Oracle® Real-Time Decisions

Integration with Oracle RTD

Version 2.2 Document Revised February 2007

Integration with Oracle RTD

Preface

The Oracle Real-Time Decisions (Oracle RTD) platform allows you to develop enterprise software solutions that analyze business process behavior and make recommendations *in real-time,* allowing you to identify and address problems and opportunities as soon as they emerge.

About this document

This guide provides information about integrating with Oracle RTD. It shows several methods of integration, including RTD Smart Clients, Web Services, and directly messaging the Real-Time Decision Server.

Intended audience

This document is intended for developers who will use the Java-based API, .NET component, or Web Services to integrate enterprise applications with Oracle RTD Inline Services. Integrators do not need to have full knowledge of Java, .NET, or Web Services, but they should be familiar with these protocols and understand how they can be used for application integration purposes. Integrators should also be familiar with using Decision Studio and Inline Services.

How to use this guide

This document is divided into the following sections:

Section 1: **Overview** provides an overview of the methods of integration to Oracle RTD.

Section 2: **Using the Java Smart Client** explains how to use the Java Smart Client to integrate to an Inline Service.

Section 3: **Using Java Smart Client JSP tags** explains how to use JSP tags for Java Smart Client integration.

Section 4: **Using the .NET Smart Client** explains how to use the .NET Smart Client to integrate to an Inline Service.

Section 5: **Zero Client Integration** explains the SOAP interface for RTD Decision Service.

Document conventions

Convention	**Description**
`monospace`	Indicates source code and program output.
bold	Indicates portions of the user interface, such as labels, tabs, and menus.
italic	Italics are used to highlight the first use of terms.
‘quote’	Indicates input required from the user.
💡	Indicates additional information that may make the task easier.
📌	Indicates additional information about the subject.

Convention	Description
⚠	Indicates actions that may result in loss of data or errors.

Section 1: Overview

Oracle Real-Time Decisions features several robust and easy-to-use ways to integrate with enterprise operational systems:

- Smart Clients: For Java and .NET environments, these components manage communication to Integration Points on the Real-Time Decision Server.
- Zero Clients: Access to Integration Points is available through Web Services as a zero client approach.

This document outlines how to use these ways to integrate with deployed Inline Services running on Oracle RTD.

For information about using Decision Studio to deploy Inline Services, see *Getting Started with Oracle RTD*. For information about the integration APIs, see the Decision Studio online help.

1.1 Choosing the best means of integration

Oracle Real-Time Decisions offers multiple means of integration. To choose the best means for your environment you should consider the platform you are working on, performance needs and the additional functionality offered by RTD Smart Client over other methods of integration.

1.1.1 About the Java Smart Client

The RTD Smart Client for Java is a component that allows easy, managed integration to deployed Inline Services for operational systems. If you are working in a Java environment, the Java Smart Client is the preferred means of integration. The Java Smart Client offers two important features above and beyond the other methods of integration: session key mapping (to facilitate HTTP session affinity management by an external load balancer) and default response handling.

The factory methods of the Java Smart Client interface take parameters representing the minimal information required to establish contact with a cluster of servers. After connecting, the component's full configuration is downloaded from the server. This way only a small set of parameters must be managed in the client application, while most of the component's configuration is centrally managed by the server's administration console.

The configuration information returned by the server to the client is shared by all the instances of the Smart Client created in the same Java virtual machine. There is a client-side class called a client-side dispatcher that manages this shared configuration and also manages session-affinity information used to dispatch requests to the correct server, based on session keys in the request.

The Java Smart Client is thread-safe, but for optimal performance a separate Java Smart Client should be created for each thread. Separate instances of the Java Smart Client share information and connections, so there is practically no penalty to having multiple instances.

Several factory methods are available to create a Java Smart Client. Most either directly or indirectly reference a properties file in the file system or on a Web server. The properties file supplies addresses for connecting to one or more servers in a single cluster as well as other properties that configure the connection to the server. Factory methods are also available to directly supply an HTTP URL and port or use a default address.

After the client's constructor communicates with one server and receives more complete configuration information, the detailed configuration is saved in a local file called the *client configuration cache*, where it can be accessed should the client restart when the server is unavailable. The configuration cache contains information such as the client's set of default responses for all integration points in all Inline Services. The client's configuration cache is updated automatically by the client whenever it changes in the server.

