by the tutorial is the construction of a Data Review page. This page provides a simple overview of all data that the user has entered into the session, with the possibility of providing links so the user may be directed to the screen in which they entered that data, allowing them the ability to change their answers without completely resetting the session.

The Interview Service provides three methods which would be useful in the construction of a Data Review Screen:

- **GetUserSetData** – when a call to this method is made the response contains all the data the user has entered into the session, formatted in an entity contained format. By making a call to this method and rendering the response you provide the user with an overview of their answers thus far.
- **ListScreens** – though the basic usage of this method provides just a list of all screens contained in the rulebase, the request can also contain the id of an attribute (or Entity), with the response then containing only the screen in which that attribute (or the Entity Collect Screen for a given Entity) is collected.
- **GetScreen** – by making a ListScreens request as described above, the resulting Screen ID can then be passed to the **GetScreen** request and the results rendered with the **CurrentValue** elements inserted into the rendered fields. This simulates returning the user to the screen in which they first entered the data.

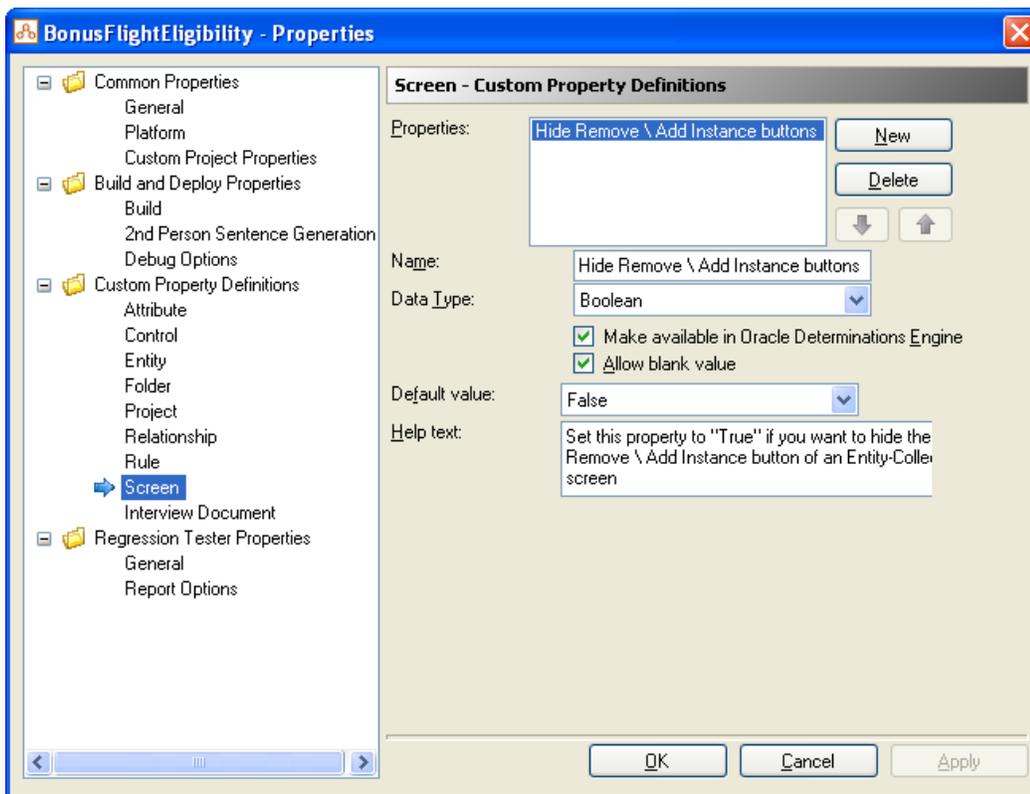
Example: Add and remove buttons on Entity Collect screens

This example demonstrates how to hide the **Add New Instance** and **Remove Instance(s)** buttons on an *Entity Collect* screen through the use of custom properties and the modification of velocity templates. Oracle Policy Modeling provides a facility to attach custom property key-value pairs to individual screens and controls during rulebase authoring. These are then exposed to the velocity templates that use them to render the Web Determinations user interface.

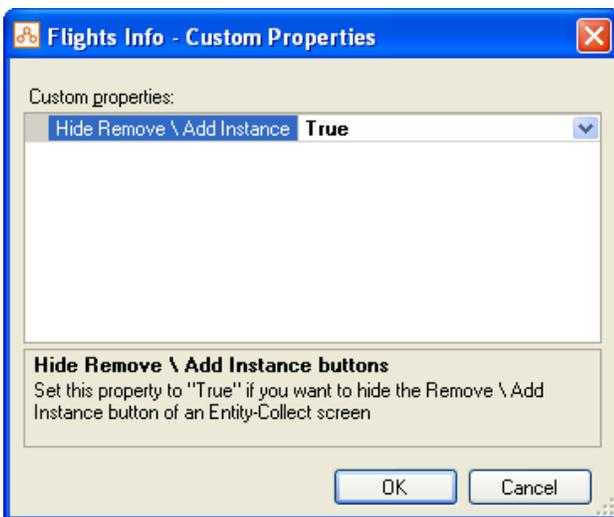
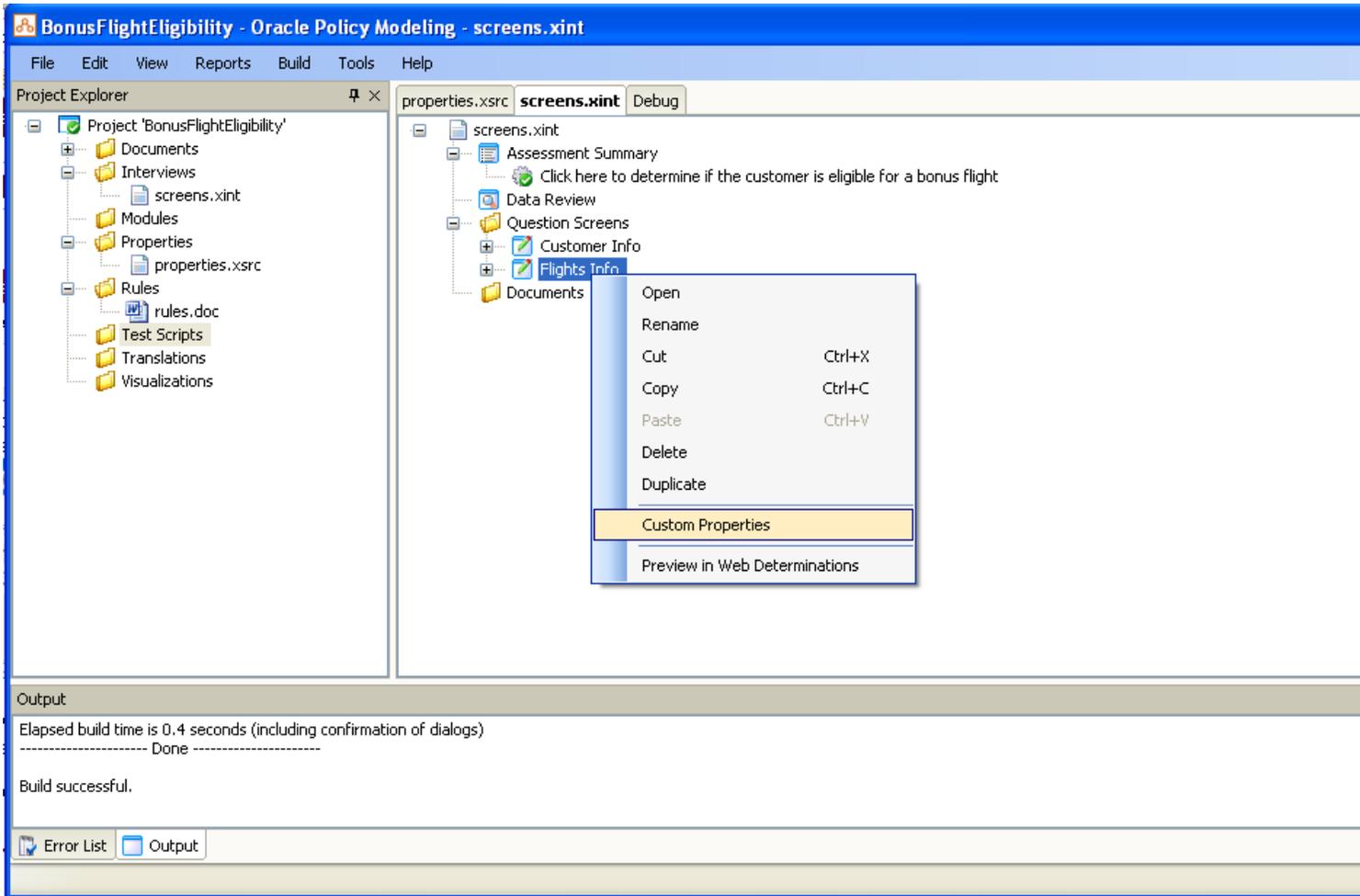
Examine the rulebase example

First, we will examine the *BonusFlightEligibility* rulebase which we will use in this example. This is a simple rulebase that will infer an airline customer's eligibility for a bonus flight based on his/her membership and flights flown details.

1. In Oracle Policy Modeling, open the *BonusFlightEligibility* rulebase project located in `<JAVA_RUNTIME_ROOT>\examples\rulebases\source\BonusFlightEligibility`.
2. Verify the rulebase's boolean Screen custom property; navigate to **File -> Project Properties -> Custom Property Definitions -> Screen**.



3. As seen in the screenshot above, the *Hide Remove\Add Instance buttons* custom property has a default value of "False". In this example, we will hide the **Add New Instance** and **Remove Instance(s)** buttons on the *Flights Info* entity collect screen. To do this, the value of the custom property *Hide Remove\Add Instance buttons* for the screen should be "True". This will tell the templates not to display the entity collect buttons on the *Flights Info* screen. To verify this, open the screens file, right-click on the *Flight Info* screen and select **Custom Properties** from the menu.



4. That's all the relevant configuration that we need to verify for this rulebase. To test, in Oracle Policy Modeling choose **Build and Debug with Screens**.

5. An interview should open, in which we can determine if "the customer is eligible for a bonus flight". We should be able to verify at this point that we can navigate to all screens. Most input controls cannot be populated since they are read-only.

Screen: [Assessment Summary \(summary\)](#)

ORACLE Web Determinations

[Data Review](#) [Save](#) | [Save As](#) | [Load](#) | [Restart](#) | [Close](#)

Rulebase: BonusFlightEligibility Locale: en-US User ID: guest

Assessment Summary

- Click here to determine if the customer is eligible for a bonus flight

Investigation: [Show in decision view](#)

Screen: [Customer Info \(customer_info\)](#)

ORACLE Web Determinations

[Summary](#) | [Data Review](#) [Save](#) | [Save As](#) | [Load](#) | [Restart](#) | [Close](#)

Rulebase: BonusFlightEligibility Locale: en-US User ID: guest

Customer Info

What is the customer's name?

Is the customer a member of the airline's loyalty program? * Yes No

What is the customer's airlines loyalty program membership date? *

Standard application copyright and disclaimer Version:10.4.0.1

Investigation: [Show in decision view](#)
Screen: [Flights Info \(flights_info\)](#)

ORACLE Web Determinations

[Summary](#) | [Data Review](#) [Save](#) | [Save As](#) | [Load](#) | [Restart](#) | [Close](#)

Rulebase: BonusFlightEligibility Locale: en-US User ID: guest

Flights Info

What is the flight's origin?

What is the flight's destination?

What is the flight's date?

What is the flight's trip distance (in kilometers)?

Remove

Modify the Velocity templates

The next task is to modify some Oracle Web Determinations templates to be able to pick-up the screen custom property value and hide the **Add New Instance** and **Remove Instance(s)** buttons as necessary. To do this, one must be familiar with the main template and control templates used by Entity-collect Question screens. Information on Oracle Web Determinations templates can be found in [Oracle Web Determinations Template Reference Guide](#).

To illustrate briefly, the entity collect screen is composed of the following templates:

Flights Info question_screen.vm

What is the flight's origin?	controls\EntityInstanceCollectGroup.vm	Paris	
What is the flight's destination?		Manila	
What is the flight's date?		12/22/11	
What is the flight's trip distance (in kilometers)?		10,742	

controls\ButtonGroup.vm
Remove

controls\ContainmentRelationshipControl.vm

controls\ButtonGroup.vm
Add New Instance
Remove Instance(s)

Essentially, what we would like to accomplish is to hide the *Remove* checkbox; and the **Add New Instance** and **Remove Instance(s)** buttons that you can see on the screen in the figure above. To do this, the following changes have to be implemented:

- `<OWD deploy dir>\WEB-INF\classes\templates\question_screen.vm`

- Set the screen custom property value to a variable.

```
## set custom property value
#set ($hideAddRemoveButtons = ${screen.getProperty("Hide Remove \ Add Instance buttons", "False")})
```

- `<OWD deploy dir>\WEB-INF\classes\templates\controls\ContainmentRelationshipControl.vm`

- Add if condition so that the **Add New Instance** and **Remove Instance(s)** buttons will not be displayed when **hideAddRemoveButtons** is "True".

```
#else ## default is to use portrait style
  #foreach( $control in $collectControls )
    ## hide Add New Instance and Remove Instance(s) buttons
    #if( ($control.getControlType() != "ButtonGroup") || ($control.getControlType() == "ButtonGroup" && $hideAddRemoveButtons))
      #parse( "controls/${control.getControlType()}.vm" )
    #end
  #end
#end
```

- `<OWD deploy dir>\WEB-INF\classes\templates\controls\EntityInstanceCollectGroup.vm`

- Add if condition so that the *Remove* checkbox will not be displayed when **hideAddRemoveButtons** is "True".

```

<div class="entity-instance-group">
  #foreach( $control in $entInstCollectControls )
    <div class="control-item">
      ## hide Remove radio button
      #if( {$control.getControlType() != "ButtonGroup"} || {$control.getControlType() == "ButtonGroup" && $hideAddRemoveButtons == "True"} )
        #parse( "controls/${control.getControlType()}.vm" )
      #end
    </div>
    <span class="control-clear"></span>
  #end
</div>

```

Test the modified templates with the rulebase

In order to test the modifications on the templates, we will create a simple Data Adaptor plugin. For testing purposes, it would simply pre-seed the interview with three "flight info" entity instances. The Java source code of the plugin is can be found in `<RUNTIME_ROOT_DIRECTORY>examples\web-determinations\hide-entity-collect-buttons\src`.

1. Compile the source code above to a JAR file. Call the resulting JAR file, *SimpleDataAdaptor.jar*.
2. Copy the *SimpleDataAdaptor.jar* file you have created to the plugins directory, where Oracle Web Determinations can find it - `<OWD deploy dir>\WEB-INF\classes\plugins`.
3. Then, deploy your rulebase in Oracle Web Determinations. Copy the rulebase's output file, *BonusFlightEligibility.zip*, to Oracle Web Determinations' rulebases directory - `<OWD deploy dir>\WEB-INF\classes\rulebases`.
4. Restart your server.
5. Open a web browser and navigate to Oracle Web Determinations, passing in any **caseID** to pre-seed the interview; for example, `<server>/<OWD context root>?caseID=777`.

ORACLE Web Determinations

Data Review

Save | Save As | Load | Restart | Close

Rulebase: BonusFlightEligibility Locale: en-US User ID: guest Case ID: 777

Assessment Summary

- [Click here to determine if the customer is eligible for a bonus flight](#)

6. Click on the goal on the *Summary* screen; you are taken to the *Customer Info* screen where you can input the following values:
 - a. "What is the customer's name?" - textbox should be pre-seeded with value "John Jones"
 - b. "Is the customer a member of the airline's loyalty program?" - tick "Yes" and click on the **Submit** button to make the next question visible.
 - c. "What is the customer's airlines loyalty program membership date?" - enter any date that is at least two years older than the current date; for example, 01/24/08.

ORACLE Web Determinations

[Summary](#) | [Data Review](#)

[Save](#) | [Save As](#) | [Load](#) | [Restart](#)

Rulebase: BonusFlightEligibility Locale: en-US User ID: guest Case

Customer Info

What is the customer's name?

John Jones

Is the customer a member of the airline's loyalty program?

* Yes No

What is the customer's airlines loyalty program membership date?

* 01/24/08

Submit

7. Click on the **Submit** button; you are taken to the *Flights Info* entity-collect screen where you can verify the following:
 - a. the screen is pre-seeded with three instances.
 - b. the *Remove* checkbox and the **Add New Instance** and **Remove Instance(s)** buttons are hidden.

ORACLE Web Determinations

[Summary](#) | [Data Review](#)

[Save](#) | [Save As](#) | [Load](#) | [Restart](#) | [Close](#)

Rulebase: BonusFlightEligibility Locale: en-US User ID: guest Case ID: 777

Flights Info

What is the flight's origin?

Paris

What is the flight's destination?

Manila

What is the flight's date?

12/22/11

What is the flight's trip distance (in kilometers)?

10,742

What is the flight's origin?

Dubai

What is the flight's destination?

Sydney

What is the flight's date?

4/26/11

What is the flight's trip distance (in kilometers)?

12,055

What is the flight's origin?

Rome

What is the flight's destination?

Singapore

What is the flight's date?

9/23/11

What is the flight's trip distance (in kilometers)?

10,027

Submit

8. Click on the **Submit** button; the goal is inferred and the interview completed.

ORACLE Web Determinations

Data Review

Save | Save As | Load | Restart | Close

Rulebase: BonusFlightEligibility Locale: en-US User ID: guest Case ID: 777

Assessment Summary

- The customer is eligible for a bonus flight. [\[Why?\]](#)

9. Alternatively, you can set the rulebase's "Hide Remove / Add Instance button" custom property in Oracle Policy Modeling to "False" for the *Flights Info* entity collect screen.
10. Rebuild the rulebase and re-deploy the rulebase output file to the server.
11. Re-start the server and start the interview.
12. On the *Flights Info* entity collect screen, verify that the *Remove* checkbox and the **Add New Instance** and **Remove Instance(s)** buttons are all visible.

ORACLE Web Determinations

Summary | Data Review

Save | Save As | Load | Restart | Close

Rulebase: BonusFlightEligibility Locale: en-US User ID: guest Case ID: 777

Flights Info

What is the flight's origin? Paris

What is the flight's destination? Manila

What is the flight's date? 12/22/11

What is the flight's trip distance (in kilometers)? 10,742

Remove

What is the flight's origin? Dubai

What is the flight's destination? Sydney

What is the flight's date? 4/26/11

What is the flight's trip distance (in kilometers)? 12,055

Remove

What is the flight's origin? Rome

What is the flight's destination? Singapore

What is the flight's date? 9/23/11

What is the flight's trip distance (in kilometers)? 10,027

Remove

Add New Instance Remove Instance(s)

Submit

Example: Assess Request xml

Below are examples of an Assess Request for the rulebase *SimpleBenefits* for both the generic and specific services. The *SimpleBenefits* rulebase can be found in [examples\rulebases\SimpleBenefits.zip](#).

Example Assess Request (generic)

The Assess Request needs to be wrapped in a standard SOAP Envelope, which contains a header, and a body. Lines 1-3 are the starting of the SOAP request. The body and envelope are closed on lines 28 and 29.

When the soap element is begun the namespace for the elements used in all the Determinations Server requests is imported through the declaration "`xmlns:typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types"`".

Lines 4-27 are the assess request and are all from the Determinations Server namespace. The assess request is doing the following:

- Creating a global-instance element for the global entity. This element contains the child entity element (lines 5-26).
- Creating an entity element for the child entity and two child instances (lines 14-25).
- In the global-instance element, specifying 2 attribute outcomes, to return with the outcome style "value-only". The outcomes being requested are `eligible_low_income_allowance` and `eligible_teenage_allowance` (lines 6-7).
- In the global entity instance, setting the attribute `claimant_income` to the number value 13000 (lines 8-10). (Number values are used for currency and number values.)
- In the global entity instance, setting the attribute `claimant_public_housing_client` to the Boolean value 'true' (lines 11-13).
- Creating two child entity instances (`child1` and `child2`) and setting the attribute `child_age` for both to the number values 16 and 8 respectively (lines 14-25).

Important Note: do not set value and specify outcome style for same attribute

Do not both set a value for an attribute and specify that attribute as an outcome in the same Assess Request. If you do this, any value set for the attribute in the Request will be ignored (in other words, only the "outcome-stlye" part of the Request will be processed for that attribute).

(The reason for this behavior is that the Response provided by the service is in the same format as the Request, to make it easier for scenarios where you wish to call the Determinations Server repeatedly with partial data in order for it to provide you with information about what additional relevant information you need to collect).

Generic Assess Request xml

```
1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
  typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
2 <soapenv:Header/>
3 <soapenv:Body>
4 <typ:assess-request>
5 <typ:global-instance>
6 <typ:attribute id="eligible_low_income_allowance" outcome-style="value-only"/>
7 <typ:attribute id="eligible_teenage_allowance" outcome-style="value-only"/>
8 <typ:attribute id="claimant_income">
9 <typ:number-val>13000</typ:number-val>
10 </typ:attribute>
11 <typ:attribute id="claimant_public_housing_client">
```

```

12     <typ:boolean-val>true</typ:boolean-val>
13 </typ:attribute>
14 <typ:entity id="child">
15   <typ:instance id="child1">
16     <typ:attribute id="child_age">
17       <typ:number-val>16</typ:number-val>
18     </typ:attribute>
19   </typ:instance>
20   <typ:instance id="child2">
21     <typ:attribute id="child_age">
22       <typ:number-val>8</typ:number-val>
23     </typ:attribute>
24   </typ:instance>
25 </typ:entity>
26 </typ:global-instance>
27 </typ:assess-request>
28 </soapenv:Body>
29 </soapenv:Envelope>

```

Example Assess Request (specific)

The specific assess request for the same rulebase needs the same standard soap wrapper as the generic request above (lines 1-3, 28-29).

The namespace declaration on line 1 is for the specific service and named for (and specific to) the *SimpleBenefits* rulebase: "`xmlns:typ="http://oracle.com/determinations/server/10.4/SimpleBenefits/assess/types"`".

Lines 4-27 are the specific assess request and are all from the SimpleBenefits namespace. The assess request is setting and request exactly the same data as the generic assess request.

- Creating a global instance (lines 5-26).
- Creating a list-child element for the child entity and two child instances (lines 14-25).
- Attribute outcomes for `eligible_low_income_allowance` and `eligible_teenage_allowance` on the global entity instance (lines 6-7).
- Setting the attribute for `claimant_income` and `claimant_public_housing_client` to 13000 and 'true' respectively (lines 8-13).
- Creating two child entity instances (child1 and child2) and setting the `child_age` attributes for both to 16 and 8 respectively (lines 15-24).

Specific Assess Request xml

```

1 <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:typ="http://oracle.com/determinations/server/10.4/SimpleBenefits/assess/types">
2 <soapenv:Header/>
3 <soapenv:Body>
4 <typ:assess-request>
5 <typ:global-instance>
6 <typ:eligible_low_income_allowance outcome-style="value-only"/>
7 <typ:eligible_teenage_allowance outcome-style="value-only"/>
8 <typ:claimant_income>

```

```
9     <typ:number-val>13000</typ:number-val>
10  </typ:claimant_income>
11  <typ:claimant_public_housing_client>
12    <typ:boolean-val>true</typ:boolean-val>
13  </typ:claimant_public_housing_client>
14  <typ:list-child>
15    <typ:child id="child1">
16      <typ:child_age>
17        <typ:number-val>16</typ:number-val>
18      </typ:child_age>
19    </typ:child>
20    <typ:child id="child2">
21      <typ:child_age>
22        <typ:number-val>8</typ:number-val>
23      </typ:child_age>
24    </typ:child>
25  </typ:list-child>
26 </typ:global-instance>
27 </typ:assess-request>
28 </soapenv:Body>
29 </soapenv:Envelope>
```

Assess Request Best Practice Tips

It is recommended you limit your Assess Request to the bare minimum of elements to return what you require. In particular:

- For an attribute you are setting a value for in the Request, only provide the value, do not set the "type" and "inferred" attributes. These attributes are typically returned in the Response and cannot be set by the Request.
- For an attribute you want the Response to return an outcome for, set either
 - the "outcome-style", or
 - (if you want different types of outcomes) both the "unknown-outcome-style" and "known-outcome-style". Note: it is not necessary or recommended that you specify all three.
- Do not set the outcome style of an attribute that you are setting a value for. It should always be returned in the Response. For more information, see [Important Note: do not set value and specify outcome style for same attribute](#).

Example: Assess Response xml

The request in the [previous example](#) will result in the following response from a Determinations Server.

Example Assess Response (generic)

The response is returned in a SOAP wrapper identical to the request. The response also returns all the values that were set in the request, so we can see the global and two child entity instances, and the attributes set in the request.

The attribute-outcome elements set in the request, have been returned as attributes in the response. In this case the values were known, so the attribute `eligible_low_income_allowance` has a boolean value of `true` (lines 11-13) and the attribute `eligible_teenage_allowance` also has the boolean value of `true` (lines 14-16).

Generic Assess Response xml

```
1 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/rulebase/assess/types">
2 <SOAP-ENV:Header>
3 <i18n:international>
4 <i18n:locale>en_US</i18n:locale>
5 <i18n:tz>GMT+0800</i18n:tz>
6 </i18n:international>
7 </SOAP-ENV:Header>
8 <SOAP-ENV:Body>
9 <typ:assess-response>
10 <typ:global-instance>
11 <typ:attribute id="eligible_low_income_allowance" type="boolean" inferred="true">
12 <typ:boolean-val>true</typ:boolean-val>
13 </typ:attribute>
14 <typ:attribute id="eligible_teenage_allowance" type="boolean" inferred="true">
15 <typ:boolean-val>true</typ:boolean-val>
16 </typ:attribute>
17 <typ:attribute id="claimant_public_housing_client" type="boolean">
18 <typ:boolean-val>true</typ:boolean-val>
19 </typ:attribute>
20 <typ:attribute id="claimant_income" type="currency">
21 <typ:number-val>13000.0</typ:number-val>
22 </typ:attribute>
23 <typ:entity id="child" inferred="false">
24 <typ:instance id="child1">
25 <typ:attribute id="child_age" type="number">
26 <typ:number-val>16.0</typ:number-val>
27 </typ:attribute>
28 </typ:instance>
29 <typ:instance id="child2">
30 <typ:attribute id="child_age" type="number">
31 <typ:number-val>8.0</typ:number-val>
32 </typ:attribute>
33 </typ:instance>
34 </typ:entity>
```

```
35 </typ:global-instance>
36 </typ:assess-response>
37 </SOAP-ENV:Body>
38 </SOAP-ENV:Envelope>
```

Example Assess Response (specific)

The response for the specific service returns the same answers as the generic service. It is returned in a SOAP wrapper, and also returns all the values that were set in the request.

The attribute-outcome elements set in the request, have been returned as attributes in the response. In this case the values were known, so the attribute `eligible_low_income_allowance` has a boolean value of `true` (lines 11-13) and the attribute `eligible_teenage_allowance` also has the boolean value of `true` (lines 14-16).

Specific Assess Response xml

```
1 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/10.4/SimpleBenefits/assess/types">
2 <SOAP-ENV:Header>
3 <i18n:international>
4 <i18n:locale>en_US</i18n:locale>
5 <i18n:tz>GMT+0800</i18n:tz>
6 </i18n:international>
7 </SOAP-ENV:Header>
8 <SOAP-ENV:Body>
9 <typ:assess-response>
10 <typ:global-instance>
11 <typ:eligible_low_income_allowance type="boolean" inferred="true">
12 <typ:boolean-val>true</typ:boolean-val>
13 </typ:eligible_low_income_allowance>
14 <typ:eligible_teenage_allowance type="boolean" inferred="true">
15 <typ:boolean-val>true</typ:boolean-val>
16 </typ:eligible_teenage_allowance>
17 <typ:claimant_public_housing_client type="boolean">
18 <typ:boolean-val>true</typ:boolean-val>
19 </typ:claimant_public_housing_client>
20 <typ:claimant_income type="currency">
21 <typ:number-val>13000.0</typ:number-val>
22 </typ:claimant_income>
23 <typ:list-child inferred="false">
24 <typ:child id="child1">
25 <typ:child_age type="number">
26 <typ:number-val>16.0</typ:number-val>
27 </typ:child_age>
28 </typ:child>
29 <typ:child id="child2">
30 <typ:child_age type="number">
31 <typ:number-val>8.0</typ:number-val>
32 </typ:child_age>
33 </typ:child>
```

```
34     </typ:list-child>
35   </typ:global-instance>
36 </typ:assess-response>
37 </SOAP-ENV:Body>
38 </SOAP-ENV:Envelope>
```

Example: Configuration for classpath-based loading (Java)

The following is an example of the [application.properties](#) file, configuring Oracle Web Determinations to load everything from the classpath.

The following assumes that the resources are placed into folders called configuration, images, resources, rulebases and templates under the WEB-INF/classes folder in the web application's deployment folder.

Note:

This works for both an exploded and unexploded WAR deployment.

```
#####  
# Core application properties, these are used to configure the application and are not available to  
# screens.  
#####  
# The locale to default to if we're displaying a screen that is not attached to a specific locale.  
  
default.locale =en-US  
  
# Set enable.debugger to true if you need to be able to debug from Oracle Policy Modeling  
enable.debugger =true  
  
#####  
# Rulebase Loading Properties  
# If rulebases are to be loaded as resource streams, this  
# property specified whether or not the Java classpath  
# is to be used to load the resources.  
  
load.rulebase.from.classpath =true  
  
rulebase.path =/WEB-INF/classes/rulebases  
cache.loaded.rulebases =false  
enable.second.person =true  
  
# Screens file content can include html authored by users in oracle policy modelling as static content.  
# These options determine whether to scan the content at application start time and verify that the  
# tags deployed in the rulebase are in the whitelist of allowable content.  
  
screens.validate.html =true  
  
# any tag not on this list will cause an exception to be thrown during rulebase loading and the rulebase  
# will not be available.  
# if additional tags are required they must be added to this list.  
  
screens.html.tags.whitelist  
=b;i;del;s;div;p;span;pre;table;td;tr;ol;ul;li;blockquote;font;a;h1;h2;h3;h4;h5;h6;img;hr;br;u
```

```
#####
```

```
# Resourcing Properties
```

```
load.messages.as.resource =true
```

```
messages.path =configuration
```

```
cache.messages =false
```

```
load.images.as.resource =true
```

```
images.path =images
```

```
cache.images =false
```

```
load.resource.as.resource =true
```

```
resources.path =resources
```

```
cache.resources =false
```

```
load.properties.as.resource =true
```

```
properties.path =configuration
```

```
cache.properties =false
```

```
load.templates.as.resource =true
```

```
templates.path =templates
```

```
cache.templates =false
```

```
load.static.content.as.resource =false
```

```
static.content.path =content
```

```
#####
```

```
# Plugins Properties
```

```
# This is a java only property used for web application servers where class path introspection is not  
# possible such as WebLogic when the war file is not exploded. In that case, this property contains the ';'   
# list of fully qualified plugin classes to load and the plugin libraries must reside in a directory that will  
# automatically loaded on the classpath
```

```
plugin.libraries=
```

```
# Default XDS data adaptor file path
```

```
xds.file.path=data
```

```
#####
```

```
# MIME type mappings for resource extensions
```

```
extension.xml =text/xml
```

```
extension.htm =text/html
```

```
extension.html =text/html
```

```
extension.css =text/css
```

```
extension.js =text/javascript
```

extension.txt =text/plain

extension.gif =image/gif

extension.jpeg =image/jpeg

extension.jpg =image/jpeg

extension.png =image/png

extension.svg =image/svg+xml

extension.tiff =image/tiff

extension.ico =image/vnd.microsoft.icon

extension.pdf =application/pdf

extension.zip =application/zip

See also:

[application.properties file](#)

[Example: Configuration for file system-based loading \(Java and .NET\)](#)

Example: Configuration for file system-based loading (Java and .NET)

The following is an example of the [application.properties](#) file, configuring Oracle Web Determinations to load everything from the file system.

The following assumes that the Java version of the application is being run, and the resources are placed into folders called **configuration**, **images**, **properties**, **resources**, **rulebases** and **templates** under the *WEB-INF/classes* folder in the web application's deployment folder.

Notes:

- The following **will not work** under Java for application servers that do not explode the WAR (for example, WebLogic) - in this case, the only option available is to use [classpath-based resource loading](#).
- Under .NET, the paths to the resources must be changed to the relative paths under the application's directory; for example, 'bin/rulebases', 'bin/configuration' and so on.
- The **cache.loaded.rulebases=false** property, combined with the fact that we are loading the rulebases from the file system, will cause a directory watcher to be invoked on the rulebases directory. This allows rulebases to be dynamically loaded, removed and updated as the contents of the rulebases directory changes.

```
#####  
# Core application properties, these are used to configure the application and are not available to  
# screens.  
#####  
# The locale to default to if we're displaying a screen that is not attached to a specific locale.  
default.locale =en-US  
# Set enable.debugger to true if you need to be able to debug from Oracle Policy Modeling  
enable.debugger =true  
  
#####  
# Rulebase Loading Properties  
  
# If rulebases are to be loaded as resource streams, this  
# property specified whether or not the Java classpath  
# is to be used to load the resources.  
load.rulebase.from.classpath =false  
  
rulebase.path =/WEB-INF/classes/rulebases  
cache.loaded.rulebases =false  
enable.second.person =true  
  
# Screens file content can include html authored by users in oracle policy modelling as static content.  
# These options determine whether to scan the content at application start time and verify that the  
# tags deployed in the rulebase are in the whitelist of allowable content.  
screens.validate.html =true  
# any tag not on this list will cause an exception to be thrown during rulebase loading and the rulebase will not
```

be available.

if additional tags are required they must be added to this list.

screens.html.tags.whitelist =b;i;del;s-

s;div;p;span;pre;table;td;tr;ol;ul;li;blockquote;font;a;h1;h2;h3;h4;h5;h6;img;hr;br;u

#####

Resourcing Properties

load.messages.as.resource =true

messages.path =configuration

cache.messages =true

load.images.as.resource =true

images.path =images

cache.images =true

load.resource.as.resource =true

resources.path =resources

cache.resources =true

load.properties.as.resource =true

properties.path =configuration

cache.properties =false

load.templates.as.resource =true

templates.path =templates

cache.templates =false

load.static.content.as.resource =false

static.content.path =content

#####

Plugins Properties

This is a java only property used for web application servers where class path introspection is not possible
such as WebLogic when the war file is not exploded. In that case, this property contains the ';' list of fully
qualified

plugin classes to load and the plugin libraries must reside in a directory that will automatically loaded on the
classpath

plugin.libraries =

See also:

[Configuration files](#)

[Example: Configuration for classpath-based loading \(Java\)](#)

Example: Create a Custom Function extension to capitalize text

A custom function extension is a method for a software engineer to make new functions available on a per-project basis.

In this example a function is required to capitalize a name entered by a user. For example, if a user enters the name "fred jones", the rulebase needs to present this as "Fred Jones". Since no built-in function exists to do this and there is no way to combine existing functions to do this, it is a good candidate for a custom function.

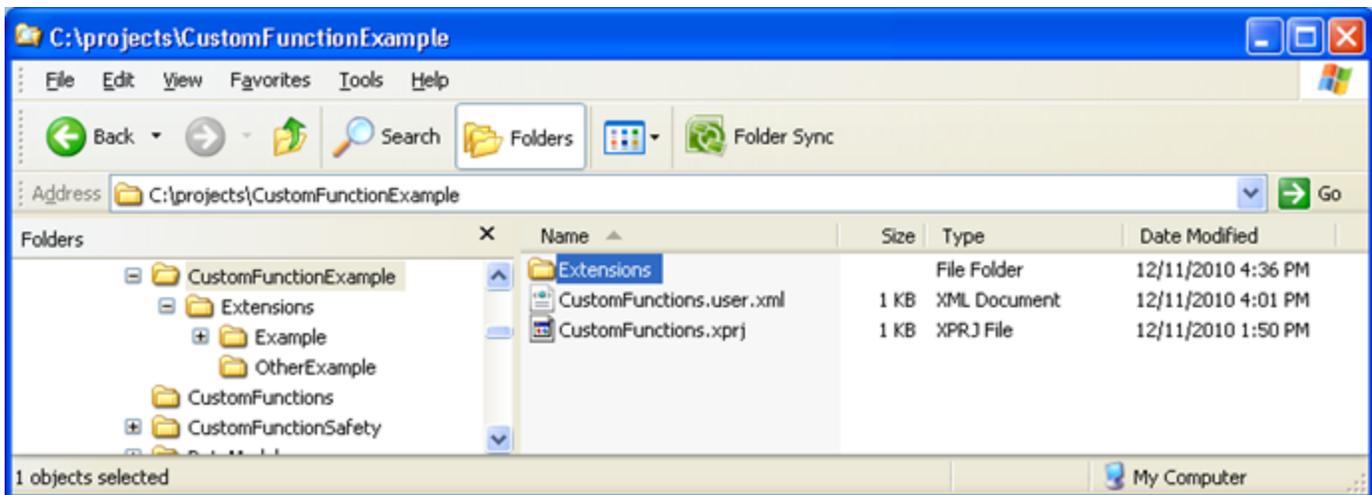
The new function is called "Capitalize" and it accepts one text value and returns another text value. This is defined in an extension.xml file (see Reference:Custom Function Extensions for the schema).

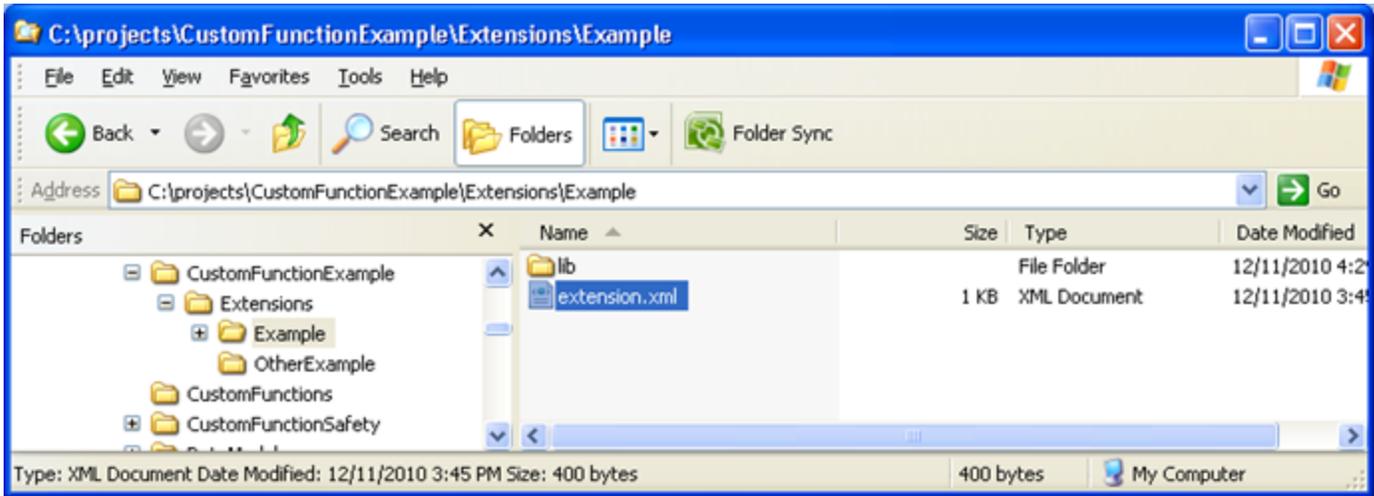
The function also needs a Java or .NET class in which the function is implemented. In this example both a Java and a .NET handler are provided, however either one of these could be left out. In practice a .NET implementation is always highly recommended so the rulebase can be used in the debugger, but it is not required if the rulebase will only be executed in a Java environment.

```
<extension>
  <functions>
    <function name="Capitalize" return-type="text">
      <arg name="entered-name" type="text"/>

      <handler platform="java"
        class="com.oracle.determinations.examples.CapitalizeFunction"/>
      <handler platform="dotnet"
        class="Oracle.Determinations.Examples.CapitalizeFunction"/>
    </function>
  </functions>
</extension>
```

The above extension.xml file should be saved inside a new subfolder of the project's "Extensions" folder. This "Extensions" folder may also have to be manually created.

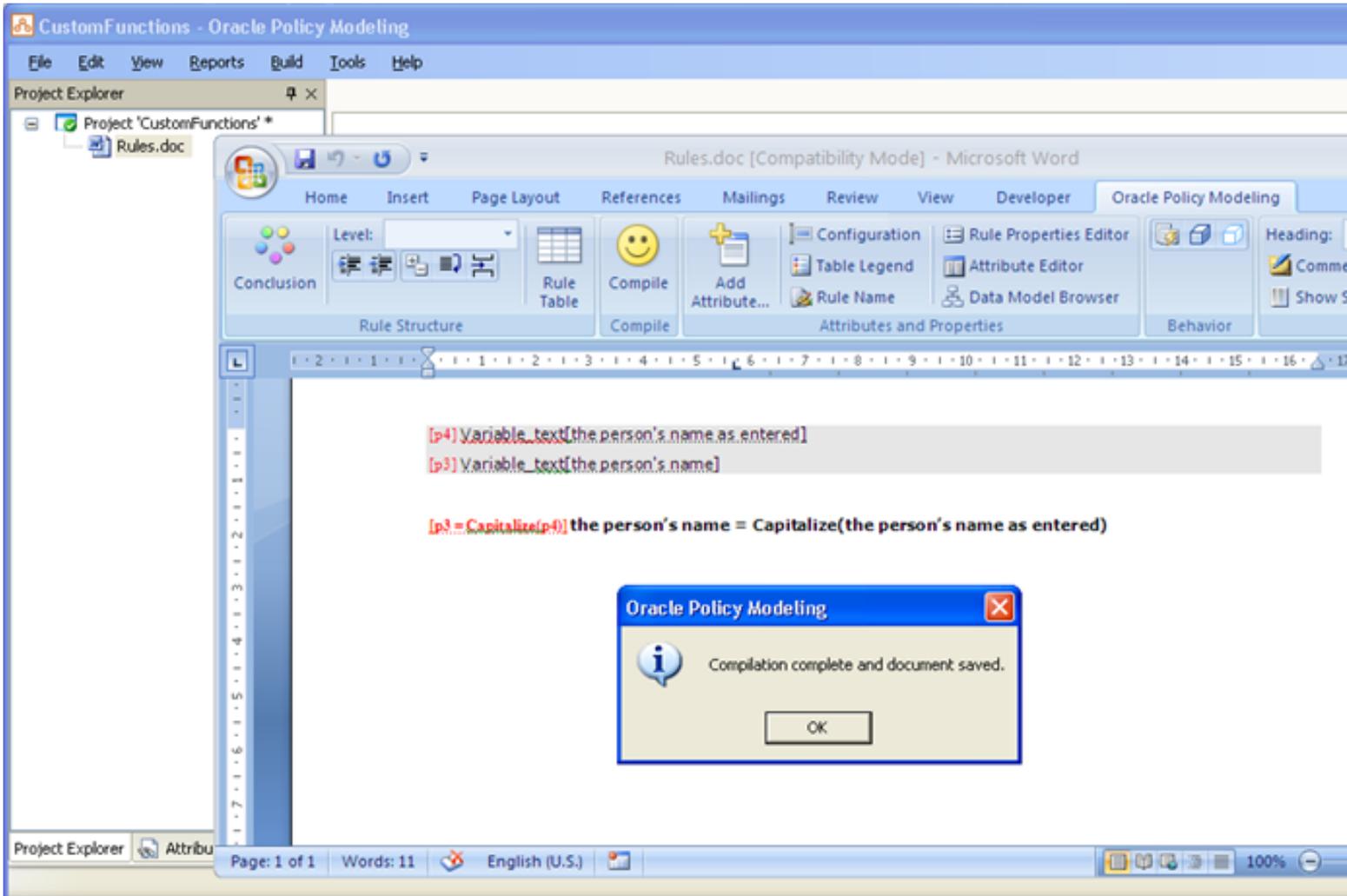




Note that the `extension.xml` file is now located in `Extensions\Example\extension.xml`, which is two levels away from the project folder (the folder containing the `xproj` file). This allows several extension folders to be copied into the extensions folder and they will all be active at once, for example in the above screenshot there is a second extension called "OtherExample" in the extensions folder.

In addition to the `extension.xml` file, a "lib" folder should be created to store the compiled code that implements the custom function. This folder will contain compiled assembly DLLs (for .NET) and JAR files (for Java).

With this in place, it is possible to compile a rule document that uses the custom function.



Attempting to build and debug this rulebase will result in an error because the class named in the extension.xml file is not available. The extension's lib folder must be populated with the code that implements the custom function.

The rulebase to use for this example can be found at:

`examples\determinations-engine\custom-functions\rulebase` in the appropriate runtime zip

For sample source code, go to:

Java:

`examples\determinations-engine\custom-functions\src` in Java runtime zip

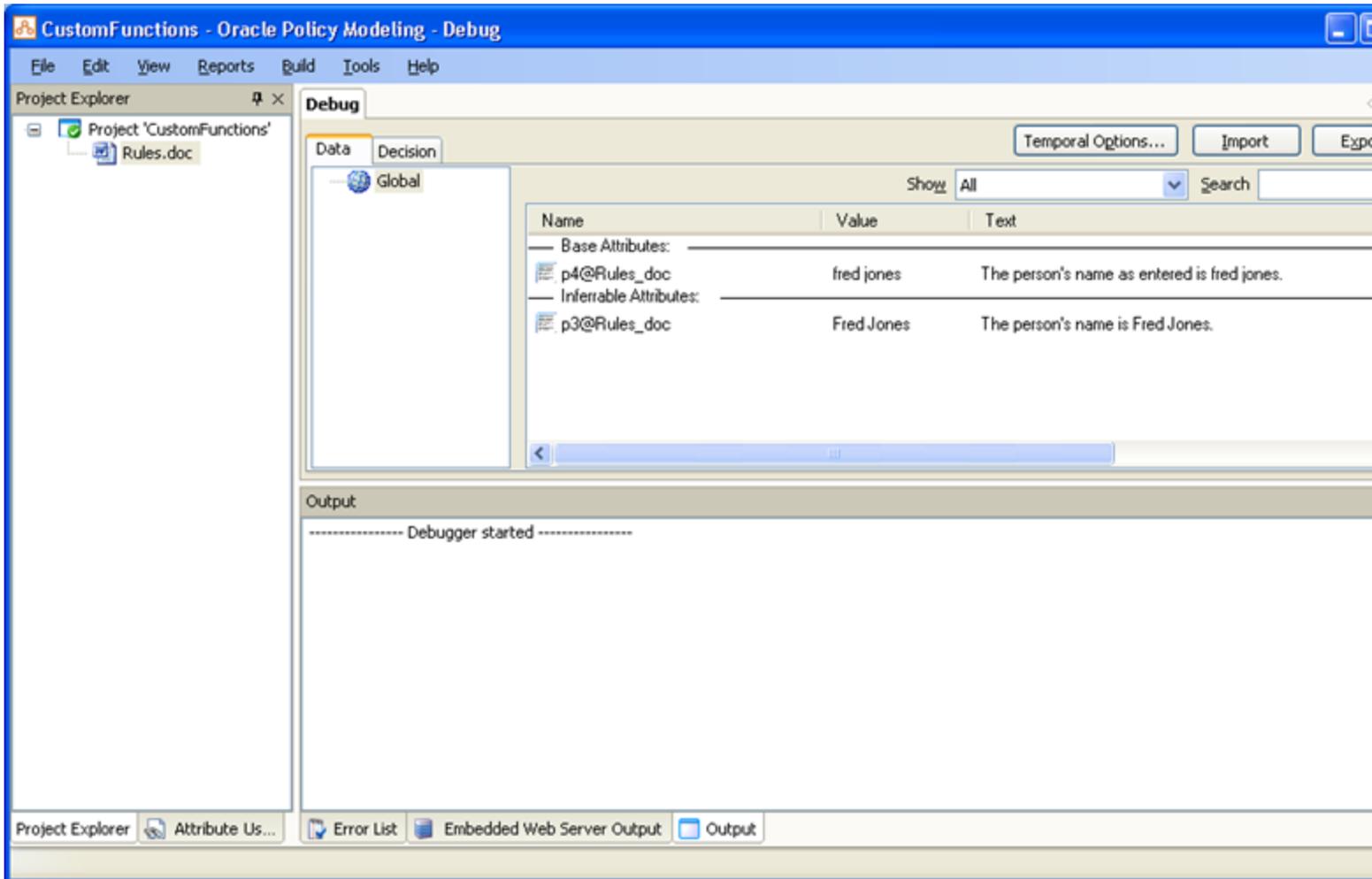
C#:

`examples\determinations-engine\custom-functions\src\csharp` in .NET runtime zip

VB:

examples\determinations-engine\custom-functions\src\vb in .NET runtime zip

The code at the above addresses should be compiled into a Java JAR or .NET assembly DLL and copied into the extension's lib folder. The rulebase can then be built and debugged, either without screens (if a .NET assembly DLL is provided) or with screens (if a Java JAR is provided).



See also:

[Write a Custom Function extension](#)

[Custom Function extensions](#)

[Example: Create a Custom Function extension to default a value](#)

Example: Create a Custom Function extension to default a value

Note: This example is somewhat more advanced than the example [Create a Custom Function to capitalize text](#) and builds on the concepts already demonstrated there.

By default a custom function does not see unknown and uncertain value, and their presence will automatically return unknown or uncertain as the result. However some functions can return a value even when some parameters are unknown.

In this example, a function is required to assume some default value in the case where a question hasn't been answered; that is, when it is unknown.

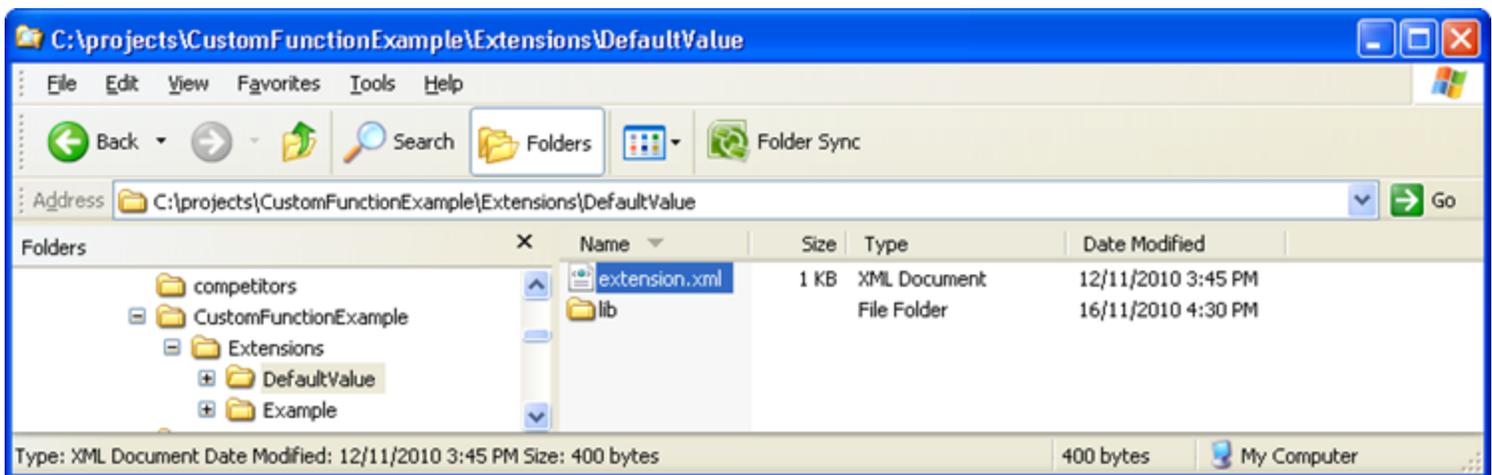
Note: Because this example function transforms unknown values into known values, it may cause a rulebase to prematurely draw conclusions when required values have not been asked. If a distinction is required between a 'genuine' conclusion and a 'default' conclusion, use of this function may not be suitable.

The function is defined inside the *extension.xml* file as follows:

```
<extension>
  <functions>
    <function name="DefaultNumber" return-type="number">
      <arg name="value" type="number"/>
      <arg name="default" type="number"/>

      <handler platform="java"
        class="com.oracle.determinations.examples.DefaultFunction"/>
      <handler platform="dotnet"
        class="Oracle.Determinations.Examples.DefaultFunction"/>
    </function>
  </functions>
</extension>
```

This file is saved inside an extension folder, which is then itself copied into the project's "Extensions" folder.



The class that implements this function is called **DefaultFunction**, derives from the **CustomFunction** class and must override three methods:

- **evaluate()** – to implement the required behaviour. Note that 'unknown' is represented with null.
- **requireKnownParameters()** – to return false. Without this override, the engine would not call the function when the first parameter was unknown, it would always return unknown on the function's behalf.
- **markRelevance()** – overridden to tell the engine which parameters should appear in a decision report, and which questions to ask during an investigation. Without this override, the engine would assume both parameters were always relevant, but in reality the second parameter is only relevant (that is, it only affects the result) when the first parameter is unknown. In practice this override is nearly always required when **requireKnownParameters** returns false.

The rulebase to use for this example can be found at:

[examples\determinations-engine\custom-functions\rulebase](#) in the appropriate runtime zip.

For sample source code, go to:

Java:

[examples\determinations-engine\custom-functions\src](#) in Java runtime zip.

C#:

[examples\determinations-engine\custom-functions\src\csharp](#) in .NET runtime zip.

Once this code is compiled into a Java JAR file or .NET assembly DLL, it should be copied into the 'lib' folder of the extension and can then be used in the project:

Conclusion Level: Rule Table Compile Add Attribute... Configuration Table Legend Rule Name Data Model Browser Rule Properties Editor Attribute Editor

[p5] Variable_number(the assumed number of children)
[p7] Variable_number(the default number of children)
[p6] Variable_number(the number of children)
[p5 = DefaultNumber(p6,p7)] the assumed number of children = DefaultNumber(the number of children, the default number of children)

Oracle Policy Modeling
Compilation complete and document saved.
OK

CustomFunctions - Oracle Policy Modeling - Debug

File Edit View Reports Build Tools Help

Project Explorer

Project 'CustomFunctions'
Rules.doc

Debug

Data Decision

Global

Temporal Options...

Import

Show All

Search

Name	Value	Text
Base Attributes:		
123 p6@Rules_doc	<unknown>	The number of children is unknown.
123 p7@Rules_doc	0.0	The default number of children is 0.
Inferable Attributes:		
123 p5@Rules_doc	0.0	The assumed number of children is 0.

Output

----- Generating Rulebase Model: Project - CustomFunctions -----

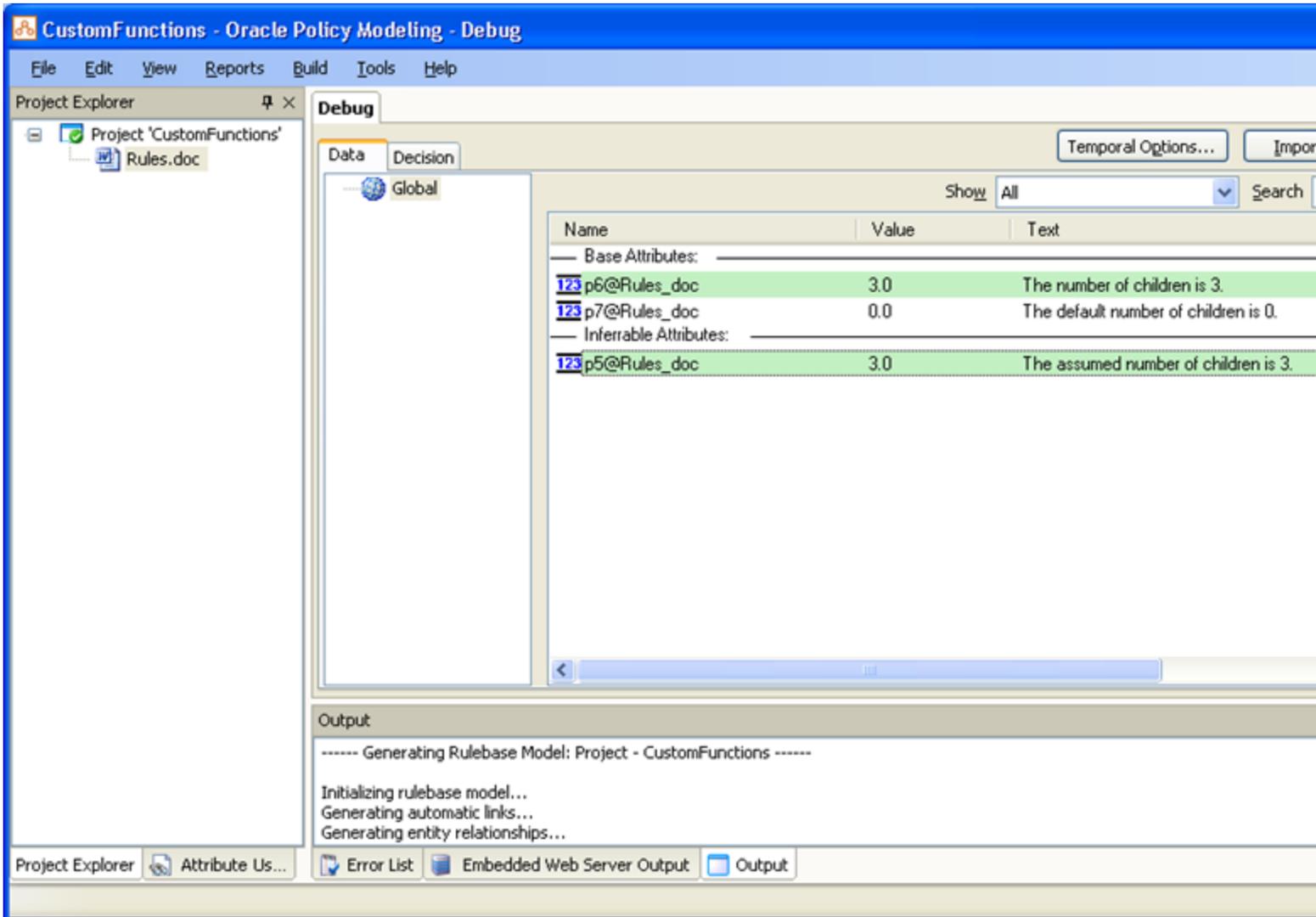
Initializing rulebase model...
Generating automatic links...
Generating entity relationships...

Project Explorer Attribute Us...

Error List

Embedded Web Server Output

Output



See also:

[Write a Custom Function extension](#)

[Custom Function extensions](#)

[Example: Create a Custom Function extension to capitalize text](#)

Example: Create a Custom Screen for the Interview Portlet

This example illustrates how to develop a custom screen to be used in the Oracle Policy Automation Interview Portlet. Specifically, it shows:

- Rendering the screen using a custom Velocity template.
- Issuing a redirect to another URI within the Interview Portlet, or to an external web application.
- Extracting submitted POST data and storing it in the session.

The source code for the example is available in the [examples/interview-portlet/custom-screen](#) directory of the Java runtime zip. The corresponding rulebase can be found at [examples/rulebases/compiled/CustomScreenExample.zip](#); the example source code consists of the following three classes:

ExampleScreen

This class implements the **CustomScreen** interface and is responsible for rendering the screen (**processAndRespond** method), as well as processing POST data submitted to the screen (**submit** method).

Importantly for the portlet, the **processAndRespond** method is invoked during the *Render* phase of the portlet and therefore is not able to issue external redirects. It can either return a **DisplayScreenResponse** or a **RedirectResponse** to another URI within the portlet.

ExampleScreenProvider

The Web Determinations platform looks for this class each time a screen is to be displayed. It determines if the screen to be displayed will be replaced by a custom screen, and if so, returns an instance of **ExampleScreen**.

ExternalRedirector

This class is an event handler for the portlet's **OnAfterProcessActionEvent**. Depending on whether the value of a certain rulebase attribute is set, it replaces the response about to be rendered with an **ExternalRedirectResponse**. The portlet then consumes this **ExternalRedirectResponse** and issues an external redirect just before the end of the action phase.

Example: Create a Rulebase Listener to preload reference data

Many rulebases have a need for reference data; that is, data that should ideally be available to the rules but for whatever reason cannot be hard-coded into the rules themselves. For example, large tables of data are easier to write rules for, if they are structured as attributes in entities.

The supplied example is a Rulebase Listener that reads data from a CSV (comma-separated values) file located in the project's "include" directory, and uses the data to pre-populate sessions that are created with the rulebase.

The CSV file is named after the entity for which it contains data; for example, *income-bracket.csv*.

The first row in the CSV file contains the public names of attributes, and each of the following rows define an entity instance with the attribute values of that row.

The CSV file and compiled rulebase listener is packaged inside the compiled rulebase zip file, so the rulebase remains self-contained.

The **ReferenceDataListener** class contains the main logic and the class implements **RulebaseListener** as follows:

- The **initialize()** method looks for CSV files inside the compiled rulebase ZIP that have the same name as an entity's public name; for example, *income-bracket.csv*. It parses and then stores the parsed **ReferenceData** object for later use.
- The **sessionCreated()** method enumerates each entity that had reference data associated, creates the entity instances and sets the attribute values.

The **ReferenceDataCSVParser** class handles number, boolean, date/datetime and text values. If a row has too many values, an exception is thrown and if a row has too few values, the extra attributes are set as uncertain.

The example has the following limitations:

- To be preloaded, entities must be contained by the global entity. An entity that is two or more steps removed from the global entity will be ignored.
- Relationships, temporal values, unknown values, and text values with commas in them are not supported.
- Preloaded data is indistinguishable in the determinations engine from ordinary user-supplied data, and if a session is saved, reference data will be saved with it. This should not be an issue with Oracle Determinations Server (since it does not save sessions) but any other application that expects to be able to reload sessions should filter this data out before doing so.

The example source code and rulebase

For the sample source code and rulebase required to run this example, go to:

Java:

`examples\determinations-engine\reference-data\java` in Java runtime zip

C#:

`examples\determinations-engine\reference-data\csharp` in .NET runtime zip

Example: Create test cases in the Batch Processor

The Batch Processor can be used to generate test cases, that can be attached to a rulebase project. The input data can be from either .csv files or from a database and can be used to generate an individual test case.

Before you start, unzip the *InsuranceFraudScore* rulebase project that is found at:

```
examples\rulebases\source\InsuranceFraudScore.zip
```

Run the Batch Processor

1. Open a command prompt and go to the `examples/determinations-batch/InsuranceFraudScore`.
2. From the command line execute the Batch Processor - this is slightly different for Java or .NET.

JAVA

```
java -jar ../../../../engine/determinations-batch.jar --config fraud_score_testcases_config.xml
```

.NET

```
..\..\..\engine\Determinations.Batch.exe --config fraud_score_testcases_config.xml
```

The Batch Processor will run, executing across the data files in the csv directory. It will write the results to a *test_cases.tsc* file.

Import the Test Case into the InsuranceFraudScore rulebase

1. Start Oracle Policy Modeling.
2. Open the *InsuranceFraudScore* rulebase; the Oracle Policy Modeling project file will be in the rulebase project that was unzipped and will be in `Development\InsuranceFraudScore.xprj`.

3. From the **File** menu choose **Add --> Add Existing File**.

InsuranceFraudScore - Oracle Policy Modeling

File Edit View Reports Build Tools Help

- New Project... Ctrl+N
- Open Project... Ctrl+O
- Close Project

- Import Project...
- Export...

Add

- Save Selected Item Ctrl+S
- Save All Ctrl+Shift+S

- Project Properties...
- Project Statistics
- Edit Verbs...

- 1 D:\OPA_10\...\InsuranceFraudScore.xprj
- 2 D:\OPA_10\...\InsuranceFraudScore.xprj
- 3 D:\...\Development\AdminSmokeTest.xprj
- 4 D:\Jira\...\Parents and Children.xprj

- Exit

- Add New Word Document
- Add New Excel Document
- Add New Translation Document
- Add New What-If Analysis Document
- Add New Screens File
- Add New Properties File
- Add New Visual Browser File
- Add New Test Script File

- Add New Folder
- Add Existing File...
- Add Existing Folder...