Part of the configuration information downloaded to a client from the server includes a set of default responses to use if the client loses contact with the server or the server fails to respond to an integration point request in a timely fashion. This maintains the Service Level Agreement (SLA) between the Real-Time Decision Server and client application regardless of individual transactional availability.

These default responses are configured at the granularity of the individual integration points; each integration point relies on its own specialized default response. When any default responses are reconfigured on the server, the changes are propagated automatically to the client's out-of-band data, bundled together with normal integration point responses.

The Java Smart Client automatically keeps track of any HTTP cookies that are returned by the Real-Time Decision Server’s Web Container. The next time the same Inline Service key is used in a request, its cookies are included in the HTTP request so that the external load balancer can route the request to the server instance that is already handling that Inline Service key.

To achieve clustering using other methods of integration, the application must track the Inline Service keys itself.

1.1.2 .NET Smart Client

For the .NET environment, a .NET Smart Client component is available. This component offers a way to call the same interfaces provided by the Java Smart Client. However, it does not offer the added functionality of maintained session affinity or default values.

1.1.3 Web Services

Any client can access the Real-Time Decision Server through Web Services. The benefit to this means of integration is the lack of code needed on the client. Web service operations are defined in a WSDL file and definitions are contained in a schema file.

1.2 About the CrossSell Inline Service

Example Inline Services are included with Decision Studio. One of these is a cross selling example.

The CrossSell Inline Service simulates a simple implementation for a credit card contact center. As calls come into the center, information about the customer and the channel of the contact is captured.

Based on what we know of this customer, a cross selling offer is selected that is extended to the customer. The success or failure of that offer is tracked and sent back to the server so that the underlying decision model has the feedback that helps to refine its ability to make a better cross selling recommendation.

The CrossSell Inline Service is used to demonstrate the various means of integration in this guide.

Several Integration Points are included in the CrossSell example. Use the following instructions to familiarize yourself with these Integration Points.

Informants execute on the server when supplied with the proper parameters. Advisors execute and also return data. In order to supply the correct parameters for calls to Integration Points, we must first identify the *Object IDs*.

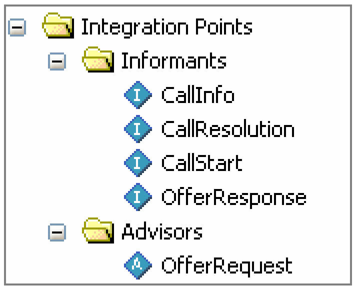
1.2.1 Using Decision Studio to identify Object IDs

1 Open Decision Studio by running `RTD_HOME\eclipse\eclipse.exe`.

2 Select **File** > **Import** to open the CrossSell Inline Service. **Import** appears.

3 Select **Existing Project into Workspace** and click **Next**. Browse for the CrossSell project at the location *RTD_HOME*`\examples\CrossSell`. Select **OK** and click **Finish**, opening the project.

4 Using the Inline Service Explorer, expand **Integration Points**. **Informants** and **Advisors** are listed below. Expand each of these to view the Integration Points. Use the Object ID toggle to show the Object ID in the Inline Service Explorer. When the toggle is highlighted, the Object IDs show in the Explorer; when not, the display label is shown.



The Object ID of the Integration Point may or may not be the same as the label. Object IDs are used to reference Integration Points in method calls.

1.2.2 Determining the Response of an Advisor

Integration Points that deliver responses are called Advisors. An Advisor's **Response** tab in Decision Studio determines the response, by identifying a parameterized Decision object that gets implicitly invoked by the Advisor. The Decision object's responsibility is to select the best Choices from its assigned Choice Group. The choice attributes that are returned are determined by the configuration set on the definition of the Choice Group.

In our example the OfferRequest integration point is an Advisor. It returns a single cross sell offer when it is invoked.

1 In Studio, select the **OfferRequest** Integration Point to view the editor.

2 On the **Response** tab, under **Decision**, look up the Decision that OfferRequest uses to return a response. It should be **OfferDecision**.

3 Double-click **OfferDecision** under **Decisions** to view its detail pane.

4 On the **Selection Criteria** tab, under **Number of Choices to Select**, find the number of responses that OfferRequest provides.

5 On the **Selection Criteria** tab, under **Choice Group**, find the Choice Group that OfferRequest uses. It should be **Offers**.

6 Under **Choices**, double-click **Offers** to see the choice attributes associated with this Choice Group. These attributes will be returned when a call to the Advisor is made.

TIP In Decision Studio, use the **Test** view to call the Advisor and see what is returned. This way, you will see the offer returned and the attributes that come with it. **Test** is available using the tab beside **Problems**. Use the **Execute Request** button to send the request to the server.

1.2.3 Knowing how to respond to the server

Inline Services are most powerful when the success or failure of a Choice is tracked and the model is self learning based on that information. To know what feedback the server needs to be self learning, you must examine the Choice Event Model.

1. In Studio, double-click the **Offer Acceptance** Choice Event Model. The editor will appear on the right.
2. On the **Choice** tab, under **Positive Outcome Events**, you see the Events that the server is interested in for learning. These are:
   - Interested
   - Purchased

   These outcomes are to be reported to the server from your Inline Service to give the proper feedback to the model.
3. The **OfferResponse** Integration Point is responsible for reporting this information.

1.2.4 Identifying session keys and arguments

To invoke an Integration Point, we must supply values for the session keys and arguments expected by the Integration Point. In the request, we must use the Object IDs defined by Decision Studio for the Integration Point's session keys and arguments. The key name must match one of the session key names defined in Decision Studio for the Integration Point.

1. Select the **CallStart** Integration Point. On the **Request** tab of the editor of the Integration Point, under the **Session Keys** list, a path to the session key is shown starting with `session`; the last name in the path is the Object ID of the session key.

   > Note: If the session key is not displayed in object format, use the **Object ID toggle** to change the display settings. Only the final object ID is necessary for the session key. For example, in the case shown above, only the final string, `customerId`, is used.

2. To identify the arguments of the Integration Point, use the detail pane of to view the **Incoming Attribute** column of the **Request** tab. The **CallStart** incoming argument is **channel**.

Section 2: Using the Java Smart Client

This section introduces using the Java Smart Client for integration. An example is included with Oracle RTD installation.

2.1 Before you get started

You must perform the following tasks first before you can work with the Java Smart Client example:

1. Install a Java Development Kit (JDK), with the `JAVA_HOME` environment variable set to its location. To obtain a JDK, go to the Sun Microsystems Web site, http://java.sun.com/products/.

2. Install the Oracle RTD files and deploy Oracle RTD to an application server. See *Installation and Administration of Oracle RTD* for full information.

3. The Java Smart Client example works with the sample CrossSell Inline Service. Because of this, you must first populate the Oracle RTD Database with the CrossSell example data, then deploy the CrossSell Inline Service using Decision Studio. For information about populating the Oracle RTD Database with the CrossSell example data, see *Installation and Administration of Oracle RTD*. For information about deploying Inline Services, see *Decision Studio Reference Guide*.

4. Start the Real-Time Decision Server. For more information, see *Installation and Administration of Oracle RTD.*

2.2 Integrating with an Inline Service using the Java Smart Client

In general, integration using the Java Smart Client includes the following steps:

1. Prepare a properties file.

2. Create a connection to the Inline Service.

3. Create a request that identifies the Integration Point to connect to and the parameters to identify the session and any other information the Integration Point needs to determine an outcome.

4. Invoke the request.

5. Gather and parse any response information from Advisors.

6. Close the connection.

2.2.1 Java Smart Client API Reference

For full information about the Java Smart Client API, see the Decision Studio online help.

2.2.2 Preparing the Java Smart Client example

For this example, the CrossSell Inline Service has been integrated to a simple command-line application to demonstrate how to use the Java Smart Client for integration.

Use the following steps to prepare the Smart Client example.

1 Locate the file *RTD_HOME*`\client\Client Examples\Java Client Example\lib\sdbootstrap.properties` and open it for editing. Comment out all properties except for `client=true`, as follows:

```
client=true

#StudioStaticFilesLocation=shared_ui/studio

#WebServerLocation=http://localhost:8080

#WorkbenchServlet=/ui/workbench
```

Then, save and close the file.

2 Open Decision Studio and choose File > Import , then select Existing Projects into Workspace and click Next.

3 For Select root directory, browse to *RTD_HOME*`\client\Client Examples\ Java Client Example` and click OK. Then, click Finish.

4 From the menu bar, select Window > Open Perspective > Java. If the Console view is not visible, select Window > Show View > Console.

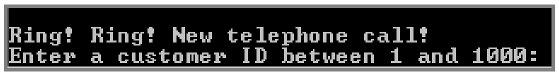
5 From the menu bar, select Run > Run.