- Add Module Link...

Output

Creating directory: C:\Documents and Settings\frankh\L
Output path: D:\OPA_10\10.4.0_work\Oracle_Policy_Au

4. Choose the *batch_test_cases.tsc* file generated from the batch run; the test cases should be added to the Policy Modeling Project.

Examine the test cases

Double-click on the *batch_test_cases.tsc* test file to open the test cases; you may be prompted to recompile the project before the test cases file is opened.

You should see 10 test cases that have been created from the ten records in the csv files when the batch processor was run.

Once the test cases have been imported into Policy Modeling you can add outcomes and expected values for each case; this will allow you to use the test cases to check the future impact of any rule changes.

For more information on using test cases, see the Oracle Policy Modeling Help topic *Create test scripts from existing data*.

Project Explorer

- Project 'InsuranceFraudScore'
 - Interviews
 - Properties
 - Rules
 - Test Scripts
 - batch_test_cases.tsc
 - Test script.tsc
 - Visualisations
 - Days since inceptions (points).dml
 - Total Fraud Score.dml
 - Young driver fronting.dml

batch_test_cases.tsc

Test Cases Outcomes

- batch_test_cases
 - case1
 - case2
 - case3
 - case4
 - case5
 - case6
 - case7
 - case8
 - case9
 - case10

Output

----- Done -----

Warning: This test script has no defined outcomes.

Look at the configuration

The configuration file used to generate the test cases for this example is almost identical to the configuration file used for the basic *InsuranceFraudScore* example. The only difference is the change of output from csv to test cases which is done by changing a single line.

The following line, specifies the output as test cases which will be written to the *batch_test_cases.tsc* file:

```
<output type="exporttsc">batch_test_cases.tsc</output>
```

Determinations Server Custom Service Example

This example demonstrates a custom service for Determinations Server and illustrates how it is possible to load reference data before calling the Assess method.

It deals with the scenario of a real-time inquiry for a claimant's medical outpatient benefits. When a claimant visits a clinic for a check-up, the system should be able to return all of the outpatient services for which the claimant is eligible and is currently entitled to.

To achieve this, we develop a custom service that meets the following requirements:

- Is based on the health cover package type
- Looks up the services that the claimant is entitled to in an XML data file
- Uses the built-in assess capabilities of determinations engine to determine the entitlement details based on the rules in the *OutpatientEligibility* rulebase.

There are two basic actions that are exposed by the custom service:

1. Return a list of available services for the claimant's card type.
2. Check if the claimant is eligible and entitled to the requested outpatient service.

How to construct this example (Java)

The `examples\custom-service` directory in *Oracle_Policy_Automation_Runtime_Java_10.2.1.zip* file contains all of the files required to set up this example.

Option 1 – Deploy a fully working example

The `examples\custom-service\webapp` directory contains the *determinations-server.war* file. This is a fully set-up instance of Determinations Server, with the custom service already installed.

1. Deploy this web application to Tomcat by copying the *determinations-server.war* file into the "webapps" directory of the Tomcat installation; for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps`.
2. Test the service by following the instructions in the section below, [Test the custom service](#).

Option 2 – Build an example from source files

Service code

The `examples\custom-service\source\Java` directory contains the Java source code for the service (the .NET source code is very similar). The service has the following two classes:

1. **OutpatientEligibilityService** – this class implements the **DSServicePlugin** interface and contains the custom service logic
2. **CustomWSDL** – this builds the WSDL for the custom service.

To build and deploy the service code, do the following:

1. Build the Java files found in `examples\custom-service\source\Java` into a JAR file, using any Java development environment. You will need to tell your Java compiler how to find the required Oracle Policy Automation dependencies, as

described in the [Create a Plugin](#) topic.

2. Name the resulting JAR file, *custom-service.jar*
3. Copy the *custom-service.jar* file you have created to the `WEB-INF\classes\plugins` directory of your determinations-server web application; for example, `C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\determinations-server\WEB-INF\classes\plugins`.

XML data file

The `examples\custom-service\source\Java` directory contains the XML data file *health-insurance-data.xml* which contains the reference data loaded by the Java code above.

It is suggested this file be placed in the `determinations-server\WEB-INF\classes\plugins` directory. Note that as the XML file location is not strictly mandated by Oracle Policy Automation, it can actually be placed anywhere on the file system, but if you do choose to locate the file in a path other than that recommended here, then that path will need to be specified in `XML_FILE_PATH` of the *OutpatientEligibilityService.java* file.

Copy the *health-insurance-data.xml* file from `examples\custom-service\source\Java` to `determinations-server\WEB-INF\classes\plugins`.

wsdl

The `examples\custom-service\source\wsdl` directory contains the WSDL templates that expose the custom service operations. Do the following:

1. Create a new directory "custom" in `determinations-server\WEB-INF\classes\templates\wsdl`
2. Copy the files from `examples\custom-service\source\wsdl` to `determinations-server\WEB-INF\classes\templates\wsdl\custom`

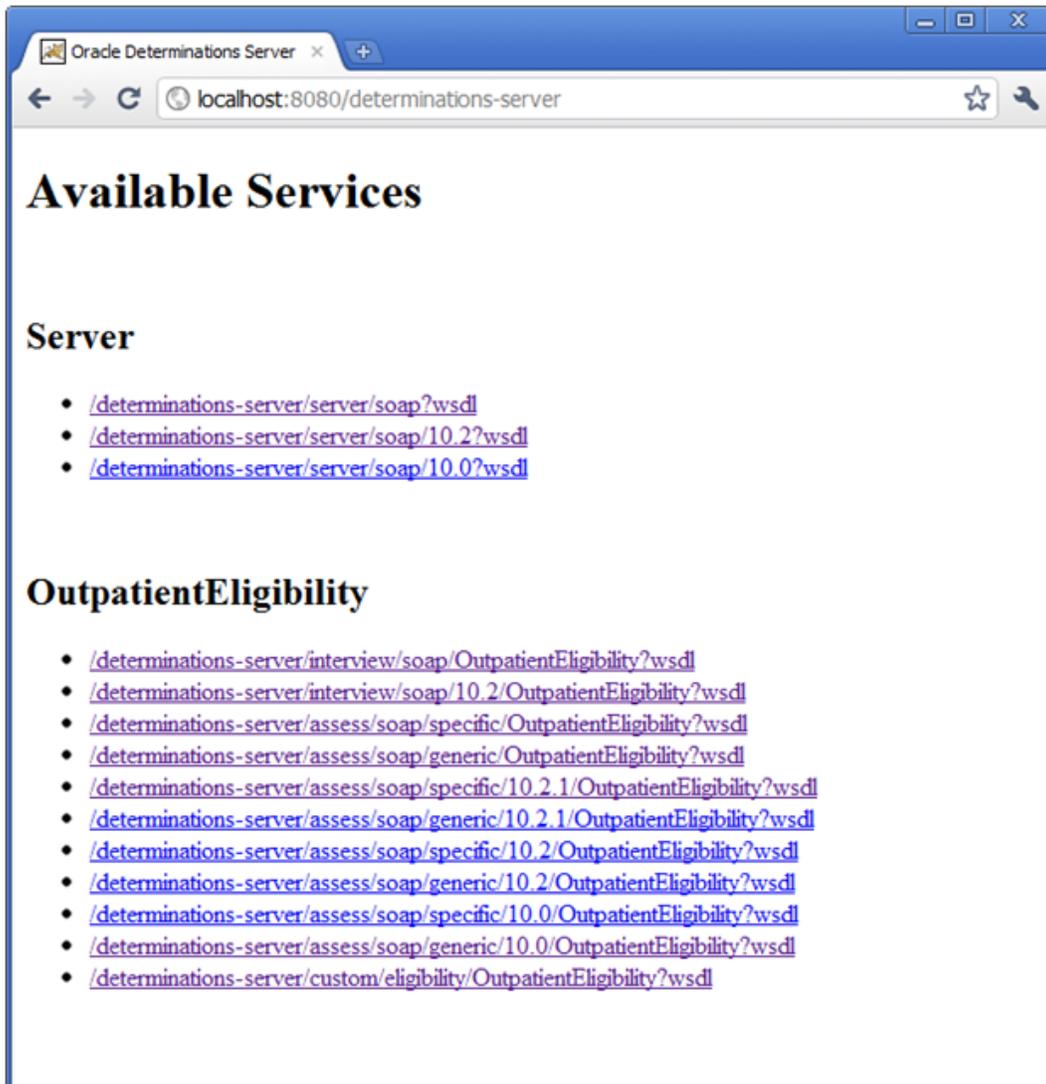
rulebase

The `examples\custom-service\rulebase` directory contains the rulebase which implements reasoning for outpatient eligibility. Do the following:

1. Build the rulebase in Oracle Policy Modeling.
2. Locate the built rulebase in `output\OutpatientEligibility.zip`.
3. Copy the file *OutpatientEligibility.zip* file into `determinations-server\WEB-INF\classes\rulebases`

Test the custom service

To test that Determinations Server has been successfully installed, access the "determinations-server" web application from a browser; for example, <http://localhost:8080/determinations-server>. You should see the following links:



To examine the custom WSDL, click the last link labelled </determinations-server/custom/eligibility/OutpatientEligibility?wsdl>. To test the new custom service, you can use a tool such as SoapUI; sample SOAP requests and expected responses can be found in the [examples\custom-service\sample-requests-responses](#).

Example - Dynamically display an error message for a control

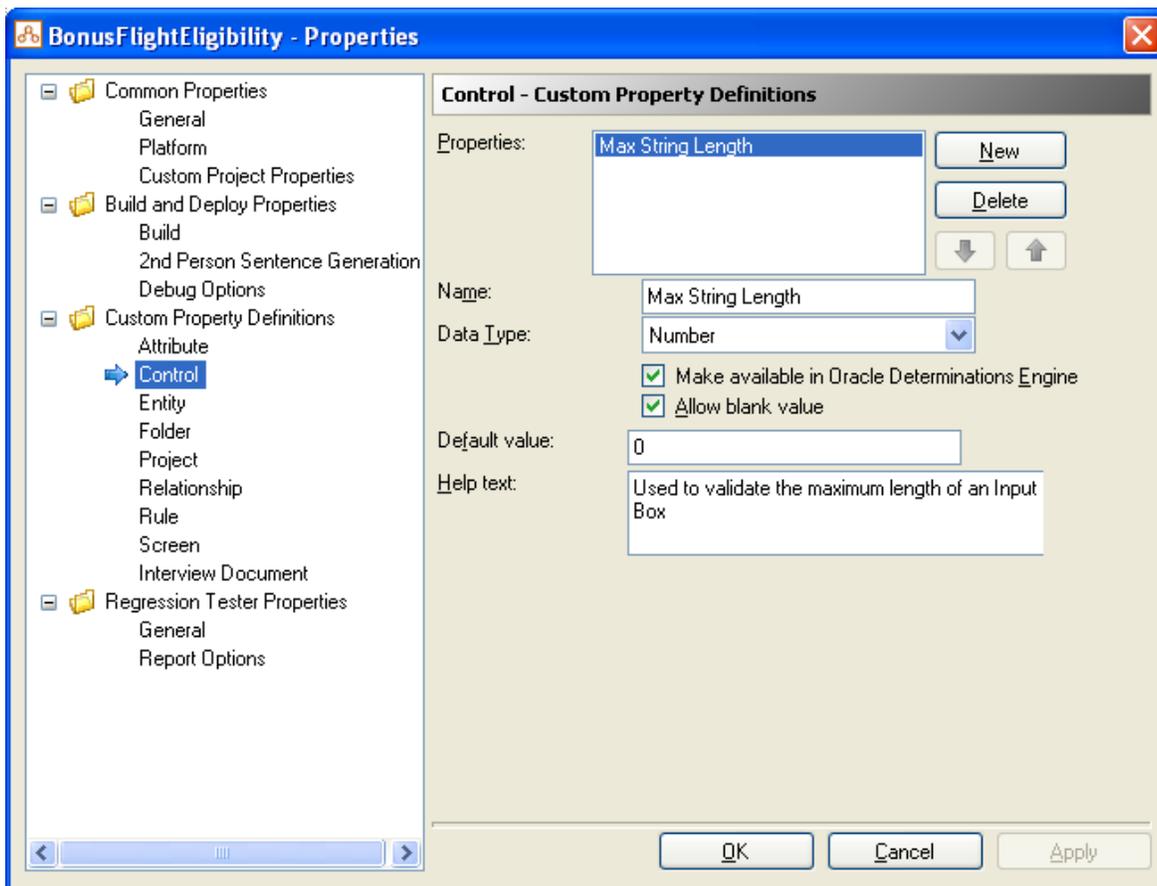
This example demonstrates how a control value entered by the user, can be dynamically validated and allow its error message to be displayed on the screen.

Construct the example:

Modify the rulebase example

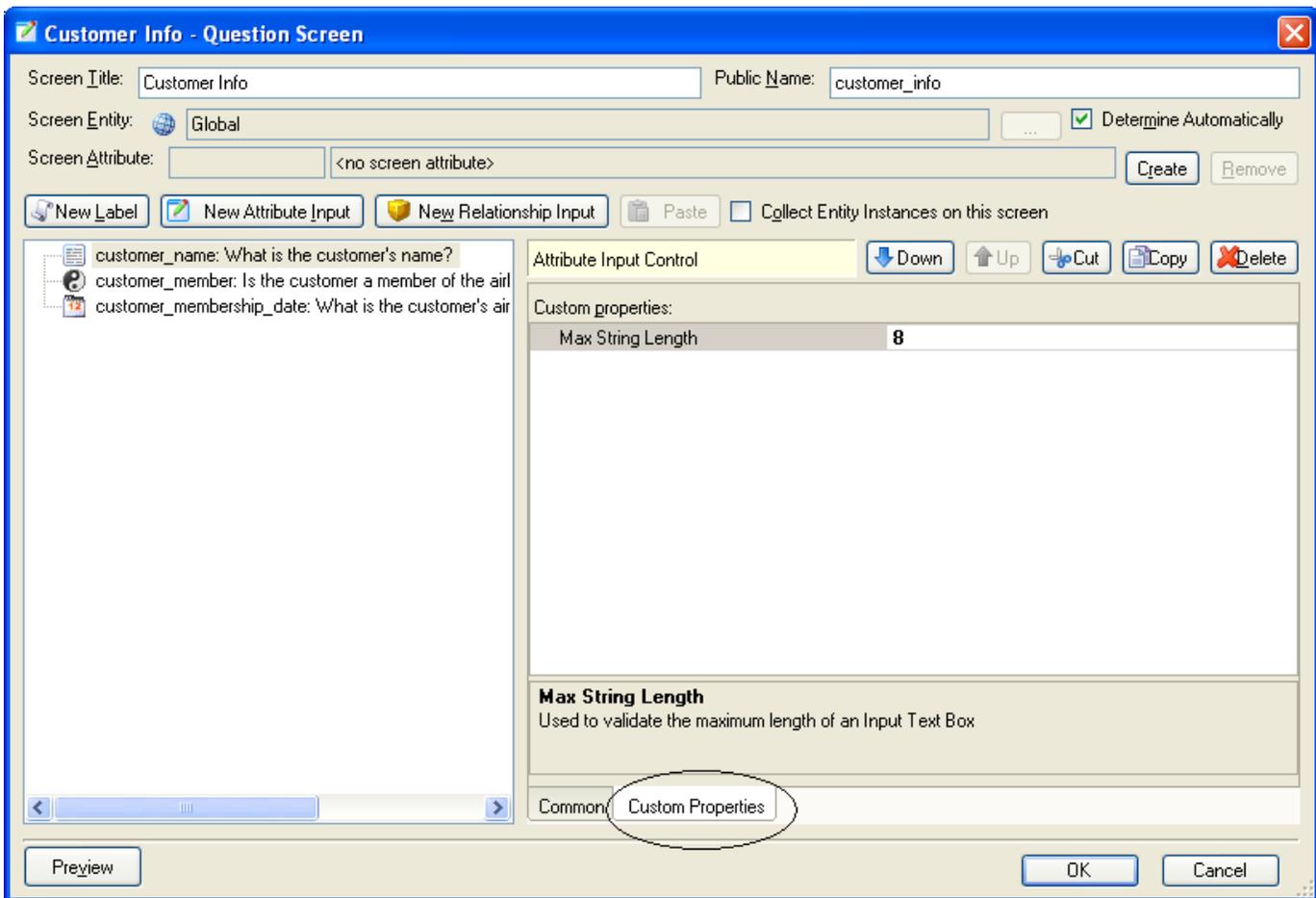
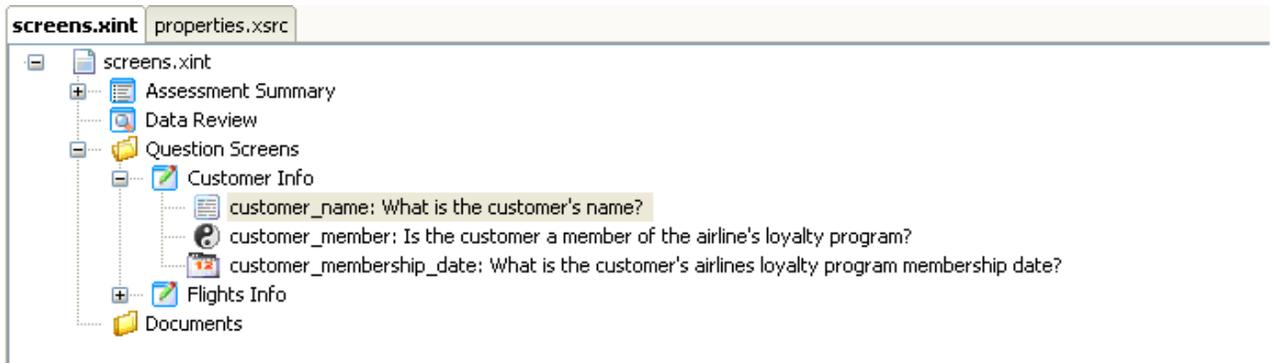
First, we need to modify the *BonusFlightEligibility* rulebase. We will define and set a custom property to allow a text input control to be validate for maximum length.

1. In Oracle Policy Modeling, open the *BonusFlightEligibility* rulebase project located in `<RUNTIME_ROOT_DIRECTORY>\examples\rulebases\source\BonusFlightEligibility`.
2. Add new *Number Control Custom Property*; Go to **File -> Project Properties -> Custom Property Definitions -> Control**.



3. As seen in the screenshot above, the "Max String Length" custom property has a default value of "0". In this example, we will validate input string length of the "the customer's name" against what is set in this property. To set the control property, do the following:

- i. Open the screens file and click on the *customer_name: What is the customer's name?* input control.
- ii. Once inside the *Question Screen* editor, click on *Custom Properties* tab and set the value of the custom property.



4. On the same view, go back to the *Common* tab and set the *Read-Only* attribute of the control to editable.
5. That's all the rulebase modifications we need. We are now ready to customize the Velocity templates.

Configure the error message

We will configure the error message in such a way that can be localized. To do this, we will set the error message inside the *messages.<locale>.properties* files. Since the rulebase is just using the English locale, we only need to add the error message in `<OWD deploy dir>\WEB-INF\classes\configuration\messages.en.properties`.

Had the rulebase been using several locales, we could add the translated error messages to each locale-specific *messages.<locale>.properties* file - this would enable the localized message to be displayed when the interview is associated to a specific locale.

```
#Dynamic validation error
InputTextMaxLengthMessage    =Character length of the input should be less than or
equal to
```

Modify the Velocity templates

The next task is to modify some Oracle Web Determinations templates. The goal of this task is to retrieve the control custom property value and subsequently process it to dynamically validate the "the customer's name" input control value. It is recommended you be familiar with the question screen, form and text input control templates in order to have a good understanding of the modifications that will be discussed below.

Information on Oracle Web Determinations templates can be found in the [Oracle Web Determinations Template Reference Guide](#).

A question screen, is usually composed of several sub-templates, but in this example, we will just focus on the three templates shown in the illustration below; *question_screen.vm*, *form.vm* and *TextInputControl.vm*. We will modify each of these to achieve our desired customization.

The screenshot shows the Oracle Web Determinations interface. At the top, there is a header with the Oracle logo and the text "Web Determinations". Below the header, there is a navigation bar with tabs for "Summary" and "Data Review". The current tab is "question_screen.vm". To the right of the tabs, there are buttons for "Save", "Save As", "Load", "Restart", and "Close". Below the navigation bar, there is a sub-header for "Customer Info" and "investigation\form.vm". The main content area is a form with a text input field labeled "What is the customer's name?" and a radio button labeled "Is the customer a member of the airline's loyalty program?". The radio button has two options: "Yes" and "No". Below the form, there is a "Submit" button. The interface is styled with a light blue background and a white border.

`<OWD deploy dir>\WEB-INF\classes\templates\question_screen.vm`

This template contains the `*<head>*` tag of the rendered HTML page. This is where we will add all of our Javascript functions:

- **validateStringLength (input, maxLength)** - this function validates the length of an input control string against a `maxLength` value passed from the control's custom property.
- **allowSubmit()** - this function is called during the **onSubmit** event of the form. It cancels submission if there are still dynamic errors on the screen.

```
<head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
```

```

    <meta http-equiv="content-language" content="{screen.getLocale()}">
    <title>{screen.getTitle()} - {header-title}</title>
    <link rel="stylesheet" type="text/css" href="{context-root-path}{re-
source-request}/reset.css" >
    <script type="text/javascript">
        ## These includes need to be present if a "form" HTML element will be
present (e.g. form.vm)
        #parse("includes/javascript-utilities.vm")
    </script>
    <style type="text/css">
    <!--
    {css-text}
    -->
    </style>
        <script type="text/javascript">
            ...
            // this function validates the input string's length against a
maxLength
            function validateStrLength (input, maxLength) {
                if (maxLength > 0){
                    if (input.value.length > maxLength){
                        document.getElementById('message_'+in-
put.id).style.visibility = 'visible';
                        input.select(true);
                        input.focus();
                    }
                    else {
                        document.getElementById('message_'+in-
put.id).style.visibility = 'hidden';
                    }
                }
            }
            // this function is called onSubmit of the form.
            // this will cancel submission of the form if there are still
"dynamic" errors
            function allowSubmit(){
                String.prototype.startsWith = function(str)
                {return (this.match("^"+str)==str)}
                var divs = document.getElementsByTagName('div');
                for (var i = 0; i < divs.length; i++) {
                    var div = divs[i];
                    if (div.id.startsWith('message_')){
                        if (div.style.visibility == "visible"){
                            return false;
                        }
                    }
                }
            }
            return true;

```

```
    }
  </script>
</head>
```

<OWD deploy dir> \WEB-INF\classes\templates\investigation\form.vm

This template contains the <form> tag of the rendered HTML page. Here, we will add an **onSubmit** function call within the <form> tag. This will suspend the submission of the form when the user clicks the **Submit** button. It will check if there are still "dynamic" errors within the screen. If errors are detected, then form submission is canceled.

```
<form name="form" accept-charset="UTF-8" method="POST" action="{post-uri}" target="{frameset-top-target}" onSubmit="return allowSubmit();">
```

<OWD deploy dir> \WEB-INF\classes\templates\controls\TextInputControl.vm

As the name of the template implies, this template is used to render text input controls of a screen. To be able to validate the text input control; first, we need to retrieve the custom property value of the control.

```
## The variable control is available in scope, and is set to the Text-
tInputInterviewControl that this template is to render
## set control custom property value
#set ($maxStringLength = ${control.getProperty("Max String Length", "0")})
```

Next, in order for us to call **validateStringLength (input, maxLength)** automatically, we will add an onBlur trigger to the control. This will enable the input control value to be validated when the user clicks out of the textbox..

```
#if(${control.getLineCount() < 2})
  <input type="text" id="{control.getEncodedID()}" onBlur=
r="validateStringLength(this, ${maxStringLength})" name="{control.getId()}"
${readOnlyString} value="{control.getDisplayValue()}" alt="{control.getText()}"
tabindex="#tabIndex()" size="{text-control-width}" ${styleAttribute} ${classAt-
tribute}>
#else
```

Lastly, we will allocate a <div> on the page that will display the error message. Notice the property `InputTextMaxLengthMessage` - this is the error message that is retrieved from the `messages.<locale>.properties`.

```
#parse( "investigation/controlMessages.vm" )
<div id="message_{control.getEncodedID()}" class="messages" style=
e="visibility:hidden">
<p class="error">${InputTextMaxLengthMessage} $maxStringLength</p>
</div>
```

We have now completed our template modifications.

Test that the error message is displayed

1. Deploy the compiled rulebase within the Web Determinations instance and restart the server.
2. Start the interview.

3. In the *Customer Info* screen, enter a customer name whose length is greater than the maximum length specified on the control custom property.
4. Click anywhere on the screen; an error should be displayed.

ORACLE Web Determinations

[Summary](#) | [Data Review](#) [Save](#) | [Save As](#) | [Load](#) | [Restart](#) | [Close](#)

Rulebase: BonusFlightEligibility Locale: en-US User ID: quest

Customer Info

Text value should only have a maximum of 8 characters.

What is the customer's name?

Is the customer a member of the airline's loyalty program? * Yes No

Notice also, when you click the **Submit** button, the form is not submitted. This is because there is still an uncorrected error on the screen.

5. Alternatively, enter a customer name whose length is equal or less than the maximum length. The error message should disappear when you click out of the text box. Furthermore, form submission can now continue.

Example: Encode the Interview Portlet's response

A sample plugin that combines the use of **OnPortletWriteResponseEvent** and **OnDecodeParamEvent** is provided in the [examples/interview-portlet/response-encoding](#) directory. This sample plugin illustrates the following:

- The post-processing of the HTML response about to be rendered. Every name attribute of an element is encoded: % symbols are replaced by a special `string _opa-perc_`.
- The pre-processing of request parameters received in in **PortletRequest.getParameterMap()**. Every parameter key is decoded: instances of `_opa-perc_` are replaced with the % symbol.

The sample [examples/interview-portlet/response-encoding](#) contains the source code, as well as the compiled *PortletResponseEncoder.jar* file. This plugin can be used as a workaround for the Liferay portal server WSRP issue (LPS-14397 <http://issues.liferay.com/browse/LPS-14397>); however, it should be noted that we do not certify the Interview Portlet on Liferay and nor do we support this sample plugin.

Example: Extract the username in the Interview Portlet

This sample illustrates the use of **OnPortletRequestEvent** to extract the user nickname property from the **USER_INFO** object provided by the portal server.

The sample code for this plugin can be found in the [examples/interview-portlet/username-extractor](#) directory.

The example consists of:

- An entry in *portlet.xml* that exposes the nickname:

```
<user-attribute>
  <description>User Nickname</description>
  <name>user.name.nickName</name>
</user-attribute>
```
- A Java class **PortletUsernameExtractor** that extracts the nickname from the Portlet Request

PortletUsernameExtractor

This class is an event handler for **OnPortletRequestEvent**. In the **handleEvent**, method, the user nickname is extracted using the following code:

```
Map userInfo = (Map)
event.getPortletRequest().getAttribute(PortletRequest.USER_INFO);
if (userInfo != null && !userInfo.isEmpty()) {
    String nickName = (String) userInfo.get("user.name.nickName");
}
```

The nickname can then be stored as an attribute in the Portlet Request using the following code:

```
event.getPortletRequest().setAttribute("user-id", nickName);
```

Extracting other attributes

The Oracle Policy Automation Interview Portlet by default exposes the nickname by way of *portlet.xml*. If other attributes need to be accessed by custom plugins, they need to be declared in *portlet.xml*.

Example: Hide eligibility criteria from a decision report depending on benefit applied for

In this example, a rulebase listener is used to install a decision report filter that can hide attributes from a decision report based on the value of another attribute. Given attribute with a name of (for example) 'foo', it looks for an attribute whose name is 'foo_show' and if foo_show has a value of false, then the attribute foo and its proof is hidden from the decision report.

rulebase-listeners.xml

This file should be created in the include folder of the rulebase project, with the following contents:

```
<listeners>
  <listener>
    <handler platform="dotnet" class="DecisionFilteringExample.ExampleRulebaseListener"/>
    <handler platform="java" class="com.oracle.determinations.example.ExampleRulebaseListener"/>
  </listener>
</listeners>
```

Java Implementation

The following code should be compiled into a JAR file and copied into the include\lib folder of the rulebase project.

```
public class ExampleRulebaseListener implements RulebaseListener {

    ExampleDecisionReportFilter m_DecisionReportFilter;

    public void initialize(Rulebase rb, Map properties, FilesContainer files) {
        m_DecisionReportFilter = new ExampleDecisionReportFilter();
    }

    public void sessionCreated(Session session) {
        session.setDecisionReportFilter(m_DecisionReportFilter);
    }
}

public class ExampleDecisionReportFilter implements DecisionReportFilter {

    public DecisionReportFilterResult getAttributeFilter(EntityInstance instance, Attribute attr) {
        Attribute filterAttr = instance.getEntity().getAttribute(attr.getName() + "_show");
        if (filterAttr != null && Boolean.FALSE.equals(filterAttr.getValue(instance))) {
            return DecisionReportFilterResult.SILENT_INVISIBLE;
        }

        return DecisionReportFilterResult.UNFILTERED;
    }

    public DecisionReportFilterResult getRelationshipFilter(EntityInstance instance, Relationship attr) {
        return DecisionReportFilterResult.UNFILTERED;
    }
}
```

C# Implementation

The following code should be compiled into an assembly DLL and copied into the include\lib folder of the rulebase project.

```
public class ExampleRulebaseListener : RulebaseListener
{
    ExampleDecisionReportFilter m_DecisionFilter;
    public void Initialize(Rulebase rb, Oracle.Determinations.Masquerade.Util.Map properties, FilesContainer files)
    {
        m_DecisionFilter = new ExampleDecisionReportFilter();
    }

    public void SessionCreated(Session session)
    {
        session.SetDecisionReportFilter(m_DecisionFilter);
    }
}

class ExampleDecisionReportFilter : DecisionReportFilter
{
    public DecisionReportFilterResult GetAttributeFilter(EntityInstance instance, RBAAttr attr)
    {
        RBAAttr filterAttr = instance.GetEntity().GetAttribute(attr.GetName() + "_show");
        if (filterAttr != null && Oracle.Determinations.Masquerade.Lang.Boolean.FALSE.Equals(filterAttr.GetValue(instance)))
        {
            return DecisionReportFilterResult.SILENT_INVISIBLE;
        }

        return DecisionReportFilterResult.UNFILTERED;
    }

    public DecisionReportFilterResult GetRelationshipFilter(EntityInstance instance, Relationship attr)
    {
        return DecisionReportFilterResult.UNFILTERED;
    }
}
```

Example: Implement Clients for the Assess Service

The following is intended to describe how to write clients that utilize the Generic and Specific Assess Service by using JAX-WS and .NET. It is assumed that the reader is reasonably proficient in web-service technology. Two samples are provided; one for generating client stub classes for JAX-WS using `wsimport` and the other for generating client stub classes using .NET.

Generating client stub classes for JAX-WS using `wsimport`

The **`wsimport`** tool reads the Assess WSDL file and generates artifacts such as the service endpoint interface (proxy object for calling operations on the web-service) and data objects that model the web-service data. The **`-d`** option specifies the target directory to store the generated client classes. The **`keep`** option ensures that the source files are kept along side the class files.

In the following example, the WSDL is the Assess Service of the example rulebase called `RuleBaseWithSimpleRelationships`.

```
wsimport -d D:/client_directory -keep http://localhost:8080/determinations-server-
/assess/soap/specific/10.2/RuleBaseWithSimpleRelationships?wsdl
```

Using the generated client stubs

The stub provides method calls that expose the available operations on the web-service (wsdl ports). In the case of the Assess Service, we have two available operations which are **`ListGoals`** and **`Assess`**.

Sample client to list goal attributes of a sample rulebase

The following example shows a simple client that invokes the **`ListGoals`** operation on the Assess Service. The request object for the Generic and Specific service is the same. This operation returns a **`ListGoal`** response object with the goal attributes for the rule base.

```
/**
 * Simple web-service client that will test the access web-service. (10.2 determinations
 * server)
 *
 * @author ramrajan
 */
public class ListGoalsClient {

    static OdsAssessServiceGeneric102RuleBaseWithSimpleRelationships assessStub;
    static OdsAssessServiceGeneric102RuleBaseWithSimpleRelationshipsType assessServiceStubType;

    public static void main(String[] args) {
        ListGoalsClient client = new ListGoalsClient();

        // set up the client stub to call operations on the web-service
        client.setUpProxyForAssessService();

        // retrieve attribute goals from the rule-base using the assess service
        client.retrieveHighLevelGoals();
    }
}
```

```

public void setUpProxyForAssessService() {
    assessStub = new OdsAssessServiceGeneric102RuleBaseWithSimpleRelationships();

    // initialise the stub
    assessServiceStubType = assessStub.getOdsAssessServiceGeneric102RuleBaseWithSimpleRelationshipsSOAP();
}

/**
 * Retrieves high level goals from the assess service
 */
public void retrieveHighLevelGoals() {

    // create a list goals request object
    ListGoalsRequest listGoalsRequest = new ListGoalsRequest();

    // invoke the List Goals operation on the assess service which returns a response object
    ListGoalsResponse listGoalsResponse = assessServiceStubType.listGoals(listGoalsRequest);

    List<ListGoalsEntityType> listGoalsEntityType = listGoalsResponse.getEntity();

    for (ListGoalsEntityType listGoal : listGoalsEntityType) {

        System.out.println("Entity Id for goal [" + listGoal.getEntityId() + "]");
        List<ListGoalsAttributeType> attributesOfGoals = listGoal.getAttribute();

        // scan through attributes
        System.out.println("Listing attributes of goal ...");

        for (ListGoalsAttributeType goalAttribute: attributesOfGoals) {
            System.out.println(" ID [" + goalAttribute.getId() + "]");
            System.out.println(" Text [" + goalAttribute.getText() + "]");
            System.out.println(" Type [" + goalAttribute.getType().toString() + "]");
        }
    }
}
}
}

```

In the example above, `assessStub.getOdsAssessServiceGeneric102RuleBaseWithSimpleRelationshipsSOAP()` calls the **getPort** method which respectively returns a static stub which we can use to call operations on the Assess Service:

```

/**
 *
 * @return
 * returns OdsAssessServiceSpecific102RuleBaseWithSimpleRelationshipsType
 */
@WebEndpoint(name = "odsAssessServiceSpecific102_RuleBaseWithSimpleRelationships_SOAP")

```

```

public OdsAssessServiceSpecific102RuleBaseWithSimpleRelationshipsType getOdsAssessServiceSpecific102RuleBaseWithSimpleRelationshipsSOAP() {
    return super.getPort(new QName("http://oracle.com/determinations/server/10.2/RuleBaseWithSimpleRelationships/assess/types",
        "odsAssessServiceSpecific102_RuleBaseWithSimpleRelationships_SOAP"), OdsAssessServiceSpecific102RuleBaseWithSimpleRelationshipsType.class);
}

```

Sample client that performs an assessment using the Specific service.

In the example client, the assess request data object and other data objects which are consumed by the **Assess** operation have already been generated by the **wsimport** tool. Refer to the assess request elements ([note:link here](#)), the request has an optional configuration element and a mandatory global instance element. The example below has the following rule in its rulebase:

```

* The teacher is happy if
*   Exists(the children, the child is happy)

```

Now, let's create a simple assess request to infer if the teacher is happy based on one of the child instances being happy. Please refer to the example code and the notes below.

1. Firstly, a new global instance needs to be created as it's compulsory element.
2. Attach the attribute that we are inferring to the global instance. Set its outcome style to decision report to indicate that we require a decision report in the response object (to provide full reasoning of why the teacher is happy or is not happy).

```

GlobalInstanceType globalInstanceType = new GlobalInstanceType();
globalInstanceType.setTeacherHappy(new BooleanAttributeType());
globalInstanceType.getTeacherHappy().setOutcomeStyle(OutcomeStyleEnum.DECISION_REPORT);

```

3. Create a child instance with a child is happy attribute, and attach it to the entity-child-list.
4. Invoking the Assess method call with the assess object should return a response object from the assess web-service.

Assess 10.2 Specific Client

```

package com.oracle.determinations.client;

import java.math.BigDecimal;
import java.util.GregorianCalendar;

import javax.xml.datatype.DatatypeConfigurationException;
import javax.xml.datatype.DatatypeFactory;
import javax.xml.datatype.XMLGregorianCalendar;

import com.oracle.determinations.server._10_2.rulebasewithsimplerelationships.assess.types.AssessRequest;
import com.oracle.determinations.server._10_2.rulebasewithsimplerelationships.assess.types.AssessResponse;
import com.oracle.determinations.server._10_2.rulebasewithsimplerelationships.assess.types.AttributeTypeEnum;
import com.oracle.determinations.server._10_2.rulebasewithsimplerelationships.assess.types.BooleanAttributeType;

```

```

import com.oracle.determinations.server._10_2.rulebasewithsimplerelationships.assess.types.DateAttributeType;
import com.oracle.determinations.server._10_2.rulebasewithsimplerelationships.assess.types.GlobalInstanceType;
import com.oracle.determinations.server._10_2.rulebasewithsimplerelationships.assess.types.NumberAttributeType;
import com.oracle.determinations.server._10_2.rule-
basewithsimplerelationships.assess.types.OdsAssessServiceSpecific102RuleBaseWithSimpleRelationships;
import com.oracle.determinations.server._10_2.rule-
basewithsimplerelationships.assess.types.OdsAssessServiceSpecific102RuleBaseWithSimpleRelationshipsType;
import com.oracle.determinations.server._10_2.rulebasewithsimplerelationships.assess.types.OutcomeStyleEnum;
import com.oracle.determinations.server._10_2.rulebasewithsimplerelationships.assess.types.TheChildEntityListType;
import com.oracle.determinations.server._10_2.rulebasewithsimplerelationships.assess.types.TheChildInstanceType;

/**
 * This client aims to test the Assess Web-Service layer. The main intention is to use a
 * simple rule-base which has an global attribute and the the outcome is determined by
 * the instances of entity-level attributes.
 *
 * <code>
 * The teacher is happy if
 * Exists(the children, the child is happy)
 * </code>
 *
 * @author ramrajan
 */
public class AssessClient {

    private static String CHILD_ENTITY_ID = "the_child";
    private static String CHILD_ENTITY_LEVEL_ATTR = "child_is_happy";

    static OdsAssessServiceSpecific102RuleBaseWithSimpleRelationships assessStub;
    static OdsAssessServiceSpecific102RuleBaseWithSimpleRelationshipsType assessServiceStubType;

    public static void main(String[] args) {
        AssessClient client = new AssessClient();
        client.setUpProxyForAssessService();
        client.testEntityRelationship();
        client.testDateAndCurrencyDataTypes();
    }

    public void setUpProxyForAssessService() {
        assessStub = new OdsAssessServiceSpecific102RuleBaseWithSimpleRelationships();
        assessServiceStubType = assessStub.getOdsAssessServiceSpecific102RuleBaseWithSimpleRelationshipsSOAP();
    }

    /**
     * Create instances of entities with their attributes set.

```

```

*
* @return
*/
public void testEntityRelationship() {
    System.out.println("Create Assess Request object.");
    AssessRequest assessRequest = new AssessRequest();

    // Everything is contained within a global instance.
    GlobalInstanceType globalInstanceType = new GlobalInstanceType();
    globalInstanceType.setTeacherHappy(new BooleanAttributeType());
    globalInstanceType.getTeacherHappy().setOutcomeStyle(OutcomeStyleEnum.DECISION_REPORT);

    globalInstanceType.setListTheChild(new TheChildEntityListType());

    // Child instance - mark
    TheChildInstanceType childInstance1 = new TheChildInstanceType();
    childInstance1.setId("mark");

    BooleanAttributeType markHappyAttribute = new BooleanAttributeType();
    markHappyAttribute.setType(AttributeTypeEnum.BOOLEAN);
    markHappyAttribute.setBooleanVal(true);
    childInstance1.setChildIsHappy(markHappyAttribute);

    // Create an entity instance for child
    globalInstanceType.getListTheChild().getTheChild().add(childInstance1);
    assessRequest.setGlobalInstance(globalInstanceType);

    AssessResponse assessResponse = assessServiceStubType
        .assess(assessRequest);
    GlobalInstanceType globalInstanceResponse = assessResponse
        .getGlobalInstance();
    System.out
        .println("Infered value for teacher being happy ["
            + globalInstanceResponse.getTeacherHappy()
                .isBooleanVal() + "]);
}

/**
 * Test currency and date attributes.
 * @return
 */
public void testDateAndCurrencyDataTypes() {

    AssessRequest assessRequest = new AssessRequest();

    // Everything is contained within a global instance.

```

```

GlobalInstanceType globalInstanceType = new GlobalInstanceType();

globalInstanceType.setPersonCompensation(new BooleanAttributeType());
globalInstanceType.getPersonCompensation().setOutcomeStyle(OutcomeStyleEnum.DECISION_REPORT);

globalInstanceType.setPersonIncome(new NumberAttributeType());
globalInstanceType.getPersonIncome().setNumberVal(new BigDecimal("25000.00"));

GregorianCalendar calendar = new GregorianCalendar();
calendar.set(1999, 01, 01);
XMLGregorianCalendar xgcal = null;
try {
    xgcal = DatatypeFactory.newInstance().newXMLGregorianCalendar(calendar);
} catch (DatatypeConfigurationException dce) {
    dce.printStackTrace();
}

globalInstanceType.setPersonDob(new DateAttributeType());
globalInstanceType.getPersonDob().setDateVal(xgcal);
assessRequest.setGlobalInstance(globalInstanceType);

AssessResponse assessResponse = assessServiceStubType
    .assess(assessRequest);
System.out.println("Is person eligible for compensation ["
    + assessResponse.getGlobalInstance().getPersonCompensation()
    .isBooleanVal() + "]);
}
}

```

Generating client stub classes using .NET

It's just a simple matter of point and click to generate a web-service assess client using Visual Studio. To generate the classes, go to **Project -> Add Service Reference**, paste the WSDL url in the **Address field** and click on **Go**. Then when you have located the right service, type in a namespace and click on **Ok**.

A simple example in .NET

```

namespace WSAssessClientSpecific10_2
{
    class AssessClient
    {
        static odsAssessServiceSpecific102_RuleBaseWithSimpleRelationships_typeClient proxy;

        static void Main(string[] args)
        {
            Console.WriteLine("Assess client specific for 10.2");
            AssessClient client = new AssessClient();
            proxy = new odsAssessServiceSpecific102_RuleBaseWithSimpleRelationships_typeClient();
        }
    }
}

```

```

        client.getHighLevelGoals();
        client.identifyTeacherIsHappy();
        client.identifyPersonEligibleForCompensation();
    }

    public void getHighLevelGoals()
    {
        listgoalsrequest request = new listgoalsrequest();
        ListGoalsEntityType[] goalsList = proxy.ListGoals(request);
        foreach (ListGoalsEntityType goalEntity in goalsList)
        {
            ListGoalsAttributeType[] attributeList = goalEntity.attribute;
            foreach (ListGoalsAttributeType attribute in attributeList)
            {
                Console.WriteLine("ID :[" + attribute.id + "] Text: [" + attribute.text + "]");
            }
        }
    }

    public void identifyTeacherIsHappy()
    {
        AssessRequest assessRequest = new AssessRequest();
        GlobalInstanceType globalInstanceType = new GlobalInstanceType();
        globalInstanceType.teacher_happy = new booleanAttributeType();
        globalInstanceType.teacher_happy.outcomestyle = OutcomeStyleEnum.decisionreport;
        globalInstanceType.teacher_happy.outcomestyleSpecified = true;

        globalInstanceType.listthe_child = new the_childEntityListType();
        globalInstanceType.listthe_child.the_child = new the_childInstanceType[1];

        the_childInstanceType childInstance = new the_childInstanceType();
        childInstance.id = "mark";
        booleanAttributeType booleanAttribute = new booleanAttributeType();
        booleanAttribute.Item = true;
        childInstance.child_is_happy = booleanAttribute;

        globalInstanceType.listthe_child.the_child.SetValue(childInstance, 0);

        assessRequest.globalinstance = globalInstanceType;
        AssessResponse assessResponse = proxy.Assess(assessRequest);
        Console.WriteLine("Is the teacher happy [" + assessResponse.globalinstance.teacher_happy.Item + "]");
    }

    public void identifyPersonEligibleForCompensation()
    {
        AssessRequest assessRequest = new AssessRequest();

```

```
GlobalInstanceType globalInstanceType = new GlobalInstanceType();
globalInstanceType.person_compensation = new booleanAttributeType();
globalInstanceType.person_compensation.outcomestyle = OutcomeStyleEnum.decisionreport;
globalInstanceType.person_compensation.outcomestyleSpecified = true;

globalInstanceType.person_income = new numberAttributeType();
globalInstanceType.person_income.type = AttributeTypeEnum.currency;
globalInstanceType.person_income.Item = new Decimal(1500000.00);

assessRequest.globalinstance = globalInstanceType;
AssessResponse assessResponse = proxy.Assess(assessRequest);
Console.WriteLine("Is the person eligible for compensation [ " + assessResponse.globalinstance.person_com-
pensation.Item + " ]");
    }
}
}
```

Example: Implement Clients for the Interview Service

The following contains notes and tips for creating an Interview Service client in .NET.

Sample Client Code

To add and remove Web References (2.0 Framework), refer to the information found at the following link:

[http://msdn.microsoft.com/en-us/library/d9w023sx\(v=VS.90\).aspx](http://msdn.microsoft.com/en-us/library/d9w023sx(v=VS.90).aspx)

The sample client's target framework is 2.0.

To add and remove Service References (3.0 Framework and later), refer to the following tutorial in this Oracle Policy Automation Developer's Guide:

[Tutorial: Create and use a C# client for Oracle Determinations Server.](#)

Interview Service Operations

For a description of the available operations, refer to the topic [Interview Service](#).

Open Session: Adding locale information in the Soap header

When creating an interview session, the default rulebase language is used if no language information is specified in the soap header.

To insert an element in the SoapHeader, one can achieve this using:

- a. SoapHeader
- b. SoapExtension, or
- c. a combination of both.

Of the three, the simplest approach is to create a class that extends SoapHeader (option b). Note that the determinations-server is expecting a <locale> element, so the root element name should be specified accordingly; otherwise, the class name will be used as the element name during serialization. In this case, the element will be ignored and the default rulebase language is used.

SoapHeaderExtension.cs

```
[System.Xml.Serialization.XmlRoot("international", Namespace = "http://www.w3.org/2005/09/ws-i18n")]
public class ODSSoapHeader : SoapHeader
{
    public String locale;
    public String tz;
}
```

Modify the proxy code to add ODSSoapHeader as a public member.

```
public partial class odsInterviewService_InterviewServiceTest : System.Web.Services.Protocols.SoapHttpClientProtocol {
    public ODSSoapHeader customHeader;
```

Apply a SoapHeader attribute on the relevant client method. In this case, OpenSession().

```
[System.Web.Services.Protocols.SoapHeader("customHeader")]
public opensessionresponse OpenSession([System.Xml.Serialization.XmlElementAttribute("open-session-request",
```

```

    Namespace="http://oracle.com/determinations/server/interview/InterviewServiceTest/types")] opessionrequest
opessionrequest) {
    object[] results = this.Invoke("OpenSession", new object[] {
        opessionrequest});
    return ((opessionresponse)(results[0]));
}

```

We now specify the locale prior to invoking OpenSession operation.

```

opessionrequest request = new opessionrequest();
odsInterviewService_InterviewServiceTest proxy = new odsInterviewService_InterviewServiceTest();
if (locale != null)
{
    ODSSoapHeader mainHeader = new ODSSoapHeader();
    mainHeader.locale = locale;
    proxy.customHeader = mainHeader;
}

opessionresponse response = proxy.OpenSession(request);

```

List Cases / Save Case / Load Case

These operations require a data adaptor plugin. No default adaptor is provided in Oracle Determinations Server. A generic error will be returned when no data adaptor is deployed.

Sending Screen Data

Investigate, Get and Set Screen operations process screen data. When sending screen data, keep in mind the following:

- All screen controls should be included in the request; otherwise an error will be thrown.
- In the case of invisible controls, these should be submitted. The value should be unchanged.
- For entity collect screens:
 - Investigate will include a template entity instance control. This will be the one with "-BLANK-" id.
 - Get Screen will only return a template control if no instances were created.

Using the sample web application

The project and related files are located in: `Oracle_Policy_Automation_Runtime_DotNet_10_3_0\examples\interview-service-client`. The project, a simple version of Web Determinations, was created to test the Interview Service operations. Some features (such as handling of input styles and custom properties) are not present.

To be able to run the project, it is assumed that:

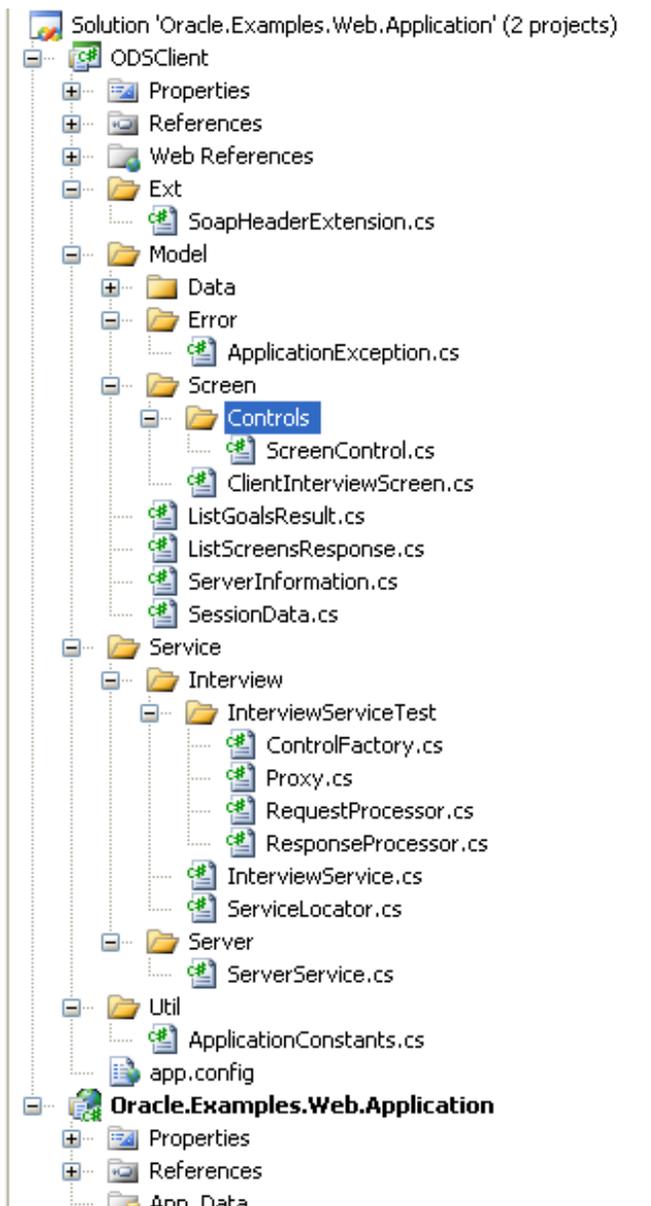
- You have Visual Studio 2008.
- ASP.NET MVC 2 is installed.
- Oracle Determinations Server (10.2 or later) is running in IIS.
- The following are deployed successfully:
 - Sample rulebase (`Oracle_Policy_Automation_Runtime_DotNet_10_3_0\examples\rulebases\InterviewServiceTest`).

- Data adaptor plugin.

The model classes are located in the ODSClient project. This is also where the Web References are defined. The sample web application can detect other rulebases deployed in ODS, but an error is thrown when attempting to create a session. To be able to handle additional rulebases, the following steps are necessary:

- Add a class that implements `ODSClient.Service.Interview.InterviewService`. This class will call the client proxy methods.
- Modify `ODSClient.Service.Interview.ServiceLocator` so that it will return an instance of that class.

[Click here to view the solution explorer.](#)

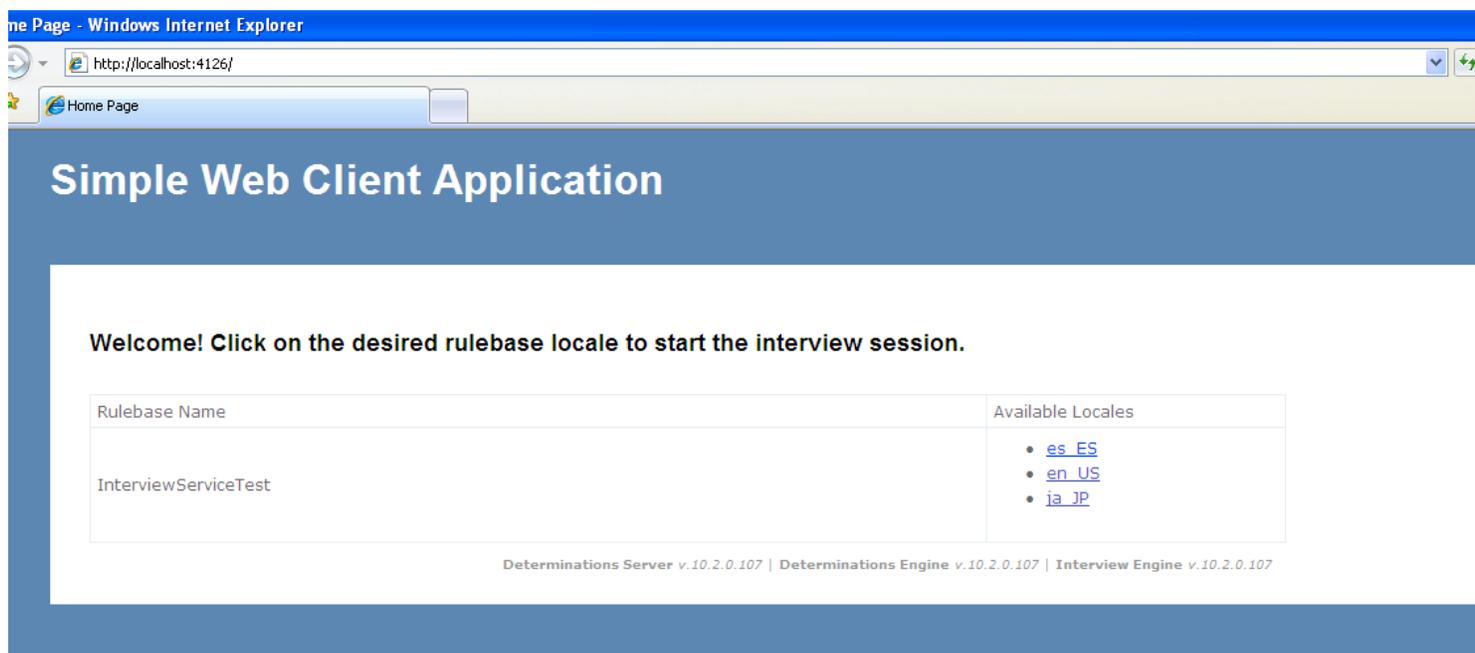


Home Page

The information displayed on the Home Page screen is the output of List Rulebases and Get Server Info (for the runtime versions displayed in the footer), both are Server Service operations.

As of release 10.2, the List Rulebases operation includes the available locales in a rulebase locale. Once the locale link is clicked, an interview session is started by calling Open Session operation. At that point, a session id will be generated.

[Click here to view the Home Page screen](#)

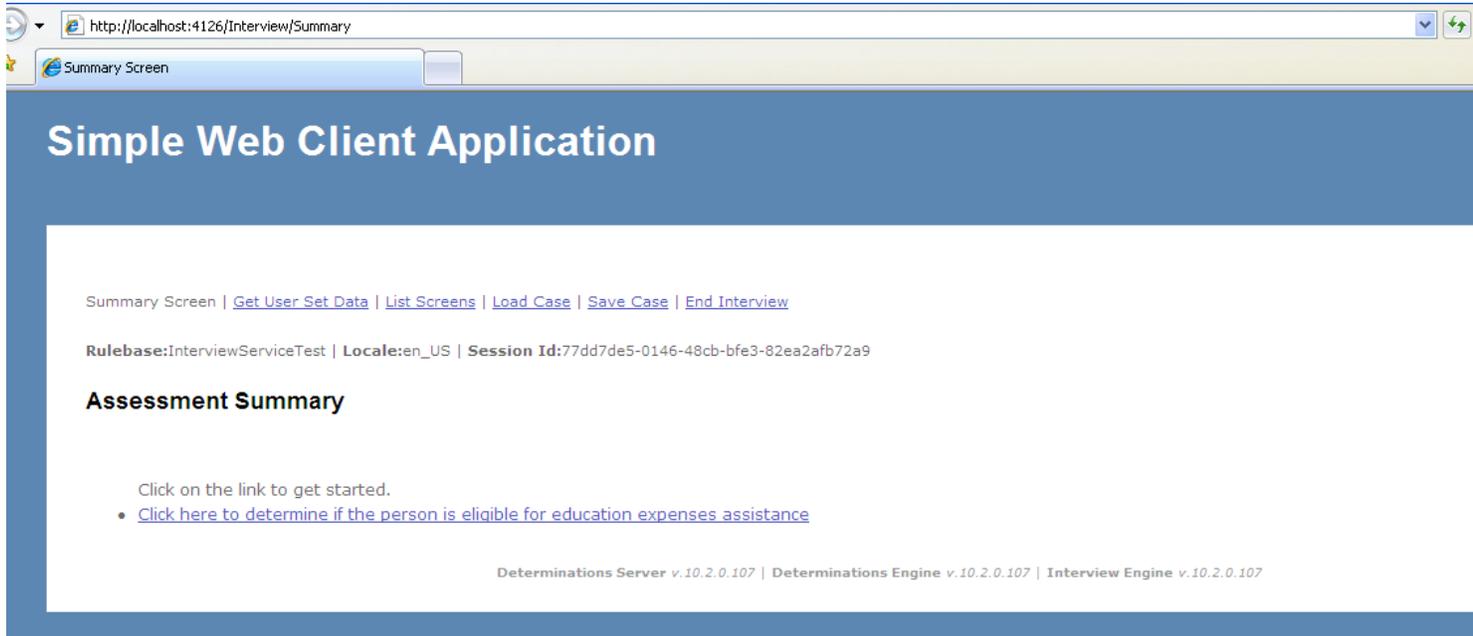


Summary Screen

If there is only one goal for the rulebase, it is possible to launch an investigation right away. For those multiple goals, a displaying a summary screen is advisable. To do this, we can:

1. Query for an available summary screen using Get Screen.
2. If none is defined, a summary screen can be built with goals returned by List Goals.
3. Once goal is selected, Investigate operation is called. The response should contain the first question screen.

[Click here to view the Summary screen](#)



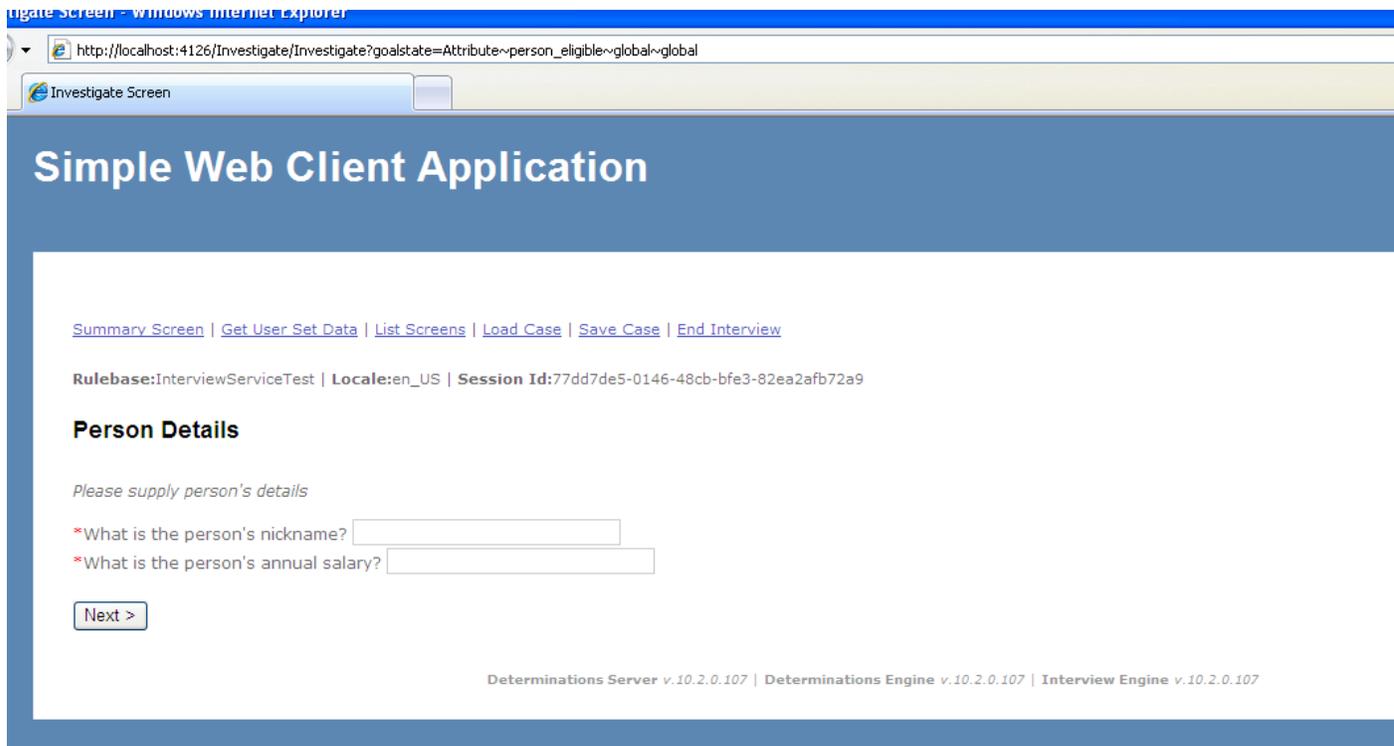
Investigate Screen

The client application simply renders the screen data included in the Determinations Server response. When form data is submitted, the screen data (with or without input value) is set in the request. If at least one of the screen controls is not declared in the request, an error is thrown. Invisible controls should also be included.

If there are errors, the response will return the current screen. Otherwise, the next screen is returned. If the goal is already known, the screen returned will be the summary screen.