6 In the Create, manage, and run configurations screen, select Java Application and click New.

7 Click Browse next to the Project field, then select JavaSmartClientExample and click OK.

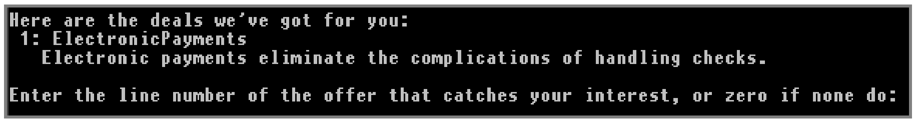
8 Click Search next to the Main class field, then select Example and click OK.

9 Click Apply, then click Run. In the Console view, the following text appears:



```
Ring! Ring! New telephone call!
Enter a customer ID between 1 and 1000:
```

10 Place the cursor after the colon, then enter a customer ID (such as 5) and press Enter. The response appears similar to the following:



```
Here are the deals we've got for you:
 1: ElectronicPayments
   Electronic payments eliminate the complications of handling checks.

Enter the line number of the offer that catches your interest, or zero if none do:
```

11 Place the cursor after the final colon, then enter 1 to select the offer. The server responds with a final message.

12 The process repeats. Enter a customer ID greater than 1000 to stop the program.

You can find the source code for this example in the file *RTD_HOME*`\client\Client Examples\Java Client Example\src\com\sigmadynamics\client\example\Example.java`. The example is explained below.

2.2.3 About Java Smart Client properties

When a client application creates a Java Smart Client, it passes a set of properties to a Java Smart Client factory that represents the component's endpoint configuration. This file contains just enough information to allow the client to connect to a server endpoint. There are additional factory methods that use default configuration values; however it is best to explicitly specify the properties. The default properties file is shown below.

The factory method uses the properties to connect to the server. When the factory connects to the server, it downloads the more complete configuration information to the client, such as the set of default responses that the client should use if it ever needs to run when the server is unavailable. The detailed client configuration is saved in a local file, the Java Smart Client configuration cache, and is updated automatically whenever the server's configuration changes.

2.2.4 Creating the properties file

1 Locate the file *`RTD_HOME`*`\client\Client Examples\Java Client Example\lib\sdclient.properties` and open it for editing. The file should appear as follows:

```
UseEndpointsInOrder = HTTP1

appsCacheDirectory = ${rootDir}/etc

timeout = 0

HTTP1.type = http

HTTP1.url = http://localhost:8080/
```

2 Modify the contents to match your server configuration. Explanations of the elements of this file are listed in the table below. In particular, make sure that you have a valid cache directory and the endpoint URL is the URL and port of your local Real-Time Decision Server. By default, this is `http://localhost:8080`.

Element	Description
`UseEndpointsInOrder`	A comma-separated list of endpoint names, indicating the order in which the endpoints should be tried when establishing an initial connection to the server cluster during the Smart Client's initialization. After initialization, this list of endpoints is irrelevant because the server will supply an updated list of endpoints. The endpoint names in this list refer to definitions within this properties file; the names are not used elsewhere.
`appsCacheDirectory`	A file URL identifying a writable directory into which the client component may save the configuration information that it gets from the server. The cache provides insurance against the possibility that the Real-Time Decision Server might be unavailable to the client application when the application initializes its client components. If `sdclient.properties` specifies a cache directory, it must already exist, otherwise, the client will use the Java virtual machine's temp directory
`timeout`	The timeout, in milliseconds, used by the original attempt to contact the server during the client component's initialization. After connecting to the server, the client uses the server's timeout, configured through JMX MBean property, `EntryPointRequestTimeout`.
`<endpoint_name>.type`	The named endpoint type. Only HTTP is supported at this time.
`<endpointName>.url`	A URL specifying the HTTP host and port of the server's HTTP endpoint. The default endpoint is http://localhost:8080.

2.2.5 Creating the Java Smart Client

1 Open the source file for the Example application at *RTD_HOME*`\client\Client Examples\Java Client Example\src\com\sigmadynamics\client\example\Example.java`.

> TIP This example source code can be used as a template for your Java Smart Client implementation.

2 The following imports are used to support Oracle RTD integration:

```
import com.sigmadynamics.client.IntegrationPointRequestInterface;

import
com.sigmadynamics.client.IntegrationPointResponseInterface;

import com.sigmadynamics.client.ResponseItemInterface;

import com.sigmadynamics.client.SDClientException;

import com.sigmadynamics.client.SDClientFactory;

import com.sigmadynamics.client.SDClientInterface;
```

3 In the main method, the Example application demonstrates several techniques for using SDClientFactory to create an implementation of `SDClientInterface`, based on the arguments supplied to the Example application.