Question Screen

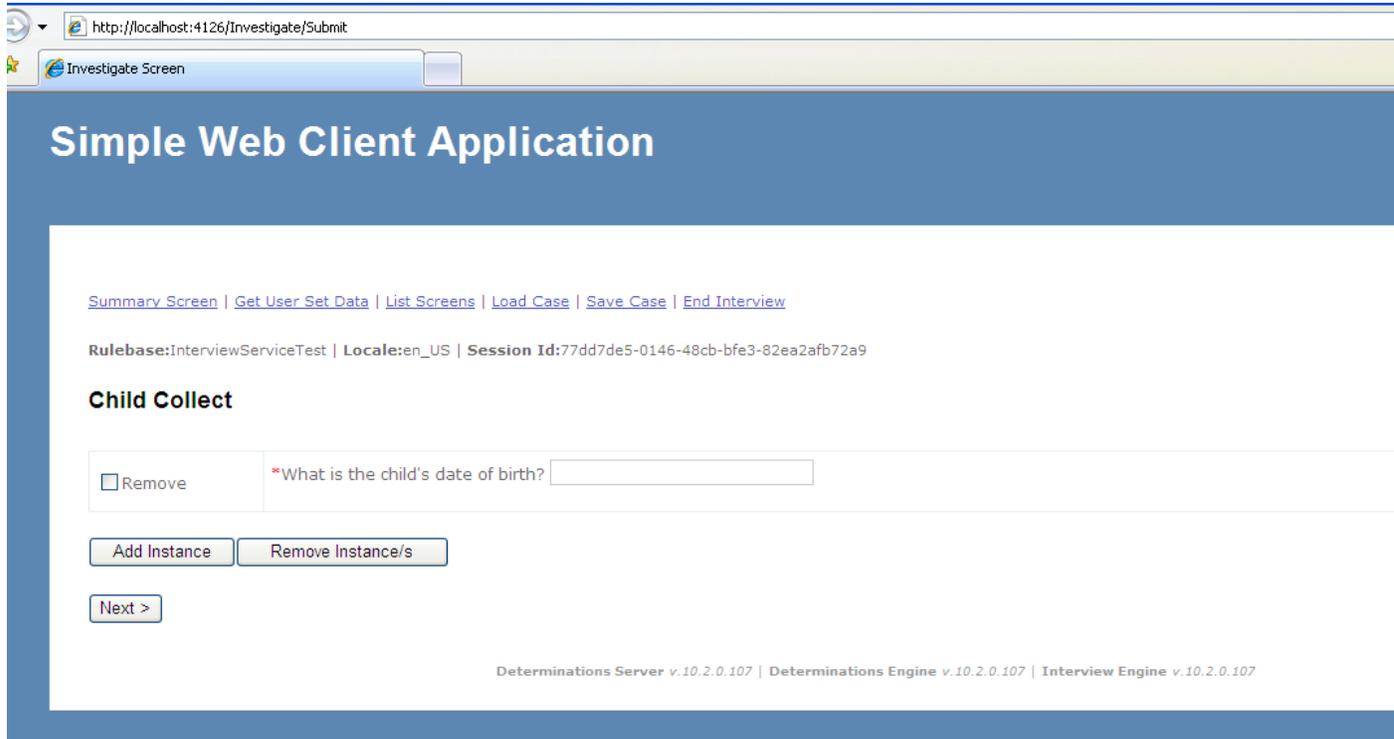
[Click here to view the Question screen](#)



Entity Collect Screen

If the screen is an Entity Collect screen, the first entity instance control can serve as a template. As with Web Determinations, the client will generate the entity instance id.

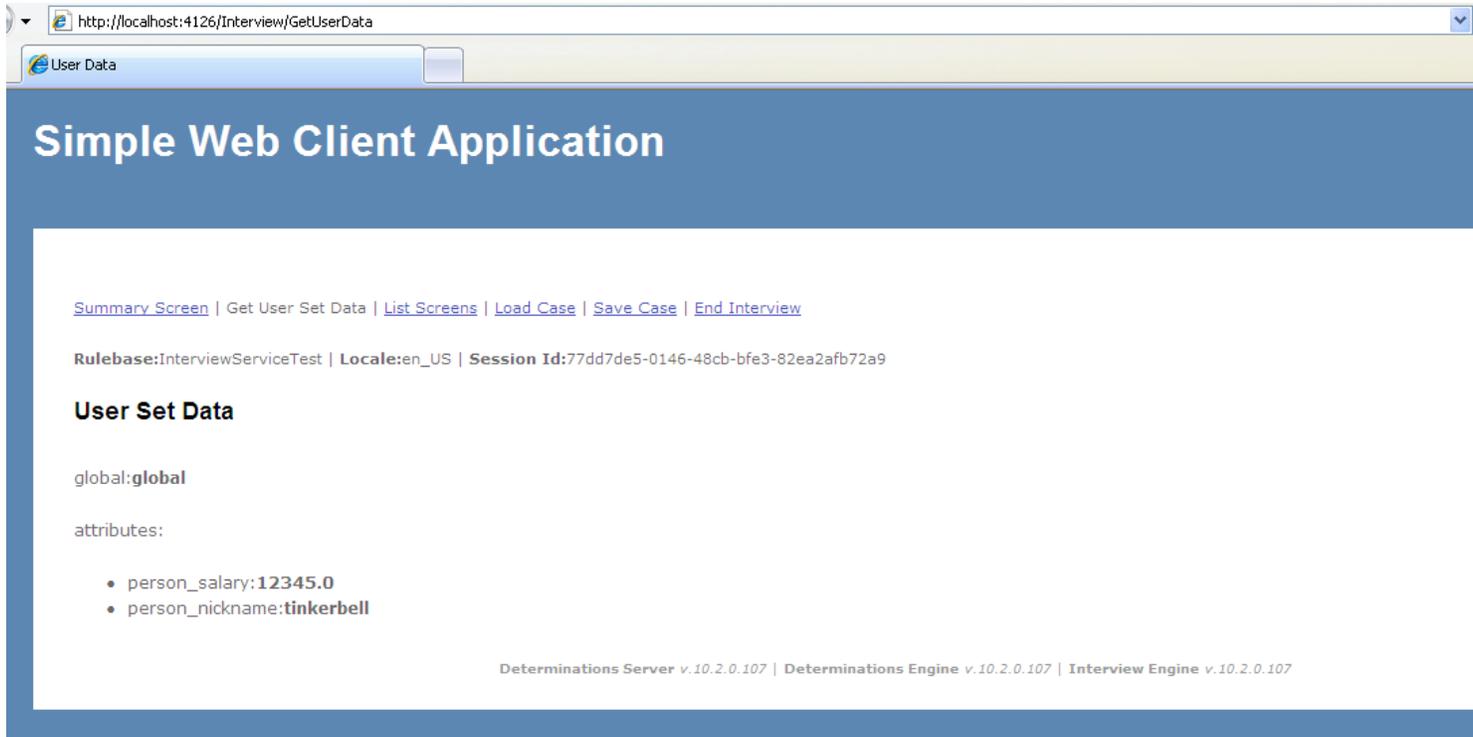
[Click here to view the Entity Collect screen](#)



Get User Set Data

The Get User Set Data page displays the current session data (Get User Set Data operation).

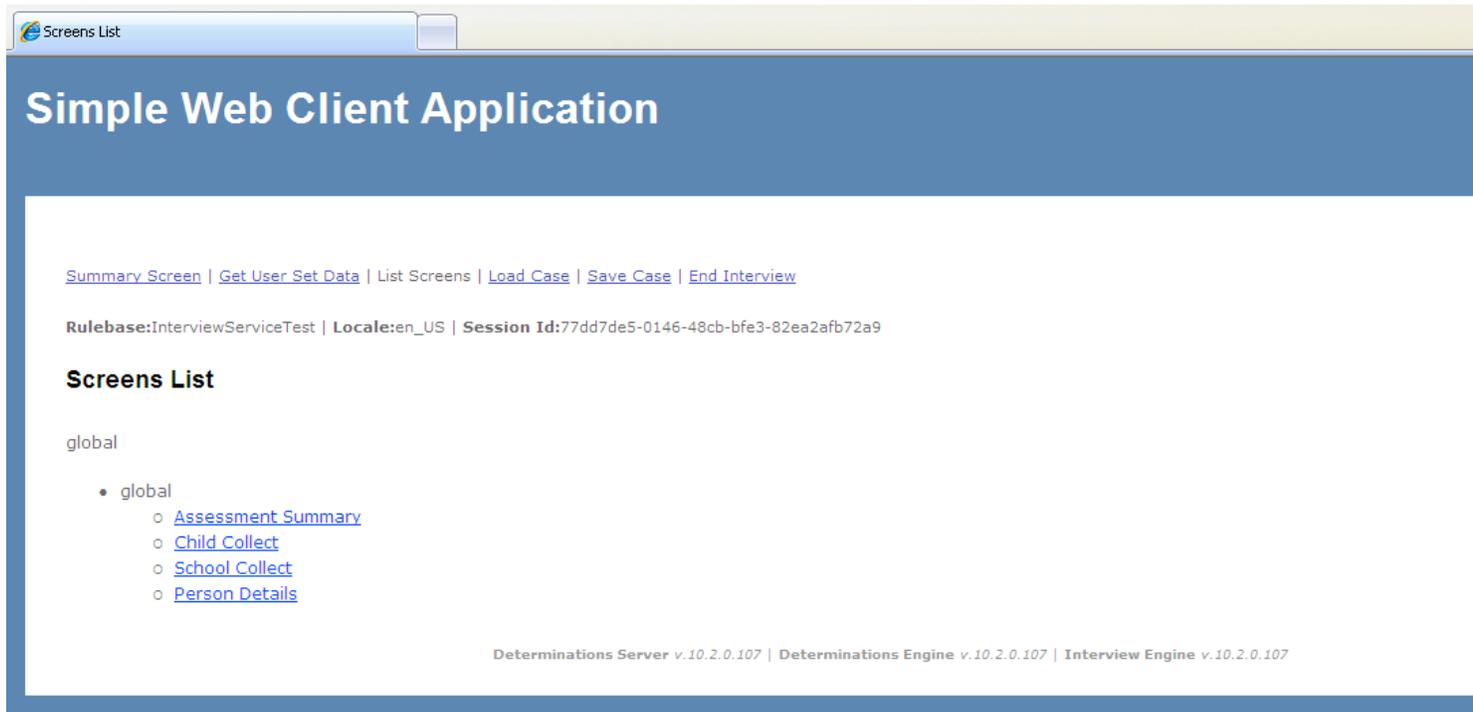
[Click here to view the Get User Set Data page](#)



List Screens

The List Screens page displays the screen instances in scope. This based on List Screens operation.

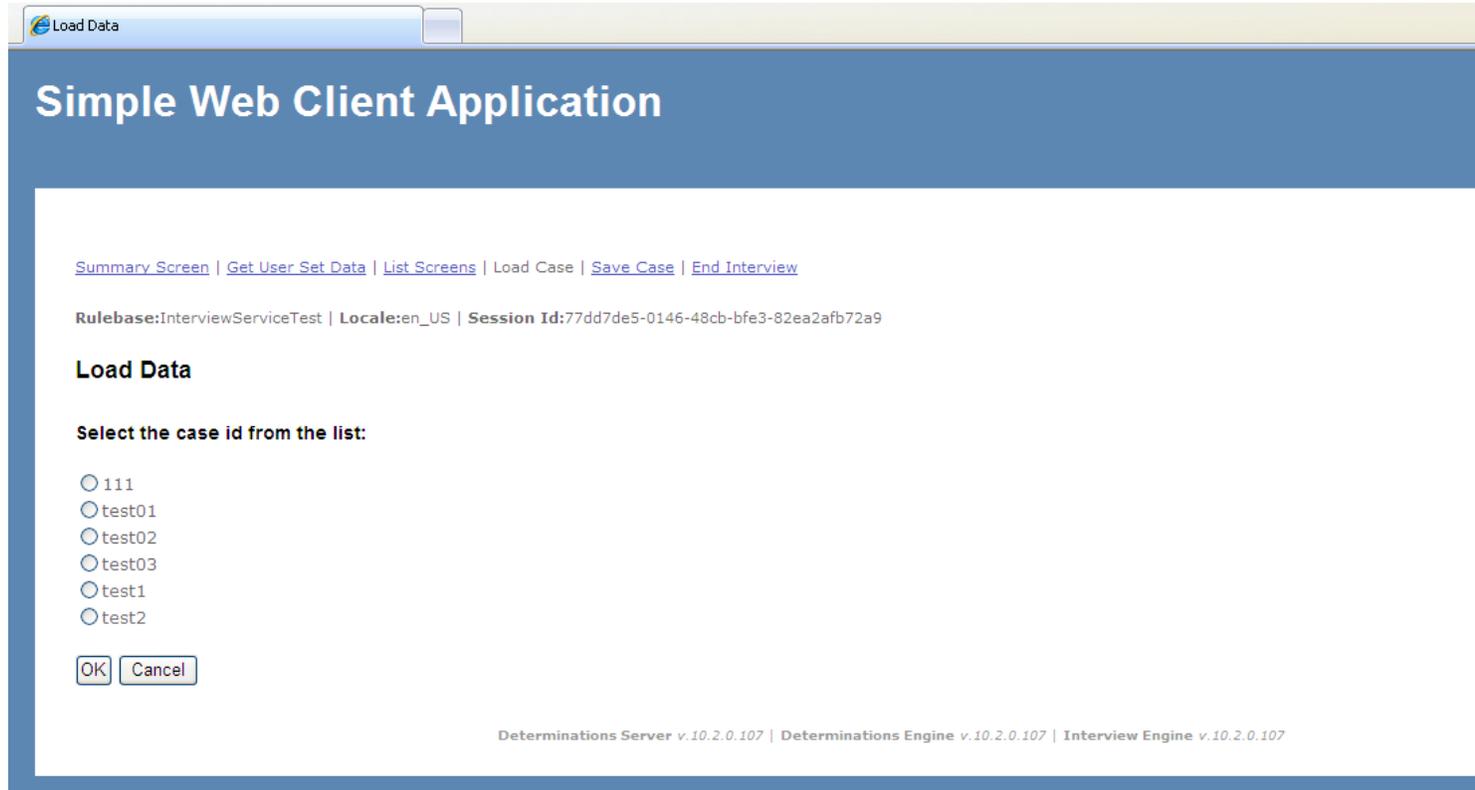
[Click here to view the List Screens page](#)



Load Case

The Load Case page initially lists the available cases (List Cases operation). If Load Case is successful, the page is redirected to the summary screen. This requires a data adaptor plugin.

[Click here to view the Load Case page](#)



The screenshot shows a web browser window with a tab titled "Load Data". The page has a blue header with the text "Simple Web Client Application". Below the header is a navigation menu with links: [Summary Screen](#), [Get User Set Data](#), [List Screens](#), [Load Case](#), [Save Case](#), and [End Interview](#). Below the navigation menu, the page displays session information: **Rulebase:** InterviewServiceTest | **Locale:** en_US | **Session Id:** 77dd7de5-0146-48cb-bfe3-82ea2afb72a9. The main content area is titled "Load Data" and contains the instruction "Select the case id from the list:". Below this instruction is a list of radio buttons with the following case IDs: 111, test01, test02, test03, test1, and test2. At the bottom of the list are two buttons: "OK" and "Cancel". The footer of the page contains the text: "Determinations Server v.10.2.0.107 | Determinations Engine v.10.2.0.107 | Interview Engine v.10.2.0.107".

Save Case

The Save Case page enables the user to save the session data. This requires a data adaptor plugin.

[Click here to view the Save Case page](#)

Simple Web Client Application

[Summary Screen](#) | [Get User Set Data](#) | [List Screens](#) | [Load Case](#) | [Save Case](#) | [End Interview](#)

Rulebase:InterviewServiceTest | **Locale:**en_US | **Session Id:**43fd792e-9507-40e1-8d4c-cf0f272367c3

Save Session Data

Save case as :

Example: Investigation using the Determinations Server Interview Service

The following is an example of an investigation using the Determinations Server Interview Service operations. The rulebase used to generate the requests and responses can be found at `examples\rulebases\compiled\InterviewServiceTest.zip` in the Oracle Policy Automation Runtime package.

Step 1 - Open a session

To begin, a session needs to be opened using the [Open Session](#) operation as follows:

Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:open-session-request/>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1100</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:open-session-response>
      <typ:success>true</typ:success>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
    </typ:open-session-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Step 2 - List all available Goals

Get a list of the available goals; it is necessary to pass in the Interview Session ID that was listed in the response from the **Open Session** call. This is done by using the [List Goals](#) operation as follows:

Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>
```

```

<soapenv:Body>
  <typ:list-goals-request>
    <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
  </typ:list-goals-request>
</soapenv:Body>
</soapenv:Envelope>

```

Response

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1100</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:list-goals-response>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:entity id="global">
        <typ:instance id="global">
          <typ:goal>
            <typ:goal-id>Attribute~person_eligible~global~global</typ:goal-id>
            <typ:goal-text>Is the person eligible for education expenses assistance?</typ:goal-text>
            <typ:has-decision-report>true</typ:has-decision-report>
            <typ:attribute-id>person_eligible</typ:attribute-id>
          </typ:goal>
        </typ:instance>
      </typ:entity>
    </typ:list-goals-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Step 3 - Commence investigation of the goal

The call to **List Goals** showed that only one top-level goal is available; now begins the investigation of this goal. As this is the beginning the investigation, there is no screen data to pass in, in the investigate request. In the request it is necessary to pass the Interview Session ID, as well as the Goal State for the goal being investigated. In this example, the goal state for the only goal that was listed in the previous call to List Goals is passed in- "*Is the person eligible for education expenses assistance?*"

This is done by using the [Investigate](#) operation as follows:

Request

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>

```

```

<soapenv:Body>
  <typ:investigate-request>
    <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
    <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
  </typ:investigate-request>
</soapenv:Body>
</soapenv:Envelope>

```

Response

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1100</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:investigate-response>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
      <typ:progress>0.0</typ:progress>
      <typ:screen id="qs$s2@Interviews_Screens_xint$global$global" name="s2@Interviews_Screens_xint" title-
e="Input person's details"
      context-entity-id="global" context-instance-id="global" type="question" is-automatic="false">
        <typ:currency-control is-visible="true" caption="What is the person's annual salary?" caption-style="" is-htm-
l="false" css-class=""
        css-style="" attribute-id="person_salary" is-mandatory="true" is-read-only="false" selection-style="">
          <typ:current-value>
            <typ:unknown-val/>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:currency-control>
      </typ:screen>
    </typ:investigate-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Step 4 - Input pertinent data relating to the subject of the investigation

The screen returned from the first **Investigate** operation contains a single input control for an attribute with type *currency*. This input is for "the person's annual salary". It is now necessary to input the value for this control - inside the *current-value* tag; an amount of \$50000 is input as follows:

Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:investigate-request>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
      <typ:screen id="qs$s2@Interviews_Screens_xint$global$global" name="s2@Interviews_Screens_xint" title-
e="Input person's details"
      context-entity-id="global" context-instance-id="global" type="question" is-automatic="false">
        <typ:currency-control is-visible="true" caption="What is the person's annual salary?" caption-style="" is-htm-
l="false" css-class=""
        css-style="" attribute-id="person_salary" is-mandatory="true" is-read-only="false" selection-style="">
          <typ:current-value>
            <typ:number-val>50000</typ:number-val>
          </typ:current-value>
          <typ:default-value>
            <typ:unknown-val/>
          </typ:default-value>
        </typ:currency-control>
      </typ:screen>
    </typ:investigate-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1100</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:investigate-response>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
      <typ:progress>0.16666666666666666</typ:progress>
      <typ:screen id="qs$s3@Interviews_Screens_xint$global$global" name="s3@Interviews_Screens_xint" title-
e="Collect the children"
      context-entity-id="global" context-instance-id="global" type="question" is-automatic="false">
        <typ:containment-relationship-control is-visible="true" caption-style="" is-html="false" css-class="" css-style=""
```

```

relationship-id="children"
  source-entity-id="global" source-instance-id="global" target-entity-id="child" relationship-type="OneToMany"
display-style="Portrait">
  <typ:entity-instance-control is-visible="true" caption-style="" is-html="false" css-class="" css-style="" entity-
id="child"
  instance-id="--BLANK--">
  <typ:date-control is-visible="true" caption="What is the child's date of birth?" caption-style="" is-html="false"
css-class="" css-style=""
  attribute-id="child_dob" is-mandatory="true" is-read-only="false" selection-style="" input-style="d">
  <typ:current-value>
  <typ:unknown-val/>
  </typ:current-value>
  <typ:default-value>
  <typ:unknown-val/>
  </typ:default-value>
  </typ:date-control>
  </typ:entity-instance-control>
  </typ:containment-relationship-control>
  </typ:screen>
  </typ:investigate-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Step 5 - Collect instances of the "child"

The last response received contained a containment relationship control. This is the control that represents an entity collect screen. We need to collect instances of "the child".

The containment relationship control contains a single entity instance control, representing an instance of the entity that is being collected. Inside the entity instance control is a single date input control for the child's date of birth.

The entity instance control in the response is a template - note that the value of the *instance-id* attribute is "-BLANK-". To collect entity instances on this screen, it is necessary to:

1. Make a copy of the template for each entity instance that we want to create.
2. Set the instance-id attribute for each instance.
3. Set the values of any child controls of each entity instance control.

Note: If there was no need to collect any entity instances, it would simply be a matter of deleting the template instance. Also note that the template entity instance control is ignored in a request - any entity instance control in a request with instance-id "-BLANK-" will be completely ignored by the server.

The following example shows how to collect two instances of "the child" with ID,s "child1" and "child2":

Request

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>

```

```

<soapenv:Body>
  <typ:investigate-request>
    <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
    <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
    <typ:screen id="qs$s3@Interviews_Screens_xint$global$global" name="s3@Interviews_Screens_xint" title="Collect the children"
      context-entity-id="global" context-instance-id="global" type="question" is-automatic="false">
        <typ:containment-relationship-control is-visible="true" caption-style="" is-html="false" css-class="" css-style=""
          relationship-id="children"
            source-entity-id="global" source-instance-id="global" target-entity-id="child" relationship-type="OneToMany"
              display-style="Portrait">
                <typ:entity-instance-control is-visible="true" caption-style="" is-html="false" css-class="" css-style="" entity-id="child"
                  instance-id="child1">
                    <typ:date-control is-visible="true" caption="What is the child's date of birth?" caption-style="" is-html="false"
                      css-class="" css-style=""
                        attribute-id="child_dob" is-mandatory="true" is-read-only="false" selection-style="" input-style="d">
                          <typ:current-value>
                            <typ:date-val>1996-01-01</typ:date-val>
                          </typ:current-value>
                          <typ:default-value>
                            <typ:unknown-val/>
                          </typ:default-value>
                        </typ:date-control>
                      </typ:entity-instance-control>
                    <typ:entity-instance-control is-visible="true" caption-style="" is-html="false" css-class="" css-style="" entity-id="child"
                      instance-id="child2">
                        <typ:date-control is-visible="true" caption="What is the child's date of birth?" caption-style="" is-html="false"
                          css-class="" css-style=""
                            attribute-id="child_dob" is-mandatory="true" is-read-only="false" selection-style="" input-style="d">
                              <typ:current-value>
                                <typ:date-val>1997-01-01</typ:date-val>
                              </typ:current-value>
                              <typ:default-value>
                                <typ:unknown-val/>
                              </typ:default-value>
                            </typ:date-control>
                          </typ:entity-instance-control>
                        </typ:containment-relationship-control>
                      </typ:screen>
                    </typ:investigate-request>
                  </soapenv:Body>
                </soapenv:Envelope>

```

Response

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1100</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:investigate-response>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
      <typ:progress>0.5</typ:progress>
      <typ:screen id="qs$s4@Interviews_Screens_xint$global$global" name="s4@Interviews_Screens_xint" title-
e="Collect the schools"
      context-entity-id="global" context-instance-id="global" type="question" is-automatic="false">
        <typ:containment-relationship-control is-visible="true" caption-style="" is-html="false" css-class="" css-style=""
relationship-id="schools"
        source-entity-id="global" source-instance-id="global" target-entity-id="school" relationship-type="OneToMany"
display-style="Portrait">
          <typ:entity-instance-control is-visible="true" caption-style="" is-html="false" css-class="" css-style="" entity-
id="school"
          instance-id="--BLANK--">
            <typ:number-control is-visible="true" caption="What is the school's number of students?" caption-style="" is-
html="false"
            css-class="" css-style="" attribute-id="school_num_students" is-mandatory="true" is-read-only="false"
selection-style="">
              <typ:current-value>
                <typ:unknown-val/>
              </typ:current-value>
              <typ:default-value>
                <typ:unknown-val/>
              </typ:default-value>
            </typ:number-control>
            <typ:text-control is-visible="true" caption="What is the school's type?" caption-style="" is-html="false" css-
class=""
            css-style="" attribute-id="school_type" is-mandatory="true" is-read-only="false" selection-style="" line-
count="1">
              <typ:current-value>
                <typ:unknown-val/>
              </typ:current-value>
              <typ:default-value>
                <typ:unknown-val/>
              </typ:default-value>
            </typ:text-control>
```

```

        </typ:entity-instance-control>
    </typ:containment-relationship-control>
</typ:screen>
</typ:investigate-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Step 6 - Collect instances of the "school"

The response from the previous step contained another containment relationship control, this time to collect instances of "the school".

The template entity instance control contains a two input controls, a text input control for "the school's type", and a number input control for "the school's number of students". Note that the text input control has two list options - "PRIMARY" or "SECONDARY". These are the only valid values that this input can be set to.

Here we will create a single instance of "the school" called "school1", that has type is "SECONDARY", and that has 1200 students:

Request

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:investigate-request>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
      <typ:screen id="qs$s4@Interviews_Screens_xint$global$global" name="s4@Interviews_Screens_xint" title-
e="Collect the schools"
      context-entity-id="global" context-instance-id="global" type="question" is-automatic="false">
        <typ:containment-relationship-control is-visible="true" caption-style="" is-html="false" css-class="" css-style=""
relationship-id="schools"
        source-entity-id="global" source-instance-id="global" target-entity-id="school" relationship-type="OneToMany"
display-style="Portrait">
          <typ:entity-instance-control is-visible="true" caption-style="" is-html="false" css-class="" css-style="" entity-
id="school"
          instance-id="school1">
            <typ:number-control is-visible="true" caption="What is the school's number of students?" caption-style="" is-
html="false" css-class=""
            css-style="" attribute-id="school_num_students" is-mandatory="true" is-read-only="false" selection-
style="">
              <typ:current-value>
                <typ:number-val>1200</typ:number-val>
              </typ:current-value>
            </typ:number-control>
            <typ:text-control is-visible="true" caption="What is the school's type?" caption-style="" is-html="false" css-
class="" css-style=""
            attribute-id="school_type" is-mandatory="true" is-read-only="false" selection-style="" line-count="1">
              <typ:current-value>

```

```

        <typ:text-val>SECONDARY</typ:text-val>
      </typ:current-value>
    </typ:text-control>
  </typ:entity-instance-control>
</typ:containment-relationship-control>
</typ:screen>
</typ:investigate-request>
</soapenv:Body>
</soapenv:Envelope>

```

Response

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1100</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:investigate-response>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
      <typ:progress>0.8333333333333334</typ:progress>
      <typ:screen id="qs$s5@Interviews_Screens_xint$child$child1" name="s5@Interviews_Screens_xint" title="Set
the child's school"
      context-entity-id="child" context-instance-id="child1" type="question" is-automatic="false">
        <typ:reference-relationship-control is-visible="true" caption="the child's school" caption-style="" is-html="false"
css-class="" css-style=""
        relationship-id="childschool" source-entity-id="child" source-instance-id="child1" target-entity-id="school" rela-
tionship-
        type="ManyToOne" display-style="Dropdown">
          <typ:set-targets></typ:set-targets>
          <typ:possible-targets>
            <typ:target display-value="school1">school1</typ:target>
          </typ:possible-targets>
        </typ:reference-relationship-control>
      </typ:screen>
    </typ:investigate-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Step 7 - Set the relationship's target

The response from the last step contained a reference relationship control. This control is used to associated already existing entity instances via a relationship. This particular control is for setting "the child's school" for "child1". The available targets for the relationship are shown inside the "possible-targets" tag. To set the relationship, the appropriate targets should be copied from the "possible-targets" list and pasted inside the "set-targets" tag.

The possible target listed is "school1", so we set this as the single target of the relationship:

Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:investigate-request>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
      <typ:screen id="qs$s5@Interviews_Screens_xint$child$child1" name="s5@Interviews_Screens_xint" title="Set
the child's school"
      context-entity-id="child" context-instance-id="child1" type="question" is-automatic="false">
        <typ:reference-relationship-control is-visible="true" caption="the child's school" caption-style="" is-html="false"
css-class="" css-style=""
        relationship-id="childsschool" source-entity-id="child" source-instance-id="child1" target-entity-id="school"
relationship-type="ManyToOne" display-style="Dropdown">
          <typ:set-targets>
            <typ:target display-value="school1">school1</typ:target>
          </typ:set-targets>
          <typ:possible-targets>
            <typ:target display-value="school1">school1</typ:target>
          </typ:possible-targets>
        </typ:reference-relationship-control>
      </typ:screen>
    </typ:investigate-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1100</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
```

```

<typ:investigate-response>
  <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
  <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
  <typ:progress>0.9166666666666666</typ:progress>
  <typ:screen id="qs$s5@Interviews_Screens_xint$child$child2" name="s5@Interviews_Screens_xint" title="Set
the child's school"
  context-entity-id="child" context-instance-id="child2" type="question" is-automatic="false">
  <typ:reference-relationship-control is-visible="true" caption="the child's school" caption-style="" is-html="false"
css-class="" css-style=""
  relationship-id="childsschool" source-entity-id="child" source-instance-id="child2" target-entity-id="school" rela-
tionship-
  type="ManyToOne" display-style="Dropdown">
  <typ:set-targets></typ:set-targets>
  <typ:possible-targets>
  <typ:target display-value="school1">school1</typ:target>
  </typ:possible-targets>
  </typ:reference-relationship-control>
</typ:screen>
</typ:investigate-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Step 8 - Set a second relationship's target

The response from the last step contained another reference relationship control. This time to the "the child's school" for "child2". Again, we set "school1" as the only target for the relationship:

Request

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
  <typ:investigate-request>
  <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
  <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
  <typ:screen id="qs$s5@Interviews_Screens_xint$child$child2" name="s5@Interviews_Screens_xint" title="Set
the child's school"
  context-entity-id="child" context-instance-id="child2" type="question" is-automatic="false">
  <typ:reference-relationship-control is-visible="true" caption="the child's school" caption-style="" is-html="false"
css-class="" css-style=""
  relationship-id="childsschool" source-entity-id="child" source-instance-id="child2" target-entity-id="school"
relationship-type="ManyToOne" display-style="Dropdown">
  <typ:set-targets>
  <typ:target display-value="school1">school1</typ:target>
  </typ:set-targets>
  <typ:possible-targets>
  <typ:target display-value="school1">school1</typ:target>

```

```

        </typ:possible-targets>
    </typ:reference-relationship-control>
</typ:screen>
</typ:investigate-request>
</soapenv:Body>
</soapenv:Envelope>

```

Response

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1100</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:investigate-response>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:goal-state>Attribute~person_eligible~global~global</typ:goal-state>
      <typ:progress>1.0</typ:progress>
      <typ:screen id="summary" name="summary" title="" context-entity-id="global" context-instance-id="global" type-
e="summary"
        is-automatic="true">
        <typ:goal-control is-visible="true" caption="The person is eligible for education expenses assistance." caption-
style="" is-html="false"
          css-class="" css-style="" goal-id="Attribute~person_eligible~global~global" entity-id="global" instance-id=-
="global" attribute-id="person_
eligible" has-decision-report="true" is-enabled="true" is-user-provided-caption-text="false">
          <typ:goal-value>
            <typ:boolean-val>true</typ:boolean-val>
          </typ:goal-value>
        </typ:goal-control>
      </typ:screen>
    </typ:investigate-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

This response is a summary screen. It indicates that the goal "The person is eligible for education expenses assistance" is true. This is the end of the investigation.

Step 9 - Get a Decision Report

The summary screen has told us that the goal "The person is eligible for education expenses assistance" is true - but why was it true? To find out, we need to

get a decision report using the Get Decision Report operation.

Request

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:get-decision-report-request>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:goal-id>Attribute~person_eligible~global~global</typ:goal-id>
      <typ:show-silent>true</typ:show-silent>
      <typ:show-invisible>true</typ:show-invisible>
    </typ:get-decision-report-request>
  </soapenv:Body>
</soapenv:Envelope>
```

Response

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1100</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:get-decision-report-response>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
      <typ:screen id="report$attr$person_eligible$global$global$Relevant$true$true" name="report$attr$person_
eligible$global$global$Relevant$true$true" title="The person is eligible for education expenses assistance." con-
text-entity-id="global"
context-instance-id="global" type="decision-report" is-automatic="true">
      <typ:decision-report-control is-visible="true" caption="The person is eligible for education expenses assistance."
caption-style=""
is-html="false" css-class="" css-style="" ignore-silent="true" ignore-invisible="true">
      <typ:attribute-node-control is-visible="true" caption="The person is eligible for education expenses assistance."
caption-style=""
is-html="false" css-class="" css-style="" id="report$attr$person_eligible$global$global$Relevant$true$true0"
is-base-level="false"
is-user-set="false" is-known="true" attribute-id="person_eligible" entity-id="global" instance-id="global">
      <typ:attribute-value>
        <typ:boolean-val>true</typ:boolean-val>
      </typ:attribute-value>
      <typ:relationship-node-control is-visible="true" caption="the children" caption-style="" is-html="false" css-
```

```
class="" css-style=""
    id="report$attr$person_eligible$global$global$Relevant>true>true1" is-base-level="true" is-user-set="true"
is-known="true"
    relationship-id="children" source-entity-id="global" target-entity-id="child" relationship-type="OneToMany"
    source-instance-id="global" set-on-screen="qs$s3@Interviews_Screens_xint$global$global">
    <typ:entity-instance-node-control is-visible="true" caption="child1" caption-style="" is-html="false" css-
class="" css-style=""
        id="report$attr$person_eligible$global$global$Relevant>true>true2" is-base-level="true" is-user-set-
t="true" is-known="true"
            entity-id="child" instance-id="child1"/>
        <typ:entity-instance-node-control is-visible="true" caption="child2" caption-style="" is-html="false" css-
class="" css-style=""
            id="report$attr$person_eligible$global$global$Relevant>true>true3" is-base-level="true" is-user-set-
t="true" is-known="true"
                entity-id="child" instance-id="child2"/>
        </typ:relationship-node-control>
        <typ:relationship-node-control is-visible="true" caption="the child's school" caption-style="" is-html="false"
css-class="" css-style=""
            id="report$attr$person_eligible$global$global$Relevant>true>true4" is-base-level="true" is-user-set="true"
is-known="true"
                relationship-id="childsschool" source-entity-id="child" target-entity-id="school" relationship-type-
e="ManyToOne"
                source-instance-id="child1" set-on-screen="qs$s5@Interviews_Screens_xint$child$child1">
                <typ:entity-instance-node-control is-visible="true" caption="school1" caption-style="" is-html="false" css-
class="" css-style=""
                    id="report$attr$person_eligible$global$global$Relevant>true>true5" is-base-level="true" is-user-set-
t="true" is-known="true"
                        entity-id="school" instance-id="school1"/>
                    </typ:relationship-node-control>
                    <typ:relationship-node-control is-visible="true" caption="the child's school" caption-style="" is-html="false"
css-class="" css-style=""
                        id="report$attr$person_eligible$global$global$Relevant>true>true6" is-base-level="true" is-user-set="true"
is-known="true"
                            relationship-id="childsschool" source-entity-id="child" target-entity-id="school" relationship-type-
e="ManyToOne"
                            source-instance-id="child2" set-on-screen="qs$s5@Interviews_Screens_xint$child$child2">
                            <typ:entity-instance-node-control is-visible="true" caption="school1" caption-style="" is-html="false" css-
class="" css-style=""
                                id="report$attr$person_eligible$global$global$Relevant>true>true7" is-base-level="true" is-user-set-
t="true" is-known="true"
                                    entity-id="school" instance-id="school1"/>
                                </typ:relationship-node-control>
                                <typ:attribute-node-control is-visible="true" caption="The person's annual salary is $50,000.00." caption-
style="" is-html="false"
                                    css-class="" css-style="" id="report$attr$person_eligible$global$global$Relevant>true>true8" is-base-
```

```
level="true" is-user-set="true"
  is-known="true" attribute-id="person_salary" entity-id="global" instance-id="global" set-on-screen-
n="qs$2@Interviews_Screens_
  xint$global$global">
  <typ:attribute-value>
    <typ:number-val>50000.0</typ:number-val>
  </typ:attribute-value>
</typ:attribute-node-control> <typ:attribute-node-control is-visible="true" caption="The child's date of birth
is 1/01/96."
  caption-style="" is-html="false" css-class="" css-style="" id="report$attr$person_eli-
gible$global$global$Relevant$true$true9"
  is-base-level="true" is-user-set="true" is-known="true" attribute-id="child_dob" entity-id="child" instance-
id="child1"
  set-on-screen="qs$3@Interviews_Screens_xint$global$global">
  <typ:attribute-value>
    <typ:date-val>1996-01-01</typ:date-val>
  </typ:attribute-value>
</typ:attribute-node-control>
  <typ:attribute-node-control is-visible="true" caption="The school's type is SECONDARY." caption-style="" is-
html="false" css-class=""
  css-style="" id="report$attr$person_eligible$global$global$Relevant$true$true10" is-base-level="true" is-
user-set="true"
  is-known="true" attribute-id="school_type" entity-id="school" instance-id="school1" set-on-screen-
n="qs$4@Interviews_Screens_
  xint$global$global">
  <typ:attribute-value>
    <typ:text-val>SECONDARY</typ:text-val>
  </typ:attribute-value>
</typ:attribute-node-control>
  <typ:attribute-node-control is-visible="true" caption="The school's number of students is 1,200." caption-
style="" is-html="false"
  css-class="" css-style="" id="report$attr$person_eligible$global$global$Relevant$true$true11" is-base-level-
l="true" is-user-set="true"
  is-known="true" attribute-id="school_num_students" entity-id="school" instance-id="school1" set-on-screen-
n="qs$4@Interviews_
Screens_xint$global$global">
  <typ:attribute-value>
    <typ:number-val>1200.0</typ:number-val>
  </typ:attribute-value>
</typ:attribute-node-control>
  <typ:attribute-node-control is-visible="true" caption="The child's date of birth is 1/01/97." caption-style="" is-
html="false" css-class=""
  css-style="" id="report$attr$person_eligible$global$global$Relevant$true$true12" is-base-level="true" is-user-
set="true"
  is-known="true" attribute-id="child_dob" entity-id="child" instance-id="child2" set-on-
```

```

screen="qs3@Interviews_Screens_
  xint$global$global">
  <typ:attribute-value>
    <typ:date-val>1997-01-01</typ:date-val>
  </typ:attribute-value>
</typ:attribute-node-control>
</typ:attribute-node-control>
</typ:decision-report-control>
</typ:screen>
</typ:get-decision-report-response>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Step 10 - Close the session

Finally, the session needs to be closed:

Request

```

<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <soapenv:Header/>
  <soapenv:Body>
    <typ:close-session-request>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
    </typ:close-session-request>
  </soapenv:Body>
</soapenv:Envelope>

```

Response

```

<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i18n-
n="http://www.w3.org/2005/09/ws-i18n" xmlns:-
typ="http://oracle.com/determinations/server/interview/InterviewServiceTest/types">
  <SOAP-ENV:Header>
    <i18n:international>
      <i18n:locale>en_US</i18n:locale>
      <i18n:tz>GMT+1100</i18n:tz>
    </i18n:international>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <typ:close-session-response>
      <typ:interview-session-id>8597fae8-44dc-4cb9-9f54-0eb037517653</typ:interview-session-id>
    </typ:close-session-response>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Example: Pass in parameters for an Interview Portlet across WSRP

We provide a sample implementation of a parameter setting portlet in the `examples/interview-portlet/parameter-setting` path of your Oracle Policy Automation Runtime Java implementation.

The portlet consists of a Java class **ExampleParameterSettingPortlet** and a deployment descriptor `portlet.xml`.

Note: In order to compile the Java class you will need an implementation of the JSR268 API.

For each parameter that the portlet wishes to set:

1. `portlet.xml` contains:
 - a. A **supported-public-render-parameter** element, nested inside the **portlet** element
 - b. A **public-render-parameter** element, nested inside the **portlet-app** element.
The namespace for each parameter needs to be `http://oracle.com/opa/interview/portlet/10.0` (that is, it needs to match the appropriate namespace defined in the interview portlet's `portlet.xml`).
2. The Java code sets every parameter using `response.setRenderParameter("param", request.getParameter("param"))`; where **"param"** is the parameter name as defined in `portlet.xml`

To use this parameter setting example:

1. Deploy the parameter setting portlet (use either `examples/interview-portlet/parameter-setting/ExampleParameterSettingPortlet.war` or compile your own as per the guidelines above).
2. Deploy the Oracle Policy Automation Interview Portlet
3. Place the parameter setting portlet and the Oracle Policy Automation Interview Portlet on the same page
4. Input the parameters in the parameter setting portlet, and click the **Submit** button to pass parameters to the Oracle Policy Automation Interview Portlet

Example: Retrieve a decision report

Once a value has been found for an attribute (possibly by investigating a goal), it is useful to be able to explain why the attribute has that particular value.

A decision report is a recursive structure that contains a series of nodes, and explains the series of steps that led to an attribute having a particular value. A simple example follows.

The applicant is eligible for a driver's license **because:**

The applicant is over 16, **because:**

The applicant's age is 18, **because:**

The applicant's date of birth is 27th February 1985, **and**

The date of application is 14th May 2003

Note:

The bold statements do not appear in a decision report, but are added here for clarity

Each line above would be represented by a **DecisionReportNode** object.

Each **DecisionReportNode** may contain other **DecisionReportNode** objects as children. In the example above, 'The date of application' and 'The applicant's date of birth' are both children of the **DecisionReportNode** containing 'The applicant's age'.

Retrieving a decision report is simply a matter of calling **Attribute.getDecisionReport()** for the Attribute whose value needs to be explained. The task of displaying the decision report to the user is more difficult. However, the recursive nature of the **DecisionReportNode** objects makes it natural to write a recursive method to display the report.

The following code fragments follow on directly from the examples shown under Investigate a Goal, and use a recursive method to display a report for the global attribute 'a5' - click on the appropriate heading to view either the Java or the C# example:

Java code example:

```
Engine eng = Engine.INSTANCE;
Rulebase rb = eng.getRulebase(...);
Session sess = eng.createSession(rb);
EntityInstance globalEI = sess.getGlobalEntityInstance();
Entity globalEnt = globalEI.getEntity();

//set attribute values here
sess.think();

Attribute goalAttr = globalEnt.getAttribute("a5");
String goalValue = (String) goalAttr.getValue(globalEI);

System.out.println("The value of attribute a5 is " + goalValue);
System.out.println("This conclusion was reached because:");

// Now get the decision report so it can be displayed.
DecisionReportNode decisionRep = goalAttr.getDecisionReport(globalEI);

// Display the decision report, starting at an indentation
```

```
// level of 0. This is a recursive method.  
displayDecisionReport(decisionRep, 0);
```

DisplayDecisionReport() is the method that displays each node of the decision report. For readability, each node of the decision report is indented under its parent node. The method is defined as follows:

```
private static void displayDecisionReport(DecisionReportNode node, int depth) {  
  
    for(int j = 0; j < depth; j++)  
    {  
        System.out.print(" ");  
    }  
    System.out.println(node.getAttributeText());  
    if(! node.isLeaf())  
    { List children = node.getChildren();  
      for(int i = 0; i < children.size(); i++)  
      {  
          displayDecisionReport((DecisionReportNode) children.get(i), depth + 1);  
      }  
    }  
}
```

C# code example:

```
Engine eng = Engine.INSTANCE;  
Rulebase rb = eng.GetRulebase(...);  
Session sess = eng.CreateSession(rb);  
EntityInstance globalEI = sess.GetGlobalEntityInstance();  
Entity globalEnt = globalEI.GetEntity();  
  
//set attribute values here  
sess.Think();  
  
RBAttr goalAttr = globalEnt.GetAttribute("a5");  
string goalValue = (string) goalAttr.GetValue(globalEI);  
  
Console.Out.WriteLine("The value of attribute a5 is " + goalValue);  
Console.Out.WriteLine("This conclusion was reached because:");  
  
// Now get the decision report so it can be displayed.  
// We pass in RELEVANT for the flags to retrieve all relevant  
// nodes for the decision report.  
DecisionReportNode decisionRep = goalAttr.GetDecisionReport(globalEI, DecisionReportFlag.RELEVANT);  
  
// Display the decision report, starting at an indentation  
// level of 0. This is a recursive method.
```

```
DisplayDecisionReport(decisionRep, 0);
```

DisplayDecisionReport() is the method that displays each node of the decision report. For readability, each node of the decision report is indented under its parent node. The method is defined as follows:

```
private static void DisplayDecisionReport(DecisionReportNode node, int depth)
{
    for (int j = 0; j < depth; j++)
    {
        Console.Out.Write(" ");
    }
    Console.Out.WriteLine(node.GetAttributeText());

    if (! node.IsLeaf())
    {
        List children = node.GetChildren();
        foreach (DecisionReportNode child in children)
        {
            DisplayDecisionReport(child, depth + 1);
        }
    }
}
```

Example: Run the Batch Processor (InsuranceFraudScore)

The following example demonstrates how to run the Oracle Policy Automation Batch Processor. It reads in csv files as input, and writes the results out as csv files.

You can find the data (csv) files and the configuration file for this example at [examples/determinations-batch/InsuranceFraudScore](#) in the Oracle Policy Automation Runtime package.

Run the Batch Processor

1. Open a command prompt and go to the [examples/determinations-batch/InsuranceFraudScore](#).
2. From the command line execute the Batch Processor - this is slightly different for Java or .NET:

JAVA:

```
java -jar ../../engine/determinations-batch.jar --config fraud_score_config.xml
```

.NET

```
..\..\engine\Determinations.Batch.exe --config fraud_score_config.xml
```

The Batch Processor will run, executing across the data files in the csv directory. It will write the results to the output directory.

Look at the Batch Processor results

The Batch Processor results are written to the output directory where you should find the same two files as in the csv (input) directory:

- global.csv
- previous_claim.csv

The output file *global.csv* has the values inferred by the Batch Processor written to the following fields:

- (total_fraud_score)
- (total_fraud_score_green)
- (total_fraud_score_amber)
- (total_fraud_score_red).

For example:

The first five columns of *global.csv* are the id row, "#", followed by four rows where output has been written (indicated by the braces, but also defined in the *fraud_score_config.xml*):

```
#, (total_fraud_score), (total_fraud_score_green), (total_fraud_score_amber), (total_fraud_score_red), ...
```

For case number 1 (the first row in *global.csv*), the following has been written to the output files.

```
1, 560, false, false, true, ...
```

Total_fraud_score = 560

Total_fraud_score_green = false

Total_fraud_score_amber = false

Total_fraud_score_red = true

So, for this case, the total fraud score is 560, giving it a "red" (high risk status).

The other fields in this csv file are the inputs, which have been copied across from the input *global.csv* file. An output file contains the inferred outputs as well as the original input values.

Look at the Batch Processor inputs

Now that you have seen the output files, it is worth looking at the original input files in the csv directory to see what has changed.

global.csv

Each row in *global.csv* represents a global entity instance, which corresponds to a separate case. Each row in this file is processed as a case and written to the output file. Each column in the csv file represents an attribute for the entity **global**.

previous_claim.csv

- Each row in the *previous_claim.csv* represents a **previous_claim** entity instance.
- A claim can have more than one previous claim associated with it, so these **previous_claim** instances must be associated with rows in the *global.csv*. In the rulebase, this is done by the containment relationship with the public name "previous_claims".
- With regards to the csv file, there is a column in the *previous_claim.csv* that is a foreign key reference to a row in the *global.csv* file. The field in *previous_claim.csv* is called "case_fk".

Look at the configuration file

The purpose of a Batch Processor configuration file is to map input data to rulebase data, and also to specify the inputs outputs and batch run parameters.

Options

Options can be set in a configuration file or on the command line. The configuration file for the *InsuranceFraudScore* batch process sets parameters such as:

- The csv input directory.
- The output type and directory (csv written to the output directory).
- The number of processors to use (1).
- The rulebase location.

```
<options>
  <csv>csv</csv>
  <output type="csv">output</output>
  <processors>1</processors>
  <rulebase>../examples/rulebases/compiled/InsuranceFraudScore.zip</rulebase>
</options>
```

Mappings

Mappings define how the input data will be mapped to Oracle Policy Automation data. For the *InsuranceFraudScore* batch process there are mappings for two entities:

- **global**
- **previous_claim**

For both of these entities there is a mapping for each attribute which references the Oracle Policy Automation attribute, the row which contains the value, and whether it is an an output value. As well as defining attributes, the relationship between **global** and **previous_claim** is defined as follows:

```
<relationship name="previous_claims" source-entity="global" rel-source="global" foreign-key="case_fk"/>
```

This element in the **previous_claim** mapping allows the batch processor to associate **previous_claim** instances with a **global** instance by the foreign key reference in the column "case_fk"

Example: Send external events to other portlets

This example shows how to develop a plugin for the interview portlet that sends an event to another portlet. The example consists of:

- An external event sending plugin that implements **OnAfterProcessActionEventHandler** to send a *GoalKnown* event when a goal becomes known during an investigation
- A sample event receiving portlet that listens to the *GoalKnown* event

Event sender

The source code for this plugin can be found in the `examples\interview-portlet\event-sending` directory of the Java runtime zip.

The plugin consists of one Java class **PortletEventSender** which implements the **OnAfterProcessActionEventHandler** interface. The **handleEvent** method accesses the **SessionContext** object in the **OnAfterProcessActionEvent** to determine whether a goal is known. If so, it raises an external event to other portlets using the following code:

```
QName qname = new QName("http://oracle.com/opa/interview/portlet/10.0", "GoalKnown");
event.getActionResponse().setEvent(qname, goal.getId());
```

Additionally, the Oracle Policy Automation Interview Portlet is declared as a publisher of the *GoalKnown* event by way of the following entries in *portlet.xml* for the Oracle Policy Automation Interview Portlet:

In the **portlet** element:

```
<supported-publishing-event>
  <qname xmlns:x="http://oracle.com/opa/interview/portlet/10.0">x:GoalKnown</qname>
</supported-publishing-event>
```

In the **portlet-app** element:

```
<event-definition>
  <qname xmlns:x="http://oracle.com/opa/interview/portlet/10.0">x:GoalKnown</qname>
  <value-type>java.lang.String</value-type>
</event-definition>
```

If you wish to send other events from the plugin, simply declare them *portlet.xml* for the Oracle Policy Automation Interview Portlet.

Event receiving portlet

We provide a sample portlet that listens to the *GoalKnown* event; the source code of which can be found in the `examples\interview-portlet\event-receiving` directory of the Java runtime zip. The event receiving portlet implements the following method to receive the event:

```
public void processEvent(EventRequest request, EventResponse response) throws PortletException, java.io.IOException {
  String eventName = request.getEvent().getName();
  String eventValue = (String) request.getEvent().getValue();
```

```
response.setRenderParameter("eventName", eventName);
response.setRenderParameter("eventValue", eventValue);
}
```

Additionally, the receiving portlet is defined as a receiver of the *GoalKnown* event by way of the following entries in its *portlet.xml*:

```
<supported-processing-event>
  <qname xmlns:x="http://oracle.com/opa/interview/portlet/10.0">x:GoalKnown</qname>
</supported-processing-event>
```

```
<event-definition>
  <qname xmlns:x="http://oracle.com/opa/interview/portlet/10.0">x:GoalKnown</qname>
  <value-type>java.lang.String</value-type>
</event-definition>
```

Example: Show or Hide an Attribute

This example shows how to show or hide an attribute control based on the value of another attribute control without the need for submitting the screen.

Create a project

First create a project that comprises a screen with two questions so we can hide/show one question based on the value of the other question.

Create the rulebase

To create the rulebase, do the following:

1. Create a new project inside Oracle Policy Modeling.
2. Add a new properties file.
3. Open it and add the text attribute "the person's gender" to the global entity. Right click on the attribute and Copy Attribute ID.
4. Add the text attribute "the person's name" to the global entity. In the Gender attribute field, paste the attribute ID from the above step. This gives automatic access to gender attribute values male and female.
5. Create a new Word rule document and open it.
6. Add the following rule to the Word document:

the person is eligible for maternity leave if

the person is pregnant

7. Click on Shortcut Rule in the Oracle Policy Modeling tab, add a rule to infer that if the gender is male then the person cannot be pregnant:

shortcut rule

the person is not pregnant if

the person's gender = "male"

8. Compile the Word document and correct any errors.
9. Close the Word document.
10. Open the properties file. Right click in the Attributes area and Generate Public Names.
11. Rename the automatically generated public names as below:
 - a. the person is pregnant -> person_pregnant.
 - b. the person is eligible for maternity leave -> eligible_maternity_leave.

Create the screens

To create the screens, do the following:

1. Inside the project, create a new screens file and open it.
2. Create a question screen to collect a person's name and find out the gender of the person and whether they are pregnant or not. These attributes will be collected on the same question screen.

- a. a text control for "the person's name".
 - b. "the person's gender" should have a radio buttons of two items - "male" and "female".
 - c. "the person is pregnant" should be a regular boolean question.
3. Create a summary screen to find out whether the person is eligible for maternity leave or not.
 - a. Add the goal "the person is eligible for maternity leave"
 4. Build & debug to ensure the rulebase is working correctly. If there are any errors, fix them before proceeding with the steps below.

We will now add some custom properties that will tell us how the visibility of one attribute is dictated by another one. Based on whether the person chooses their gender as male or female, the question about whether the person is pregnant or not either hidden or shown. If male, the question about being pregnant is no longer applicable. We do this by adding the following three custom properties.

- **AttributeToCheck**
this is the attribute whose value will decide whether we will display the second question or not.
- **AttributeToShowOrHide**
this is the attribute whose visibility will be toggled based on the value of **AttributeToCheck**.
- **ExpectedAttributeValue**
this is the value against which the selected value of **AttributeToCheck** will be matched. If the value matches the expected value then show the second screen, otherwise hide it.

Next, go to **File -> Project Properties -> Screen** and add the following property name-value pairs to the screen:

Name	Default_value
AttributeToCheck	gender
AttributeToShowOrHide	person_pregnant
ExpectedAttributeValue	female

The sample rulebase can be found at the following location in either the Java or .NET runtime zip file: [examples\rule-bases\source>ShowHideAttributeExample.zip](#).

Modify the velocity templates

The Velocity templates need to be modified to recognize when a person chooses answer to one question, and based on that either hide or show the second question. The files for this example can be found at [examples\web-determinations\custom-show-hide-control](#) in both the Java and .NET runtime zip files

The customization comprises the following files:

File	Description
templates/investigation/form.vm	This is the modified version of form.vm from the web-determinations template. This collects the custom properties set for a screen. If the control being displayed is the same as the one associated with AttributeToCheck, it attaches a click handler to the control. When an attribute value is selected, the click handler gets fired and sets the visibility of the control associated with AttributeToShowOrHide based on the selected value and the expected value.
templates/javascript/clickHandler.js	Javascript code for attaching the click handler and controlling the visibility.
templates/question_screen.vm	Modified to include the javascript file.

Run the example

To run the example use the *ShowHideAttributeExample* project under `examples\rulebases\source` in the runtime zip file, do the following:

1. Open the project in Oracle Policy Modeling and Build and Debug with screens; this will create a *Release* directory.
2. Copy over the templates directory (in the Java or .NET runtime zip file) from:
`examples\web-determinations\custom-show-hide-control\src`
to:
`examples\rulebases\ShowHideAttributeExample\Release\web-determinations\WEB-INF\classes`
3. Build and (Debug or Run) with screens

Example: Use the OnInterviewSessionCreatedEvent to pre-seed data into a newly created session

The most common implementation for **OnInterviewSessionCreatedEvent** Handlers is to pre-form additional configuration on a newly created Interview Session; for example, pre-seeding it with data before the interview commences. This example shows how to pre-seed data from values provided on the query string.

Setup

To deploy this example, do the following:

1. Copy the rulebase .zip file (*PocketMoneyComputation.zip*) from `examples\rulebases\compiled` to the rulebase folder in Web Determinations (for example, `<webroot>\WEB-INF\classes\rulebases`).
2. Install the *PreSeedValues.jar* file in Web Determinations (found at `examples\web-determinations\preseed-values`).

How the sample works

This sample works by checking query parameters of the request that starts the **InterviewSession** and then pre-seeds any provided values for the attributes "the person" and "the pocket money base rate". The expected query string parameters are key from the attribute's public name; that is:

- The query string parameter **person_name** controls the value of the "the person" attribute.
- The query string parameter **base_rate** controls the value of the "the pocket money base rate" attribute.

Example:

If we start a session with the following URL:

```
http://myserver/web-determinations/startsession/PocketMoneyComputation/en-GB?person_name=Fred&base_rate=2.50
```

it will start an **InterviewSession** and pre-seed the person as Fred and the pocket money base rate as 2.50.

Example: Use the Batch Processor with a database

This example demonstrates how to run the Oracle Policy Automation Batch Processor read data from database tables, and writing the values back to the database.

Note: Currently, only the Java version of the Batch Processor can read and write to the database, so this example can be run for the Java Batch Processor only.

The database

This example uses the sample "HR" database for Oracle 10g or 11g. You can easily create this schema by ensuring that the *Sample Schemas* option is turned on when creating a new database via the Oracle Database Configuration Assistant.

SQL scripts are provided to create (and also remove) some extra tables; these scripts do the following:

1. Create some extra tables in the HR schema to hold Batch Processor return data **OPA_EMPLOYEES**, **OPA_LOCATIONS** and **OPA_DEPARTMENTS**. These tables each have a single column and a foreign key back to the original table.
2. Create some extra views to join the OPA tables to the original tables (**OPA_EMPLOYEES_VIEW**, **OPA_LOCATIONS_VIEW**, **OPA_DEPARTMENTS_VIEW**).
3. Populate each created table with an empty reference for each row in its corresponding HR table.

The Batch Processor reads data from the created views and writes the output data back to these views. This technique of combining tables through views and then having the Batch Processor read and write to this view, is useful as it allows you to write Batch Processor output to a dedicated table without affecting the original table.

It is not necessary to read and write to a view, but this allows us to demonstrate a Batch Processor which does not affect the original source tables.

Set up the database for the Batch Processor run

In order to run this example, the extra tables and views described above are required. An **sqlplus** script is provided to create the necessary objects. To run the SQL scripts, do the following:

1. Open a command prompt and go to the `examples/determinations-batch/HRDatabase` directory.
2. Start **sqlplus** and connect to the database with the sample "HR" schema using a user with permissions to create tables and views (for example, SYS as SYSDBA).
3. Execute the script `hr_create_tables.sql` by typing **@hr_create_tables.sql <return>**.
4. Run the Batch Processor.

Also, you need to ensure that the user "HR" is unlocked. This user is locked by default in the sample database. The configuration file assumes that password for the user HR is "HR". To unlock the HR user, do the following:

1. Start **sqlplus** and connect to the same database as above.
2. Execute the following command: **alter user HR account unlock identified by HR;<return>**.

This command will unlock the user HR and set the password for this user to HR. If you choose to use another password, you will need to modify the database url in the configuration.

From a command prompt in the `examples/determinations-batch/HRDatabase` directory, execute the Batch Processor with the following command:

```
java -jar ../../engine/determinations-batch.jar
```

The batch process will run, executing across the data in the database. The results will be written back to the database.

Looking at the Batch Processor results

The Batch Processor should successfully complete with a statement such as: *Finishing batch processor. Cases processed: 23 ...*

Each case consists of a row from the **OPA_LOCATIONS_VIEW** view, and related rows from the **OPA_DEPARTMENTS_VIEW** and **OPA_EMPLOYEES_VIEW** views.

The results are written back to these views, but are also written to the **OPA_LOCATIONS**, **OPA_DEPARTMENTS** and **OPA_EMPLOYEES** tables.

For each row of each table there is a single output value as shown in the following table:

table	updated column	description of output
opa_location	total_salary	the total salaries for the location (the sum of all the departments total salaries)
opa_department	total_salary	the total salaries for the department (the sum of all the employee salaries)
opa_employee	long_service_date	the date at which an employee is eligible for long service leave, based on their hire date

You can view the values written back to the table by looking at the views or the Oracle Policy Automation tables; for example:

```
SELECT * FROM HR.OPA_EMPLOYEES_VIEW;
```

will return all the data for employees, including the **long_service_date** column.

Looking at the rules

You can look at the rulebase used by this Batch Processor example, by going to the rulebase project located at: [examples/rulebases/source/human-resources.zip](#).

If you unzip and open the project, and examine the *Rules.doc* document, you will see the simple rules which calculate the Batch Processor output.

The data model can be found in the *properties.xsrc* file.

Looking at the configuration file

The file *config.xml* is automatically picked up by the Batch Processor because it is called *config.xml* and it is in the working directory. The configuration file contains the mappings from the tables to Oracle Policy Automation data and also the run options for the Batch Processor.

Options

```
<options>
  <base>opa_locations_view</base>
```

```

<rulebase>../../rulebases/compiled/human-resources.zip</rulebase>
<database>
  <url>jdbc:oracle:thin:HR/HR@localhost:1521:xe</url>
  <driver>oracle.jdbc.OracleDriver</driver>
  <driversrc>.\lib\ojdbc5.jar</driversrc>
</database>
<output type="db" />
</options>

```

The options above specify how the Batch Processor will run, including the database connection information. In this section you can see the base table, the path to the rulebase used, the database connection information (including url, driver and driver source) and the output specification (to the database).

Mappings

```

<mappings>
  <mapping entity="global" table="opa_locations_view" primary-key="location_id">
    <attribute name="street-address" field="street_address" />
    <attribute name="city" field="city" />
    <attribute name="postal-code" field="postal_code" />
    <attribute name="state-province" field="state_province" />
    <attribute name="location-total-salary-cost" field="total_salary" out-
put="true" />
  </mapping>

  <mapping entity="department" table="opa_departments_view" primary-key="department_
id">
    <attribute name="department-name" field="department_name" />
    <attribute name="department-total-salary-cost" field="total_salary" out-
put="true" />
    <relationship name="all-departments" source-entity="global" foreign-key-
y="location_id" />
  </mapping>

  <mapping entity="employee" table="opa_employees_view" primary-key="employee_id">
    <attribute name="first-name" field="first_name" />
    <attribute name="last-name" field="last_name" />
    <attribute name="hire-date" field="hire_date" />
    <attribute name="salary" field="salary" />
    <attribute name="commission-pct" field="commission_pct" />
    <attribute name="long-service-date" field="long_service_date" output="true"/>
    <relationship name="all-employees" source-entity="department" foreign-key-
y="department_id" />
  </mapping>
</mappings>

```

The mappings above specify how the database tables will be translated to the rulebase data model. In this section you can see the entities are mapped to tables, with entity attributes mapped to columns of the relevant tables. Relationships between the entities, including containment, are specified by the foreign key relationships between the tables; for example:

- The entity *employee* is obtained from the table view **opa_employees_view**, and the entity attributes *first-name*, *last-name* and so on are all mapped to columns in the table.
- The containment relationship *all-employees* is specified by the foreign key relationship between department and employee using the **department_id** column of the employee table. This column is a reference to the department that the employee belongs to.

Removing the Batch Processor tables and views

When you have finished with this example, you may want to remove the extra tables and views created; to do this:

1. Open a command prompt and go to the [examples/determinations-batch/HRDatabase](#) directory.
2. Start **sqlplus**.
3. Connect to the database with the sample "HR" schema using a user with permissions to create tables and views; for example, SYS as SYSDBA.
4. Execute the script *hr_remove_tables.sql* by typing: **@hr_remove_tables.sql <return>**.

This should remove the following tables and views: **OPA_EMPLOYEES**, **OPA_LOCATIONS**, **OPA_DEPARTMENTS**, **OPA_EMPLOYEES_VIEW**, **OPA_LOCATIONS_VIEW**, and **OPA_DEPARTMENTS_VIEW**.

Commentary - Sample Code (DerbyCommentary)

This sample Commentary plugin returns HTML content. It retrieves the HTML content from a DERBY database, using the target. The Commentary plugin in this example does the following:

- It returns HTML content for commentary
- It can verify whether a target screen/attribute has available commentary or not
- It retrieves the HTML content from a DERBY database

The sample code will cover the following:

- Create a Commentary plugin that returns HTML content
- Demonstrate how to use the 'target' screen/attribute ID together with the InterviewSession object to verify that data for the target exists in a datasource
- Demonstrate how to use the 'target' screen/attribute ID together with the InterviewSession object to retrieve data from a datasource

Setup

This sample code needs the following to run:

- DerbyCommentary (Commentary plugin)
- *Parents and Children* rulebase ([examples\rulebases\compiled\Parents and Children.zip](#))
- DERBY SQL database ([examples\interview-engine\derby-commentary\src\create_db.sql](#))
 - Further information about the DERBY database can be found at <http://db.apache.org/derby>

About the DERBY SQL database

The DERBY SQL database is created by using *create_db.sql* which is located in [examples\interview-engine\derby-commentary\src](#).

The 'COMMENT_LOCATION' column contains the ID of the 'target' to match.

The 'COMMENT_CONTENT' column contains the HTML commentary content for the 'target'.

Commentary
COMMENT_ID
COMMENT_LOCATION
COMMENT_CONTENT

About the DerbyCommentary plugin

- This is a demonstration on creating a Commentary plugin that only returns HTML content by setting **isCommentaryRedirect()** to return false all the time
- Uses the 'target' screen/attribute ID together with the locale of the interview (from InterviewSession) to check the DERBY database if content for the target exists

- Uses the 'target' screen/attribute ID together with the locale of the interview (from InterviewSession) to retrieve data from the DERBY database.
- The DERBY database stores the commentary data for each target as full HTML, so the plugin does not do any processing of the commentary data, it simply returns it as is.

To setup this scenario:

1. Create the SQL Database using *create_db.sql* which can be found in `examples\interview-engine\derby-commentary\src`
2. Copy the rulebase .zip file (for example, *Parents and Children.zip*) from `examples\rulebases\compiled` to the rulebase folder in Web Determinations (for example, `<webroot>\WEB-INF\classes\rulebases`)
3. Copy and install the *DerbyCommentary.jar* file (located in `examples\interview-engine\derby-commentary`) into Web Determinations; for more information, refer to [Create a Plugin](#)
4. Copy the DERBY libraries to the library folder in Web Determinations (for example, `<webroot>\WEB-INF\lib`)
5. Ensure that DERBY is run in network server mode
6. Run a Web Determinations Interview

How the DerbyCommentary plugin works

Once registered, Web Determinations will call **DerbyCommentary.isCommentaryEnabled()** when a screen is to be displayed; the DerbyCommentary plugin always returns true.

When a screen is to be displayed, Web Determinations calls **DerbyCommentary.hasCommentary()** method for the screen itself, to check if commentary information exists for the screen. The same method is also called for each attribute control to be displayed in the screen. **DerbyCommentary.hasCommentary()** uses the target to query DERBY database and check if commentary data is available for the target screen/attribute.

If **DerbyCommentary.hasCommentary()** returns true for a target, its label is rendered as a link.

When the user clicks on the label link, Web Determinations calls **DerbyCommentary.isCommentaryRedirect()** to check if the commentary for the target (the clicked label link) returns URL or HTML content; the DerbyCommentary plugin always returns false, since it can only provide commentary in HTML content.

Web Determinations then calls **DerbyCommentary.getCommentaryContent()** to get the HTML commentary content for the target. DerbyCommentary queries the DERBY database and retrieves the commentary for the target. As mentioned before, the DERBY database has stored the commentary in pure HTML format, so DerbyCommentary does not need to process the data from DERBY database into HTML. DerbyCommentary encapsulates the HTML string in a TypedInputStream, and returns it to Web Determinations for display. A sequence similar to the above, can be found in [Commentary Plugin](#)

Source Code

To view the source code for the DerbyCommentary sample, refer to `examples\interview-engine\derby-commentary` in the Java runtime zip file.

Commentary - Sample Code (RedirectCommentary)

The Commentary plugin in this example does the following:

- It returns URLs for commentary
- It can verify whether a target screen/attribute has available commentary or not
- It retrieves the URL for a target via a simple target-URL Map object

The sample code will cover the following:

- Create a Commentary plugin that exclusively returns URL for commentary
- Demonstrate how to use the 'target' screen/attribute ID together with the InterviewSession object to verify that URL for the target exists in the mapping object
- Demonstrate how to use the 'target' screen/attribute ID together with the InterviewSession object to retrieve the URL

Setup

This sample code needs the following to run:

- RedirectCommentary (Commentary plugin)
- *Parents and Children.zip* rulebase

About the RedirectCommentary plugin

- This is a demonstration on creating a Commentary plugin that only returns URLs by setting **isCommentaryRedirect()** to return true all the time
- Uses the 'target' screen/attribute ID together with the locale of the interview (from InterviewSession) to check the mapping object if the target exists
- Uses the 'target' screen/attribute ID together with the locale of the interview (from InterviewSession) to retrieve the URL for the target in the mapping object

To setup this scenario:

1. Copy the rulebase .zip file (*Parents and Children.zip*) from `examples\rulebases\compiled` to the rulebase folder in Web Determinations (for example, `<webroot>\WEB-INF\classes\rulebases`)
2. Copy and install the *RedirectCommentary.jar* file or the *RedirectCommentary.dll* file (located in `examples\interview-engine\redirect-commentary` in either the Java or .NET runtime zip) into Web Determinations; for more information, refer to [Create a Plugin](#).

If you wish to modify the RedirectCommentary code, do the following:

- a. Copy the code for the RedirectCommentary plugin
 - b. Modify the static map (variable `redirectMap`) used for the targets and URL mappings in the code if necessary
 - c. Modify the **getInstance(arguments)** method for registration of the plugin with the rulebase if necessary
 - d. Compile and JAR (or build the DLL in .NET) the RedirectCommentary
3. Run a Web Determinations Interview

How the RedirectCommentary plugin works

Once registered, Web Determinations will call the **RedirectCommentary.isCommentaryEnabled()** when a screen is to be displayed; the RedirectCommentary plugin always returns true.

When a screen is to be displayed, Web Determinations calls **RedirectCommentary.hasCommentary()** method for the screen itself, to check if commentary information exists for the screen. The same method is also called for each attribute control to be displayed in the screen. **RedirectCommentary.hasCommentary()** uses the target information and the locale information from InterviewSession to verify if a URL exists for the target in the mapping object

If **RedirectCommentary.hasCommentary()** returns true for a target, its label is rendered as a link.

When the user clicks on the label link, Web Determinations calls **RedirectCommentary.isCommentaryRedirect()** to check if the commentary for the target (the clicked label link) returns URL or HTML content; the RedirectCommentary plugin always returns true.

Web Determinations then calls **RedirectCommentary.getCommentaryContent()** to get the URL for the target. RedirectCommentary simply retrieves the value from the mapping object using the target String and the locale from the InterviewSession as the key. RedirectCommentary returns the URL value as is, and Web Determinations displays the webpage of the URL as commentary.

A similar sequence of the above can be seen in [Commentary Plugin](#)

Source Code

To view the source code for the RedirectCommentary sample, refer to [examples/interview-engine/redirect-commentary](#) in either the Java or .NET runtime zip file.

Custom Control - BenefitCode Walkthrough Example

This example demonstrates a custom control plugin for Web Determinations. For more general information on constructing and debugging custom control plugins, see [Create a Custom Control](#)

In our example, we have a kind of attribute in the rulebase (a benefit code) where the attribute is stored as a single text string, but the string has multiple parts that we want the end-user to edit with separate HTML widgets.

The example custom control provider replaces any text input control where the screen author in Oracle Policy Modeling has set the control's HTML class to 'BenefitCode'.

The example includes two re-usable abstract classes, *InterviewControlWrapper* and *InputInterviewControlWrapper*, which delegate as much as possible to the source control in the rule engine runtime layer, allowing your custom formatter to concentrate on special cases.

To construct this example:

Build a custom control plugin

Build a plugin which has the following four classes:

- *BenefitCodeControlProvider* -- this is the class that Web Determinations will look for and load on startup. It can determine which rulebases or data sessions it wants to attach to; our example plugin will attach to anything. Our class will look for text controls attributes with the HTML class 'BenefitCode' and replace them by a custom control.
- *BenefitCodeControl* -- this is the custom control itself. It is tightly coupled with the same-named *BenefitCodeControl.vm* velocity template. The control and the template work together to render the control and interpret the user input.
- *InterviewControlWrapper* -- this is an abstract convenience class which defers to the rule engine layer's version of the control wherever it can, for all controls.
- *InputInterviewControlWrapper* -- this is an abstract convenience class for input controls, which adds methods to defer to the rule engine layer for input (attribute) controls.

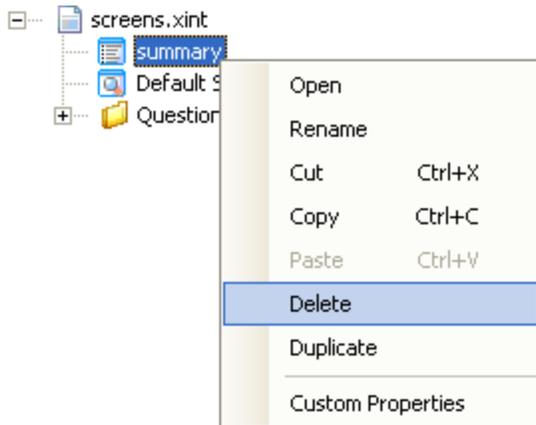
1. Copy the uncompiled rulebase .zip file found at [examples\rulebases\source\BenefitCodeExample.zip](#) (in either the Java or .NET runtime zip file) to a place on your machine where it can be unzipped ready for use.
2. Copy the *CustomControlBenefitCode.jar* file or the *CustomControlBenefitCode.dll* file from [examples\web-determinations\custom-control](#) (in either the Java or .NET runtime zip file) to your rulebase's plugins directory, where Oracle Web Determinations can find it. If your rulebase is at `<...>\BenefitCodeExample`, the correct directory for your plugin is `<...>\BenefitCodeExample\Release\web-determinations\WEB-INF\classes\plugins`. If the Release directory does not exist, Build and Debug with Screens to create it.
3. Copy the velocity template file *BenefitCodeControl.vm* from [examples\web-determinations\custom-control\src](#) to your rulebase's templates directory. If your rulebase is at `<...>\BenefitCodeExample`, the correct directory for your control template is `<...>\BenefitCodeExample\Release\web-determinations\WEB-INF\classes\templates\controls\BenefitCodeControl.vm`. The file name is important here - it **must** match the name of the custom control class exactly or the file will not be found.
4. Build and Debug with Screens; your plugin should now be loading.

Customize the rulebase to use the plugin

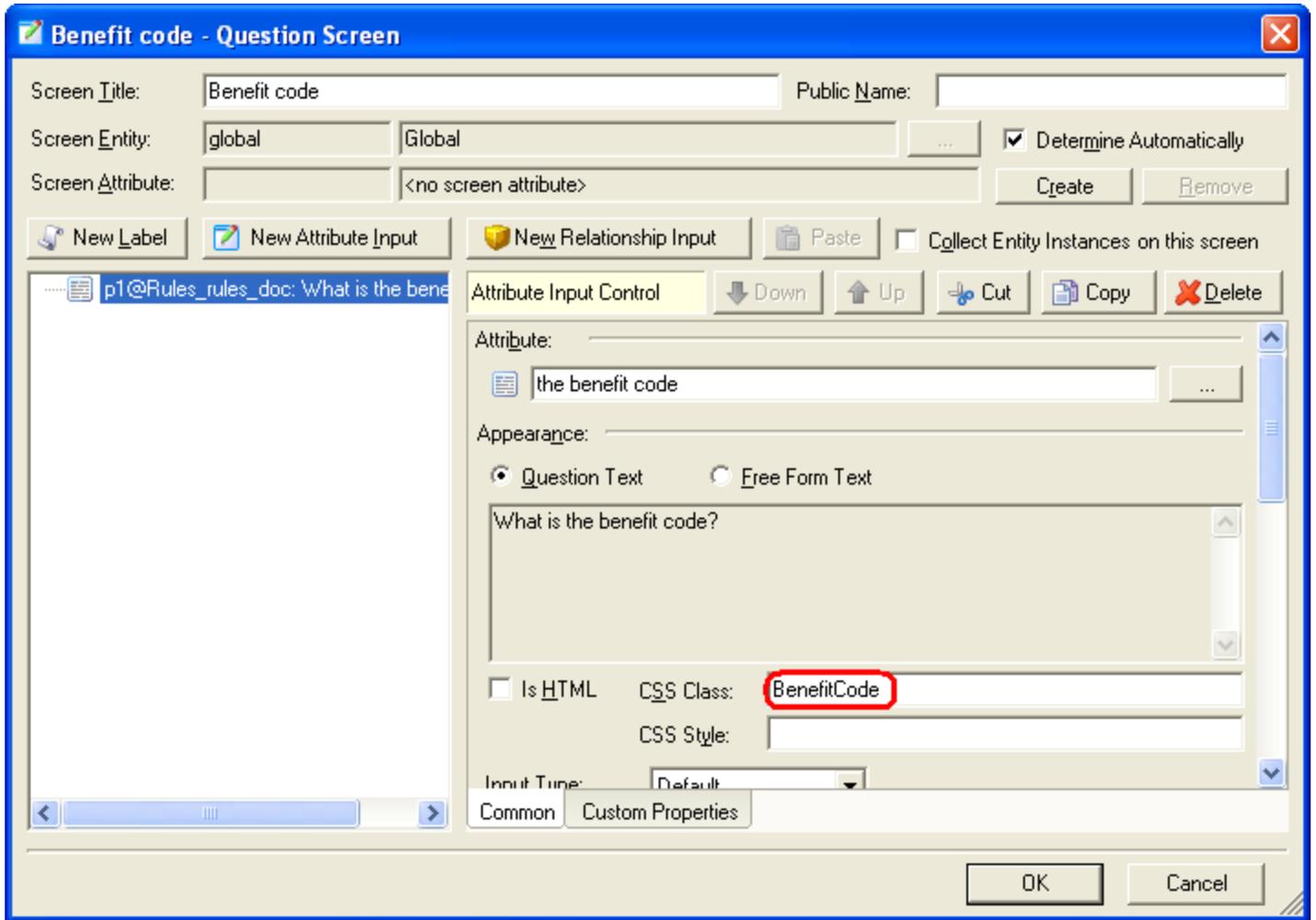
BenefitCodeExample is already setup to use the plugin, but here are details on how to configure any rulebase to use it.

Now we want to invoke our plugin to use our custom control. The plugin is looking for an HTML class, "BenefitCode" so we need to create a screen and put that class on the control.

1. In Oracle Policy Modeling, right-click the *Screens* folder, and choose "Add New Screens File". Call it "screens".
2. By default, screens files are created with an empty summary screen. A real application would carefully customize this screen, but in our simple example we will delete it to use the automatic summary screen instead.



3. Right-click the *Question Screens* folder and select, "New Question Screen"
4. On the new screen, add a control for our benefit code attribute, and set the HTML class of the attribute to "BenefitCode"



5. Click **OK**, save, and build and debug again. Your custom control should now be working!

Benefit code

What is the benefit code?

* Retirement - 1234

6. Note that when you view a decision report or data review screen, the raw text value will be printed; for example, "RET-1234" in this case. If you want your attribute values to be formatted specially even when printed outside of your custom control, you will need to implement a custom formatter as well. See [Formatter Plugin](#)

Source Code

To view the source code for the Custom Control sample, refer to [examples\web-determinations\custom-control](#) in either the Java or .NET runtime zip file.

Custom Control – CalendarDateControl Walkthrough Example

This example demonstrates a custom control plugin for Web Determinations; for more general information on constructing and debugging custom control plugins, see [Create a Custom Control](#).

This example shows a calendar that is written in javascript to assist the user in selecting a date for an attribute input of a type that is a date in the question screen. The calendar looks like this:



<< < February 2012 > >>						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29			

The example custom control provider replaces any date input control where the custom property **UseCalendarDateControl** is set to *True* and the input type is set to *Single Edit (Default)*.

It includes two re-usable abstract classes, **InterviewControlWrapper** and **InputInterviewControlWrapper**, which delegate as much as possible to the source control in the rule engine runtime layer, allowing the custom formatter to concentrate on special cases.

Construct the example

The custom control plugin is built using the following classes:

CalendarDateControlProvider

the class that Web Determinations will look for and load on startup. It can determine which rulebases or data sessions it wants to attach to; our example plugin will attach to anything. Our class will look for date controls with custom property **UseCalendarDateControl** that is set to *True* and the input type that is set to *Single Edit (Default)* and replace them by a custom control.

CalendarDateControl

the custom control itself. It is tightly coupled with the same-named *CalendarDateControl.vm*. The control and template work together to render the control and interpret the user input.

InterviewControlWrapper

an abstract convenience class which defers to the rule engine layer's version of the control wherever it can, for all controls.

InputInterviewControlWrapper

an abstract convenience class for input controls, which adds methods to defer to the rule engine layer for input (attribute) controls.

The steps for building this custom control are:

1. Copy the uncompiled rulebase .zip file found at `examples\rulebases\source\HealthyEating.zip` to a place on your machine where it can be unzipped and ready for use.
2. Copy the `CalendarDateControl.jar` file found at `examples\web-determinations\calendar-date-control` to your rulebase's plugins directory, where Oracle Web Determinations can find it. If your rulebase is at `<...>\HealthyEating`, the correct directory for your plugin is `<...>\HealthyEating\Release\web-determinations\WEB-INF\classes\plugins`. If the 'Release' directory does not exist, Build and Debug with screens once, to create it.
3. Copy the velocity template file `CalendarDateControl.vm` from `examples\web-determinations\calendar-date-control\src` to your rulebase's templates directory. If your rulebase is at `<...>\HealthyEating`, the correct location for your control template is `<...>\HealthyEating\Release\web-determinations\WEB-INF\classes\templates\controls\CalendarDateControl.vm`. The file name is important here – it must match the name of the custom control class exactly or the file will not be found.
4. Copy the calendar javascript file `calendar.vm` from `examples\web-determinations\calendar-date-control\src` to your rulebase's includes directory. If your rulebase is at `<...>\HealthyEating`, the correct location for your javascript file is `<...>\HealthyEating\Release\web-determinations\WEB-INF\classes\templates\includes\calendar.vm`.

Note: The calendar javascript file was renamed to have a .vm extension because it will not be recognized by the velocity engine unless it has a .vm extension.

5. Modify the velocity template for the question screen to include the calendar javascript file. If your rulebase is at `<...>\HealthyEating`, the velocity template for the question screen is located at `<...>\HealthyEating\Release\web-determinations\WEB-INF\classes\templates\question_screen.vm`.

Locate the line where `javascript-utilities.vm` is used in the velocity template and add a reference for the calendar javascript file. The line to be inserted should have the format `#parse("includes/<javascript.vm>")`. For this example, it should be `#parse("includes/calendar.vm")`.

The following is a sample modification:

```
question_screen.vm - Notepad
File Edit Format View Help
#if ( ${opa-commentary-type} == "frameset" )
    #set($commentaryTarget = ${opa-commentary-frameset})
#end

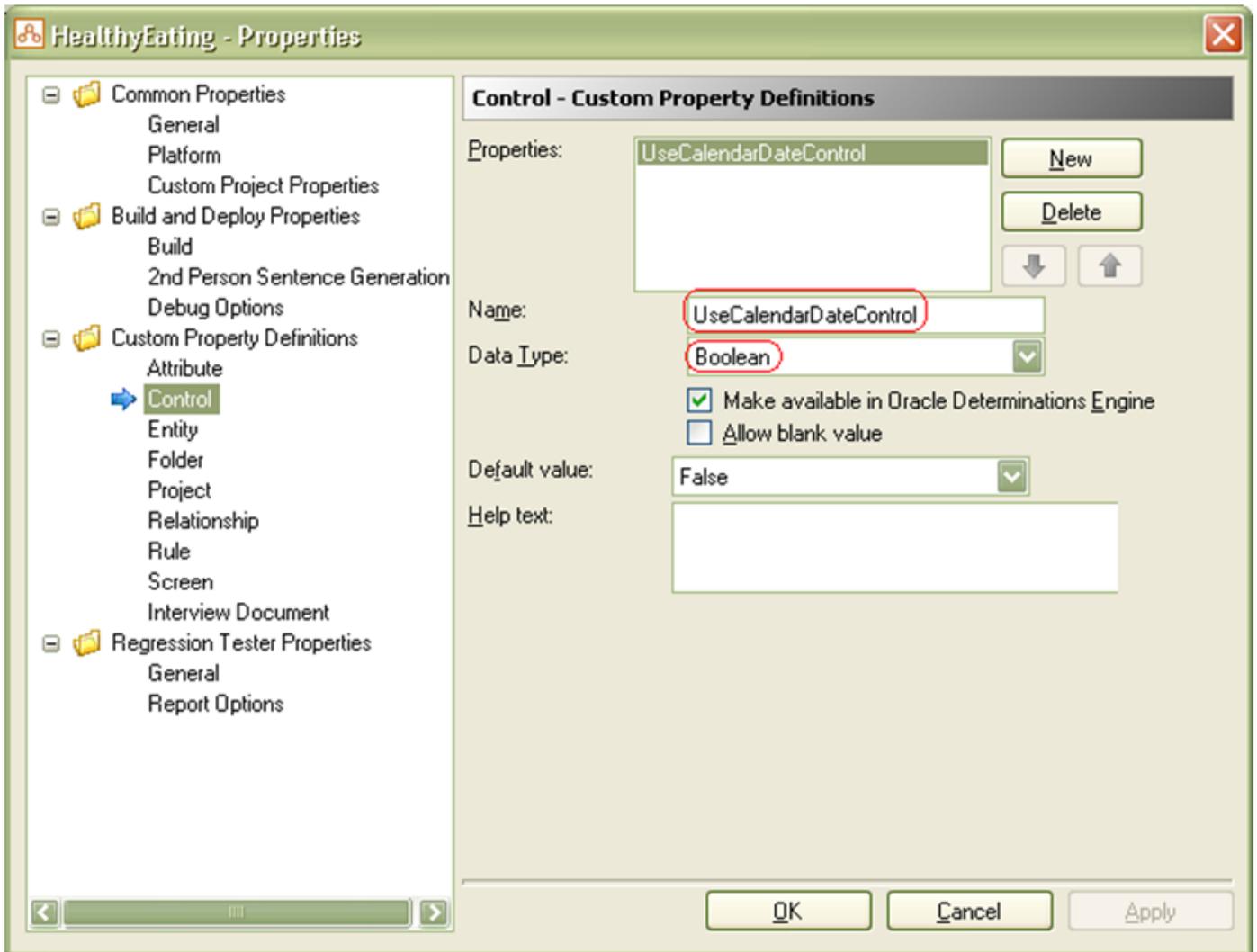
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html lang="${screen.getLocale()}" dir="${pagedirString}">
  <head>
    <meta http-equiv="content-type" content="text/html; charset=utf-8">
    <meta http-equiv="content-language" content="${screen.getLocale()}">
    <title>${screen.getTitle()} - ${header-title}</title>
    <link rel="stylesheet" type="text/css" href="${context-root-path}${resource-request}/reset.css">
    <script type="text/javascript">
      ## These includes need to be present if a "form" HTML element will be present (e.g. form.v
      #parse("includes/javascript-utilities.vm")
      ## Include the javascripts that will be used by the question screen and give them extensio
      #parse("includes/calendar.vm")
    </script>
    <style type="text/css">
  <!--
  ${css-text}
  -->
```

A modified velocity template file for the question screen is also found at [examples\web-determinations\calendar-date-control\src\question_screen.vm](#), if you would like to replace your existing velocity template.

Customize the rulebase to use the plugin

The *HealthyEating* rulebase is already setup to use the plugin, but here are details on how to configure any rulebase that contains a date attribute input control to use it.

1. In Oracle Policy Modeling, go to the **File** menu and select **Project Properties...**
2. Add a new custom property definition for a control. Name the new custom property **UseCalendarDateControl** and change its data type to boolean; for example:



3. Click **OK** to accept the custom property.
4. Open the screen file that contains the *Questions Screens* folder
5. Edit the date attribute input control found in the *Question Screens* folder. Make sure the input type is set to **Single Edit (Default)** and the custom property **UseCalendarDateControl** is set to *True*; for example:

Attribute Input Control

Down Up Cut Copy Delete

Attribute:

Appearance:

Question Text Free Form Text

Assessment date:

Is HTML CSS Class:
CSS Style:

Input Type:

Style:

Input type:

Dynamic Default Value:

Attribute:

Value:

Common Custom Properties

Attribute Input Control

Down Up Cut Copy Delete

Custom properties:

UseCalendarControl	True
--------------------	------

UseCalendarControl

Common Custom Properties

6. Click **OK** to accept the changes
7. Save the changes and build and debug with screens. Your custom control should now be working:

Assessment date:

21/02/12

Submit

<<			<			Febru		
Sun	Mon	Tue	Wed	Thu	Fri	Sat	Sun	Mon
5	6	7	8	9	10	11	12	13
12	13	14	15	16	17	18	19	20
19	20	21	22	23	24	25	26	27
26	27	28	29	30	31			

Source Code

To view the source code for the *CalendarDateControl* sample, refer to [examples\web-determinations\calendar-date-control](#) in the Java runtime zip file.

Custom Control - Filtered Dropdown Selection List Walkthrough Example

This example demonstrates how to create a filtered dropdown selection list (FilterListCode) which, when a user starts typing, only displays the relevant options.

Create a project

We first need a rulebase that uses a dropdown list to collect an attribute value during an investigation.

Create the rulebase

1. Create a new project inside Oracle Policy Modeling.
2. Add a new properties file.
3. Open it and add the text attribute "the person's profession" to the global entity.
4. Create a new Word rule document and open it.
5. Add the following rule to the Word document:
the person is eligible if
 the person's profession = "IT"
6. Compile the Word document and fix any errors.
7. Close the word document.
8. Open the properties file. Right click in the Attributes area and Generate Public Names.
9. Rename the automatically generated public name for "the person is eligible" to "person_eligible".

Create the screens

1. Inside the project, create a new screens file and open it.
2. Create a question screen to collect "the person's profession" as a dropdown list.
3. Create a new attribute input for "the person's profession". Change the 'Input Type' to a 'Drop Down List', and specify selection items to add entries for various professions to the dropdown list.
4. Create a summary screen for "the person is eligible".
5. Build & debug to ensure the rulebase is working correctly. If there are any errors, fix them before proceeding with the steps below.

Modify the Velocity templates

We now modify the Velocity templates to add the filtering functionality. The filtered dropdown list (also called a combo-box) is created by using the existing Web Determinations template *DropdownSelection.vm* found under [templates/investigation](#).

Note: Combo box search functionality is unsupported in Opera browsers and hence disabled.

The effect is achieved by using an input field along with the dropdown list. We show or hide the dropdown list based on where the focus is on the page. The user can use the collapse and expand icons to access the dropdown list from the input field; for example, if

a user is typing in the input text field, then the dropdown list is shown. This list shown will only have the options that match the text in the input field. If the user clicks anywhere else other than the expand/collapse icons, the dropdown list will be made invisible. The files can be found under `examples\web-determinations\filtered-list-control\src\templates`. The customization consists of the following files:

File	Description
<code>templates/investigation/Dropdown-selection.vm</code>	Contains the layout of the input text field and the dropdown list that make up a combo-box.
<code>templates/javascript/comboBox.js</code>	Contains the code for the combo-box effect.
<code>templates/javascript/onClick.vm</code>	Contains the code for handling the visibility of the dropdown selection based on where the user clicks on the page.
<code>templates/javascript/filter.js</code>	Contains the code for filtering a list of options based on some text.
<code>templates/question_screen.vm</code>	Modified to include the javascript files.

Run the example

To run the example, do the following:

1. Use the *FilterListExample* project under `examples\rulebases\source`.
2. Open the project in Oracle Policy Modeling.
3. Build and Debug with screens; this creates a *Release* directory.
4. Copy the files from:

`examples\web-determinations\filtered-list-control\src\templates`

to:

`FilterListExample\Release\web-determinations\WEB-INF\classes\templates`.

5. Build and (Debug or Run) with screens.

Create a Custom Validator for control validation

This sample plugin demonstrates custom validation for an individual control. For an example of how to validate an entire screen, refer to the [Create a Custom Validator for screen validation](#) example.

About the sample

In this example, we validate the age details entered by the user to ensure it is a number greater than 0 and less than 130.

The sample code demonstrates:

- Handling the **OnValidateControlEvent** event
- Extracting the current value of a control and validating it using custom logic
- Adding an error to the **TransactionResult** object in the **OnValidateControlEvent** event

The sample code needs the following to run:

- *CustomControlValidator* plugin ([examples\interview-engine\custom-control-validator](#))
- *CustomValidatorExample* rulebase ([examples\rulebases\compiled\CustomValidatorExample.zip](#))

To setup this scenario

1. Copy the rulebase .zip file (*CustomValidatorExample.zip*) from [examples\rulebases\compiled](#) to the rulebase folder in Web Determinations (for example, `<webroot>\WEB-INF\classes\rulebases`)
2. Copy and install the *CustomControlValidator.jar* file (located in [examples\interview-engine\custom-control-validator](#)) into Web Determinations; for more information, refer to [Create a Plugin](#). If you wish to modify the *CustomControlValidator* code, do the following:
 - i. Copy the code for the *CustomControlValidator* plugin
 - ii. Modify the validation code in the **handleEvent** method
 - iii. Compile and JAR or DLL the *CustomControlValidator*
3. Run a Web Determinations Interview

Source

To view the source code for the *CustomScreenValidator* sample, refer to [examples\interview-engine\custom-control-validator](#) in the Java or .NET runtime zip file.

Create a Custom Validator for screen validation

This sample plugin demonstrates screen-wide custom validation. Custom validation of screens is useful when data validity depends on a combination of values from different controls. For an example of how to validate an individual control, refer to the [Create a Custom Validator for control validation](#) example.

About the sample

In this example, we validate the country and postcode details entered by the user using the following rules:

- Australian postcodes must consist of exactly 4 digits
- Postcodes in other countries are not validated

The sample code demonstrates:

- Handling the **OnValidateScreenEvent** event
- Extracting the current value of a control and validating it using a Regular Expression
- Adding an error to the **TransactionResult** object in the **OnValidateScreenEvent** event

The sample code needs the following to run:

- *CustomScreenValidator* plugin ([examples\interview-engine\custom-screen-validator](#))
- *CustomValidatorExample* rulebase ([examples\rulebases\compiled\CustomValidatorExample.zip](#))

To setup this scenario

1. Copy the rulebase .zip file (*CustomValidatorExample.zip*) from [examples\rulebases\compiled](#) to the rulebase folder in Web Determinations (for example, `<webroot>\WEB-INF\classes\rulebases`)
2. Copy and install the *CustomScreenValidator.jar* file (located in [examples\interview-engine\custom-screen-validator](#)) into Web Determinations; for more information, refer to [Create a Plugin](#). If you wish to modify the *CustomScreenValidator* code, do the following:
 - i. Copy the code for the *CustomScreenValidator* plugin.
 - ii. Modify the validation code in the **handleEvent** method.
 - iii. Compile and JAR or DLL the *CustomScreenValidator*.
3. Run a Web Determinations Interview

Source

To view the source code for the *CustomScreenValidator* sample, refer to [examples\interview-engine\custom-screen-validator](#) in the Java or .NET runtime zip file.

Data Adaptor - Sample Code (DerbyDataAdaptor)

The sample DataAdaptor in this example does the following:

- Allows the user to load and save interview data from an SQL database
- Allows the user to view cases available for loading in an SQL database
- Loading and saving entities and their attributes
- Loading and saving a many-many relationship

The sample code demonstrates:

- Creating a custom Data Adaptor plugin that performs all the main Data Adaptor functionality of list, load, and save
- How to retrieve data from the datasource and use the retrieved data to build InterviewUserData using Rulebase model data for load() method (more information in [Understanding the InterviewSession](#))
- How to use the Rulebase model and instance data to build the data to be persisted to the datasource - in this case the SQL inserts (more information in [Understanding the InterviewSession](#))

The sample code is for requirements such as:

- The user can save Interview data
- The user can reload saved Interview data
- The user needs to be able to select which saved Interview data to load
- The Interview data needs to be saved to a datasource so, for example, other systems can use the base or inferred data from the interview
- Pre-seeding the Web Determinations Interview with data from an existing datasource (see [Data Adaptor - Common Scenarios](#) on how to initiate pre-seeded Web Determinations Interviews)
- The user can manually save Interview data for progress, and load it in the future.

Setup

This sample code needs the following to run:

- DerbyDataAdaptor (Data Adaptor plugin - see source below)
- The *Parents and Children* rulebase ([examples\rulebases\compiled\Parents and Children.zip](#))
- DERBY SQL database ([examples\interview-engine\data-adaptor\src\create_db.sql](#))
 - Further information about the DERBY database can be found at <http://db.apache.org/derby>

About the Parents and Children rulebase

The rule is not relevant for this sample, it is the entities, attributes, and relationships that we are interested about (since the Data Adaptor needs to save and load them)

The **Rule entities** it uses are: Global, child, person.

The **relationships** are:

source	relationship type	target	source text	target text
person	m-m	child	personschild	childsparents

The **attributes** relevant to this example:

Global

- the sun is shining

Child

- the child's name
- the child is eating ice-cream

Person

- the person's name

About the DERBY Database

The DERBY SQL database is created by using *create_db.sql* which is located in `examples\interview-engine\data-adaptor\src`.

The Database tables are:

WD_CASE (represents the Global entity)
WD_CASE_ID (VARCHAR(255), PK)
SUN_SHINING(INT)

CHILD
CHILD_ID (INT, PK-IDENTITY)
WD_CASE_ID (VARCHAR(255),FK)
CHILD_NAME (VARCHAR(255))
EATING_ICECREAM(INT)

PERSON
PERSON_ID (INT, PK – IDENTITY)
PERSON_NAME(VARCHAR(255))

CHILD_PARENT (represents the m:m relationship 'childsparents')
CHILD_ID (INT, PK and FK)
PARENT_ID (INT, PK and FK)

About the Derby Data Adaptor plugin

The sample plugin can do the following:

- list all the Case ID's available for loading
- load instance data for the 'child' entity via the Child table
- load instance data for the 'person' entity via the Person table
- load relationship data between the 'child' instances and the 'person' instances
- save 'child' instance data - including its attributes
- save 'person' instance data - including attributes
- save the relationships between the 'child' instances and the 'person' instances

All the main methods of the DerbyDataAdaptor (**listCases**, **load** and **save**) use the methods for connecting and disconnecting to the DERBY database (**connectDBObjects**, **closeDBObjects**). Because those methods connect to the database, they must put the method logic in a try-catch block.

To setup this scenario:

1. Create the SQL Database using *create_db.sql* which can be found in `examples\interview-engine\data-adaptor\src`
2. Copy the rulebase .zip file (*Parents and Children.zip*) from `examples\rulebases\compiled` to the rulebase folder in Web Determinations (for example, `<webroot>\WEB-INF\classes\rulebases`)
3. Copy and install the *DerbyDataAdaptor.jar* file (located in `examples\interview-engine\data-adaptor`) into Web Determinations; for more information, refer to [Create a Plugin](#)
4. Copy the DERBY libraries to library folder in Web Determinations (for example, `<webroot>\WEB-INF\lib`)
5. Ensure that DERBY is run in network server mode
6. Run a Web Determinations Interview

How the DerbyDataAdaptor works

The DerbyDataAdaptor is triggered by the following actions (detailed in the [Data Adaptor Plugin](#) page), that is:

- When the user clicks on the Load, the DataAdaptor **listCases()** is called
- When the user selects a case in the 'Load Case' screen, the DataAdaptor **load()** is called
- If the interview is to be pre-seeded using the constructed URL, the DataAdaptor **load()** is called
- When the user clicks **Save**, the Data Adaptor **save()** is called without going through the 'Save As' process.

- Even if the session already has a Case ID associated to it, the Data Adaptor generates a new Case ID and saves the interview data to that Case ID. The Data Adaptor essentially does not save into an existing Case ID

listCases()

The **listCases()** method is straightforward. It needs to retrieve all the available cases that can be 'loaded'. These cases are stored in the WD_CASE table, so this method simply queries the WD_CASE table, retrieves all the ID's, and returns them to be displayed in the 'Load' screen.

load()

The **load()** method builds the InterviewUserData to be returned with the following steps:

1. The user is authenticated
2. The method connects to the database
3. Creates a new interview user data
4. Retrieves the global record in WD_CASE table by using the case ID
5. Gets the global interview entity instance in the created interview user data
6. Sets the attribute 'sun_shining' in the global instance
7. Retrieves all children from the CHILD table by using the case ID
8. For each child retrieved in step 7;
 - a. An interview entity instance is created for the child
 - b. The 'child_name' and 'eating_icecream' are set in the child's interview entity instance.
 - c. The child's interview entity instance containment parent is set to the global interview entity instance.
9. The global interview entity instance containment for child entity is set to complete.
10. Retrieves all people from the CHILD table by using the case ID
11. For each person retrieved in step 10;
 - a. An interview entity instance is created for the person
 - b. The 'person_name' attribute is set in the person's interview entity instance.
 - c. The person's entity instance containment parent is set to global interview entity instance.
12. The global interview entity instance containment for person entity is set to complete.
13. For each person interview entity instance created in step 11, the relationship 'personschildren' is set by adding the appropriate child interview entity instance/s using the table CHILD_PARENT.
14. The database objects are closed and connection is terminated.

save()

The **save()** method accesses the Rulebase model data (Rulebase object) and the Session instance data (Session object) to save the current Interview user data. For this sample code, the global, child and person entities' instances are saved.

Only base attributes for all entities are saved.

The save process is as follows:

1. The user is authenticated
2. The method connects to the database
3. The global entity instance from the session is written in the WD_CASE table with fields WD_CASE_ID and SUN_SHINING .
4. For each contained child entity instance in the global entity instance, the child entity instance is written in the CHILD table with fields WD_CASE_ID, CHILD_NAME and EATING_ICECREAM.
Note: When a record is written in the CHILD table, the CHILD_ID is automatically generated.
5. For each contained person entity instance in the global entity instance, the person entity instance is written in the PERSON table with fields WD_CASE_ID and PERSON_NAME
Note: When a record is written in the PERSON table, the PERSON_ID is automatically generated.
6. The m:m relationship 'parentschildren' is recorded in the table CHILD_PARENT by using the generated CHILD_ID field values in table CHILD and PERSON_ID field values in table PERSON.
7. The database objects are closed and connection is terminated.

Source Code

To view the source code for the DerbyDataAdaptor sample, refer to [examples/interview-engine/data-adaptor](#) in the Java runtime zip file.

Data Adaptor - Sample Code (Autosave with Derby)

The sample Autosave scenario in this example does the following:

- During the investigation of a goal, as the user answers questions the DataAdaptor 'save' function is called to save after each screen
- After completing the investigation of the goal, the DataAdaptor 'save' function is called to save
- An ID is automatically created for the user when he/she starts the Web Determinations Interview
- The DataAdaptor 'save' function saves all the attributes that has an instance value provided by the user

The sample code demonstrates

- How to setup an Event Handler to handle events (for this scenario, to handle Events fired when the user answers a question, and completing a goal)
- Calling the DataAdaptor functions from an Event Handler
- The Event Handler need to have access to the SessionContext (Platform Events) to provide the necessary input arguments for DataAdaptor **save()** (**case ID** and **InterviewSession**)
- Using an Event Handler and Data Adaptor together to implement 'autosaving' functionality during a Web Determinations Interview

The sample code provides functionality for requirements such as the need to be able to autosave based on triggers.

Setup

This sample code needs the following to run:

- DerbyAutoSave plugin
- ExtFrameworkAutoSave (rulebase)
- DERBY SQL database ([examples\web-determinations\auto-save\src\create_db.sql](#)
 - Further information about the DERBY database can be found at <http://db.apache.org/derby>

About the DERBY SQL database schema

The DERBY SQL database is created by using *create_db.sql* which is located in [examples\web-determinations\auto-save\src](#).

AUTOSAVE
ID
age
income
old
low_income
old_low_income

AUTOSAVE

eligible

About the Plugins:

- The Event Handler that triggers the Data Adaptor to save is the **AutosaveTrigger**,
 - The AutosaverTrigger handles the **OnGetScreenEvent** and **OnInvestigationEndedEvent** (both Events are a Platform event and thus has access to SessionContext - see [Events and Event Handlers](#))
 - The AutosaverTrigger handles both Events by implementing the EventHandler interface for both events.
 - It needs to handle both Events because:
 - The OnGetScreenEvent calls the handler for each event. To prevent the handler from calling the Data Adaptor **save()** outside of an investigation (for example, *Summary* screen, *Data Review*), the handler only calls the Data Adaptor **save()** for **OnGetScreenEvents** when a **goal** is currently being investigated
 - With the restriction of a goal needing to be investigated, the Data Adaptor **save()** is not called when the investigation ends, thus failing to save the user's answers in the last Question screen
 - The **OnInvestigationEndedEvent** is useful for this issue - it calls the handler only when an investigation is finished, and thus saving the user's answers in the last question screen
- The Data Adaptor that handles the save functionality is the **AutosaveSaver**, and it saves onto the DERBY datasource
- The DataAdaptor and Event Handler plugins are configured to work on a rulebase named 'ExtFrameworkAutoSave'.

To setup this scenario:

1. Create the SQL Database using *create_db.sql* which can be found in [examples\web-determinations\auto-save\src](#)
2. Copy the rulebase .zip file (*ExtFrameworkAutoSave.zip*) from [examples\rulebases\compiled](#) to the rulebase folder in Web Determinations (for example, [<webroot>\WEB-INF\classes\rulebases](#))
3. Copy and install the *DerbyAutoSave.jar* file (located in [examples\web-determinations\auto-save](#)) into Web Determinations; for more information, refer to [Create a Plugin](#)
4. Copy the DERBY libraries to the library folder in Web Determinations (for example, [<webroot>\WEB-INF\lib](#))
5. Ensure that DERBY is run in network server mode
6. Run a Web Determinations Interview

Note:

If you need to modify the database connection details or you need class files for your specific database implementation, then you will need to edit the source code, recompile and JAR the DerbyAutoSave.

How the Autosave works

The autosave functionality happens:

- After every investigation question screen
- After an investigation of a goal is completed

When the above events are met, the AutosaveTrigger Event Handler calls the **save()** method of the AutosaveSaver Data Adaptor. The AutosaveSaver simply saves all of the attributes into the Autosave table in the DERBY database.

Table **AUTOSAVE**

ID	age	income	old	low_income	old_low_income	eligible
VARCHAR(255)	DOUBLE	DOUBLE	INT	INT	INT	INT

Table 'Autosave' view - start of investigation, to completing an investigation

While going through the Interview, when the user starts an investigation of a goal and the current Case ID is empty - a new Case ID will be created. You can see this in the database; for example:

ID	age	income	old	low_income	old_low_income	eligible
8821b2f8-76e3-4762-a4a5-af8c10aac521	NULL	NULL	NULL	NULL	NULL	NULL

After the Case ID is generated, as the user answers the questions in the investigation (for example, age=31 below), answers and inferred data will be added onto the same row item in the database (for example, old = Yes, eligible = Yes).

ID	age	income	old	low_income	old_low_income	eligible
8821b2f8-76e3-4762-a4a5-af8c10aac521	31	NULL	Yes	NULL	NULL	Yes

Source Code

To view the source code for the DerbyAutoSave sample, refer to [examples\web-determinations\auto-save](#) in the Java runtime zip file.

Document Generator - Sample Code (TxtDocumentGenerator)

The Document Generator in this example does the following:

- Generates a 'txt' file when the "custom-document-type" Custom Property is detected
- Sends other requests to the BIPublisherDocumentGenerator

The sample code will cover the following:

- Create a simple Document Generator plugin
- Basic methodology to access instance data in the Web Determinations Interview session (passed into the called Document Generator method)
- Use of the existing BIPublisherDocumentGenerator class

Setup

This sample code needs the following to run:

- The 'HealthyEating' rulebase that can be found at [examples\rulebases\compiled\HealthyEating.zip](#) in either the Java or .NET Oracle Policy Automation Runtime package.
- TxtDocumentGenerator (Document Generator plugin)

About the Document Generation Rulebase:

- Setup to display document generation links for 'HTML' and 'PDF' documents using the default Document Generator plugin
- The document links are only displayed once the goal attribute ('visibility_summary_end' - 'the end of interview items should be displayed on the summary screen') has been determined and reached.
- For the HTML and PDF document links, they are driven by template files in the rulebase package
- To setup display of 'TXT' files using TxtDocumentGenerator:
 1. Copy the HealthyEating rulebase source from [examples\rulebases\source](#) (in either the java or .NET runtime zip file) to a place on your machine where it can be unzipped.
 2. Unzip and then open the HealthyEating rulebase in Oracle Policy Modeling.
 3. Go to **File -> Project Properties**.
 4. Select **Custom Property Definitions -> Interview Document**.
 5. Add new property "custom-document-type". Click **OK**.
 6. Open *Screens.xint*.
 7. Under *DocGen* folder, create new Document.
 8. Set document name (that is, "Text Document").
 9. Select any output type and click **OK**.
 10. Right click on the "Text Document" icon and select **Custom Properties**.
 11. Set the value of "custom-document-type" to any string. Click **OK**.
 12. In the summary screen, add a new Document Control to reference "Text Document".
 13. Set the visibility attribute to "visibility_summary_end". Click **OK**.

About the TxtDocumentGenerator plugin:

- This is a demonstration of how to create a simple Document Generator plugin, and using the InterviewSession object passed to the generateDocument() method to access data from the interview
- This Document Generation plugin generates a .txt file
- Only returns a .txt file if the custom property "custom-document-type" is detected and the property value is not blank.

To setup this scenario:

1. Open the HealthyEating rulebase in Oracle Policy Modeling, and run **Build**.
2. Copy the compiled rulebase .zip file (*HealthyEating.zip*) from the project output directory to the rulebase folder in Oracle Web Determinations (for example, `<webroot>\WEB-INF\classes\rulebases`)
3. Copy and install the *TxtDocumentGenerator.jar* file or the *TxtDocumentGenerator.dll* file (located in `examples\interview-engine\document-generator` in either the Java or .NET runtime zip file) into Web Determinations; for more information, refer to [Create a Plugin](#)
4. Run a Web Determinations Interview.

Source Code

To view the source code for the TxtDocumentGenerator sample, refer to `examples/interview-engine/document-generator` in either the Java or .NET runtime zip file.

Formatter - Sample Code

LatitudeFormatter Example for Custom Formatter Plugins

This example demonstrates a formatter plugin for Oracle Web Determinations. For more general information on constructing and debugging plugins, see the topic, [Create a Plugin](#)

The example custom formatter defers to the default formatter for all values except for attribute values where a Number attribute has a custom property IsLatitude set to "True".

For these, it formats and parses numbers as Latitude values eg. 28°32'45.63"S

The example includes a re-usable abstract class, PassthroughFormatter, which delegates all formatting to the default formatter, allowing your custom formatter to concentrate on special cases.

To construct the example:

Test that the LatitudeFormatterExample rulebase is working correctly.

1. Copy the uncompiled *LatitudeFormatterExample.zip* rulebase source from [examples\rulebases\source](#) in either the Java or .NET runtime zip file, to a location on your machine where it can be unzipped ready for use with Oracle Policy Modeling.
2. In Oracle Policy Modeling, choose **Build->Build and Debug..., With screens**.
3. An interview should open, in which you can determine if an applicant is eligible for the imaginary polar-resident benefits by entering a number, such as "-30" or "4000"
4. Note that at this point our web application is accepting these bad latitude values!
5. After determining your eligibility, click on, "Why?" You should get a display like this, showing your input:

The person is not eligible.

- The person does not live in a polar region.
- The latitude of the person's primary residence is -30.

You should now have a functioning rulebase. If your rulebase is not working correctly so far, you should fix any problems before proceeding to the customization steps.

Build a formatter plugin

The next task is to build a plugin. Our plugin has two classes:

- LatitudeFormatterPlugin -- this is the class that Web Determinations will look for and load on startup. It can determine which rulebases or data sessions it wants to attach to; our example plugin is reusable across multiple rulebases and so will attach to anything. Our class will look for number attributes with the custom attribute 'IsLatitude' and format them specially. Commented Java source code for the plugin class can be found below.
- PassthroughFormatterPlugin -- this is an abstract convenience class which defers to the default formatter for all attributes. Commented Java source code can be found below.

Copy the *LatitudeFormatter.jar* file or the *LatitudeFormatter.dll* file (located in [examples\web-determinations\custom-formatter](#) in either the Java or .NET runtime zip) to your rulebase's plugins directory, where Oracle Web Determinations can find it. If your rulebase is at [<...>\LatitudeExample](#), the correct directory for your plugin is [<...>\LatitudeExample\Release\web-](#)

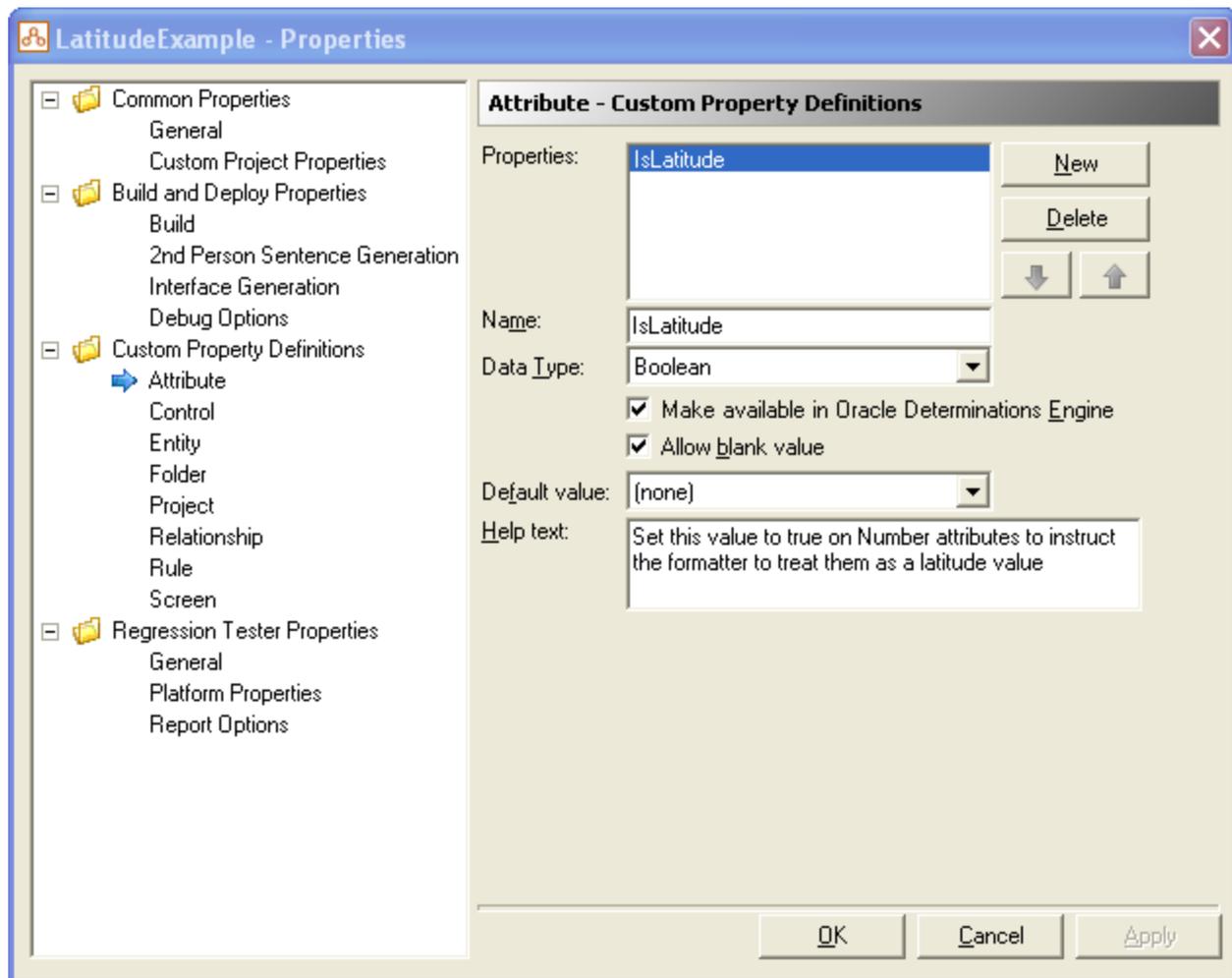
determinations\WEB-INF\classes\plugins. If the Release directory does not exist, Build and Debug your rulebase once to create it; your plugin should now be loading, but it won't be doing anything visible yet.

Customize the rulebase to use the plugin

Now we want to invoke our plugin to format our rulebase's "the latitude of the person's primary residence" attribute as a latitude. The plugin is looking for a custom property, "IsLatitude" so we need to create that property and set it on the special attribute.

Note: Although this customization has already been done for you in the provided sample rulebase (*LatitudeFormatterExample.zip* found in `examples\rulebases\compiled`), the following is a description of how to do it in general:

1. In Oracle Policy Modeling, choose **File->Project Properties...**, add a Boolean custom property called "IsLatitude", and click **OK**:



2. Right-click the "Properties" folder, and choose "Add New Properties File". Call it "properties"
3. Open the properties file, right-click inside the empty "Attributes" section for the global entity, and choose, "Generate Public Names..."
4. Make sure the checkbox is ticked for the attribute, "the latitude of the person's primary residence"

5. The attribute should now show up in the view. Double-click it to edit its properties:

Attribute Editor - a1

ID: p1 Entity: global

Public name: a1 Document: properties.xsrc

Data type: Number

Text: the latitude of the person's primary residence

Min value: Max value: RegExp:

Default gender: Impersonal (it)

Gender attribute: ...

Plural

Allow Substitution

Question: What is the latitude of the person's primary residence?

Statement: the latitude of the person's primary residence is %a1%.

Uncertain: the latitude of the person's primary residence is uncertain.

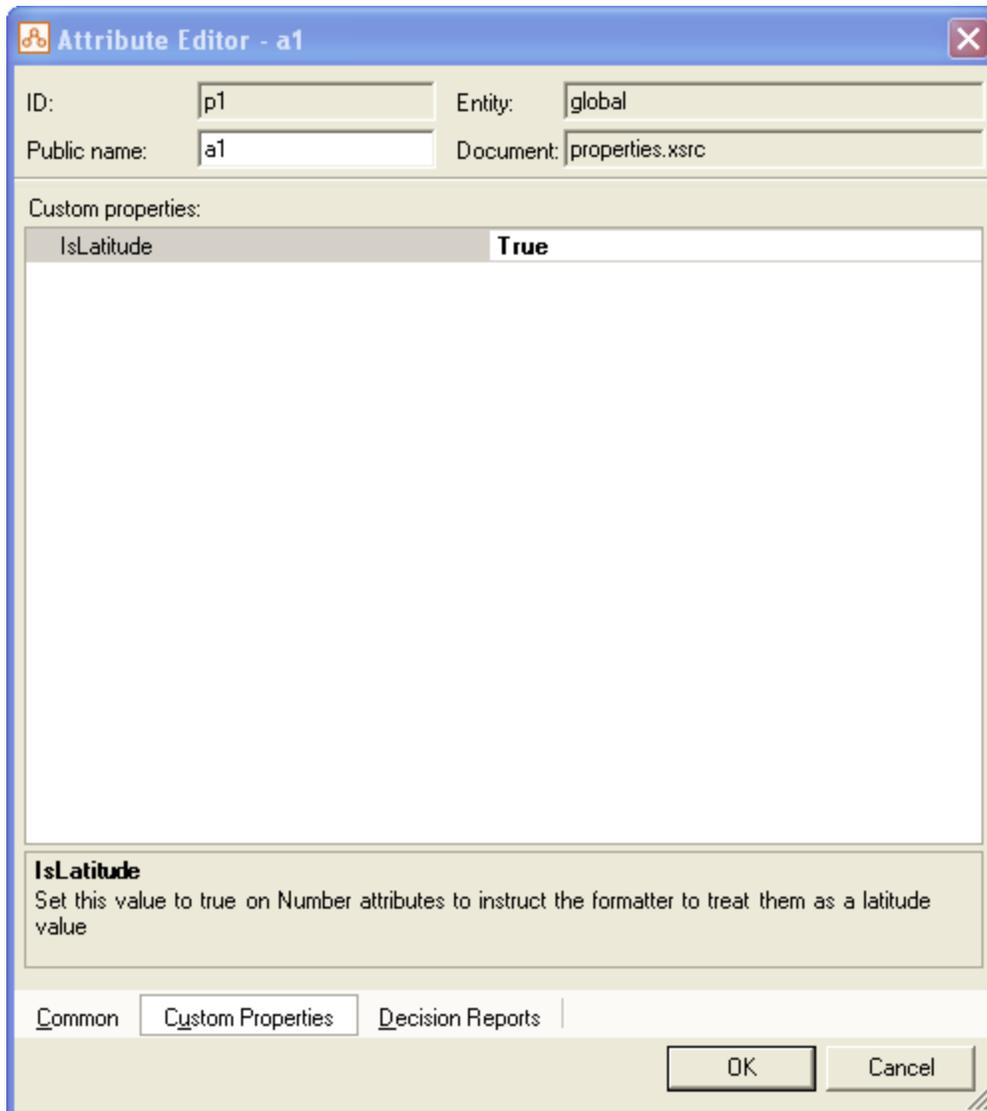
Unknown: the latitude of the person's primary residence is unknown.

Override...

Common Custom Properties Decision Reports

OK Cancel

6. Click the "Custom Properties" tab at the bottom of the dialog, and add a value of "True":



7. Click **OK**, save, and build and debug again. Your custom formatter should now be working! On the latitude question, you may enter a number as before, or you may enter a latitude value such as: 30°12'13"S (Because the ° character can be hard to type, the formatter also accepts a space as in 30 12'13"S)
8. Whether you enter a decimal number or a properly-formatted latitude, the custom formatter will reject values beyond the north or south pole, and show a properly-formatted latitude on the decision report screen:

The person is not eligible.

- The person does not live in a polar region.
 - The latitude of the person's primary residence is 30°12'13"S.

Source Code

To view the source code for the LatitudeFormatter sample, refer to [examples\web-determinations\custom-formatter](#) in either the Java or .NET runtime zip file.

List Provider - Sample Code

The sample ListProvider in this example does the following:

- Retrieves data from a table in an SQL database, based on the List input control to be populated
- Uses the retrieved data to build a list of items, and returns the list to be used in the List input control

The sample code demonstrates

- Accessing the List input control object to read its metadata
- How to retrieve data from an SQL database and use that data to build the list of items for the List input control
- Creating the List of ListOptions to be returned by the method `getListOptions()`, the core method of the class

The sample code provides functionality for requirements such as:

- Loading the list items of a List input control from a database
- Dynamically loading the list items of a List input control, instead of the list items provided by the rule author
- Loading list items of more than one List input controls

Setup

This sample code needs the following to run:

- List Provider plugin (`examples\interview-engine\list-provider` in either Java or .NET runtime zip).
- ExtFrameworkListProvider (rulebase).
- An SQL database - for example, Derby for Java or SQL Express for .NET. Refer to the db script in either of the runtime zips.

About the ExtFrameworkListProvider Rulebase

The rules are not relevant for this sample, it is the attributes that the List Provider needs to provide list options for

The **attributes** relevant to this example:

Global

- `political_alignment`
- `state`

About the SQL database

AU_STATES
ID
STATE_SHORTNAME
STATE_FULLNAME

POLITICAL_ALIGNMENTS
ID
PA_NAME

About the Derby List Provider plugin

- This is a demonstration on creating a simple List Provider plugin, using the (List) InputControl object, the Attribute object linked to the InputInterviewControl, and the InterviewSession object (which contains both Rulebase model and instance data)
- Uses the Attribute's name to determine which database table to use for retrieving the list items data
- Demonstrates how to use the data from the database to build the list of ListOptions, the return object.

To setup this scenario:

1. Create the SQL Database using *create_db.sql* which can be found in `examples\interview-engine\derby-list-provider\src`; for the .NET version, first create the ListProvider database using a tool such as MS SQL Server Management Studio Express.
2. Copy the rulebase .zip file (*ExtFrameworkListProvider.zip*) from `examples\rulebases\compiled` to the rulebase folder in Web Determinations (for example, `<webroot>\WEB-INF\classes\rulebases`)
3. Copy and install the *DerbyListProvider.jar* file (located in `examples\interview-engine\derby-list-provider`) into Web Determinations; for more information, refer to [Create a Plugin](#).
4. For the Java version, ensure that DERBY is run in network server mode and copy the DERBY libraries to the library folder in Web Determinations (for example, `<webroot>\WEB-INF\lib`)
5. For the .NET version, ensure that SQL Server is running and the ASPNET user account has access to the database.
6. Run a Web Determinations Interview

Note:

If you need to modify the database connection details or you need class files for your specific database implementation, then you will need to edit the source code, recompile and create the JAR or DLL.

How the DerbyListProvider works

The DerbyListProvider is always called when a List input control is displayed during the Web Determinations Interview. The DerbyListProvider allows the Web Determinations to use the default list items on the List input control (default list items - the list of values provided by the rule author) unless the Attribute associated to the List input control is 'states' or 'political_alignment'. The DerbyListProvider returns a 'null' object to let the Web Determinations know to use the default list items.

If the Attribute of the List input control is 'states' or 'political_alignment', DerbyListProvider accesses the DERBY database, access either the AU_STATES or POLITICAL_ALIGNMENTS table (depending on the Attribute), and gets the list items. It then constructs a List of ListOptions from the data retrieved. The DerbyListProvider returns the list of ListOptions, which tells Web Determinations that it should use this list as the items in the List input control instead of the default list.

Source Code

To view the source code for the sample, refer to `examples\interview-engine\derby-list-provider` in the Java runtime zip file or `examples\interview-engine\list-provider` in the .NET runtime zip file.

Rulebase Resolver - Sample Code (DerbyRulebaseService)

The most common implementations of [Rulebase Resolver Plugins](#) are for:

- Using a custom datasource to store and retrieve rulebases
- Customizing the rulebase functionality provided by the default Rulebase Resolver Plugin

The Rulebase Resolver Plugin in this example:

- Uses a custom datasource (a Java DERBY database) for retrieving rulebases.
- Stores each rulebase XML data as a Binary Large Object (BLOB).
- Is an approach to storing and retrieving rulebases in a relational database.

Setup

The sample code that follows needs the following to run:

- DerbyRulebaseResolver (Rulebase Resolver plugin)
- A set of rulebases (and modules) that will run/use the plugin
- DERBY SQL database (information about DERBY database in <http://db.apache.org/derby>)

To setup this scenario:

1. Create the SQL Database. It doesn't have to be DERBY, but that is what the source code is set up for. For the database schema, see below.
2. Create the Rulebase in Oracle Policy Modeling, and run Build.
3. Make sure that DERBY is running in network server mode.
4. Upload the rulebase.zip from the rulebase output folder by using the java program 'BlobInsert <rulebase output folder>'. You can upload other rulebases if you want by running the program several times.
5. Copy the code for the DerbyRulebaseService plugin (you may need to modify the database connection details).
6. Compile and JAR the DerbyRulebaseService (you may need class files for your specific database implementation).
7. Install the JAR file in Web Determinations. For more information, refer to [Create a Plugin](#).
8. Copy the DERBY library files to the library directory of your deployment (for example, <webroot>\WEB-INF\lib).
9. Run a Web Determinations Interview.

Database Schema

There are two tables in the database, one called "RULEBASE" which stores all the available rulebases, with the following columns:

Name	Type	Description
ID	VARCHAR(255) (Primary Key)	Corresponds to the rulebase's unique identifier
CONTENT	BLOB (2M)	The rulebase archive

Derby Create Table Command:

```
create table RULEBASE(ID VARCHAR(255) not null primary key, CONTENT BLOB(2M));
```

The second table called "MODULE" stores the available modules, with the following columns:

Name	Type	Description
ID	VARCHAR(255) (Primary Key)	Corresponds to the module's unique name
CONTENT	BLOB (2M)	The module archive

Derby Create Table Command:

```
create table MODULE(NAME VARCHAR(255) not null primary key, CONTENT BLOB(2M));
```

How the DerbyRulebaseResolver works

When an instance of the Oracle Determinations Interview Engine is created, it first loads one of the default RulebaseResolverPlugins. When the plugin resolver discovers a custom RulebaseResolverPlugin (that is, DerbyRulebaseResolver) is available, it uses this to replace the default Rulebase Resolver.

When the initialize method is called, DerbyRulebaseResolver plugin queries the table to retrieve a list of all the rulebases and modules that exist in the database, adds them to a change set and pushes them into the RulebaseService via the applyChangeSet(...) method. The Rulebase Service then loads all the rulebases in the change set ready to be used.

The getInstance method implementation tests that the database is available before returning an initialized instance of the plugin.

Source Code

To view the source code for the RulebaseResolver sample, refer to [examples\interview-engine\rulebase-resolver](#) in the Java runtime zip file.

See also:

[Load rulebases and rule modules in the Determinations Engine](#)