These arguments are passed to `getClient`, where the proper factory method is identified.

```
SDClientInterface client = getClient(args);
```

There are several factory methods used to create a Java Smart Client. By examining `getClient`, we see the various methods:

```
private static SDClientInterface getClient(String[] args ){
    try{
      if ( args.length == 0 )
        return getClientWithDefaultPropertiesFile();
```

> Creates a Java Smart Client with the default properties file using `create(java.lang.String)`. The default properties file is referenced above.

```
      if ( "-h".equals(args[0])){
        if ( args.length < 2 )
          return getClientWithDefaultHttpAddress();
```

> Creates a Java Smart Client with the default HTTP address of `http://localhost:8080`. This is the default installation URL and port of the Real-Time Decision Server. Uses `createHttp(java.lang.String, int, boolean)`.

```
        return getClientWithHttpAddress( args[1]);
      }
```

Creates a Java Smart Client with a supplied HTTP address. This is the address and port of your Real-Time Decision Server, if it is not at the default address. Uses `createHttp(String)`.

```
      if ( "-u".equals(args[0])){
        if ( args.length < 2 )
        {
          System.out.println("Missing properties file URL
          argument" );
          System.exit(-1);
        }
        return getClientWithPropertiesFileURL( args[1] );
      }
```

Creates a Java Smart Client with the information supplied in the properties file at the address specified. Uses `createFromProperties`.

```
      if ( "-f".equals(args[0])){
        if ( args.length < 2 )
        {
          System.out.println("Missing properties filename
argument" );
          System.exit(-1);
        }
        return getClientWithPropertiesFileName( args[1] );
      }
```

Creates a Java Smart Client with the information supplied in the properties file. Uses `createFromPropertiesURL`.

```
      System.out.println("Unrecognized argument");
    }catch (SDClientException e ){
      e.printStackTrace();
    }
    System.exit(-1);
    return null;
  }
```

These methods are summarized in the Java Smart Client API section of the Decision Studio online help.

2.2.6 Creating the request

1 Next, the client application creates a request to send to the Real-Time Decision Server.

`SDClientInterface` is used to create a request object:

```
createRequest(String appName, String integrationPointName);
```

> The `appName` parameter is the name of a server-resident application, developed in Decision Studio.
>
> The `integrationPointName` parameter is the name of the application's Informant or Advisor that is to receive the request.
>
> See Section 1.2.1, Using Decision Studio to identify Object IDs to locate these values.

In our example, the request is created here:

```
IntegrationPointRequestInterface request  =
client.createRequest(INLINE_SERVICE_NAME, "CallStart");
```

2 The request object provides a method to set a single session key; call it separately for each key.

```
void setSessionKey(String keyName, String keyValue);
```

> By example, if an Integration Point's session key is listed in Decision Studio as **session.customer.customerId**, you would pass **customerId** as the key name to `setSessionKey`.
>
> See Section 1.2.4, Identifying session keys and arguments to locate these values.

In the example application the request session key is populated using:

```
request.setSessionKey( SESSION_KEY, sCustID );
```

where `SESSION_KEY` was set

```
static final String SESSION_KEY = "customerId";
```

and `sCustID` was captured from the command-line input.

3 The request object provides two methods to set a single argument; call the appropriate one separately for each argument. The first method accepts an argument having a string value. The second method accepts an array of string values.

```
void setArg(String argName, String argValue);

void setArg(String argName, String[] argValue);
```

> The argument name should match one of the input names listed in Decision Studio for the Integration Point.

See Section 1.2.4, Identifying session keys and arguments to locate these values. The value of this argument should be determined from the application design.

In the example application, the request is populated using:

```
request.setArg( "channel", "Call");
```

2.2.7 Invoking the request

1. After populating the request, the client application calls the `invoke` method of `SDClientInterface` to send the request to the server and receives an `IntegrationPointResponseInterface` representing an array of choices calculated by the server.

```
IntegrationPointResponseInterface
invoke(IntegrationPointRequestInterface request);
```

In the example application, this call is made:

```
client.invoke(request);
```

Note: If the client application wants to send a request for which it does not expect a response, and for which message delivery sequence is not critical, it can use the `invokeAsync` method instead of `invoke`.

Requests sent through `invokeAsync` are not guaranteed to arrive at the server before requests sent through subsequent `invokeAsync` or `invoke` calls. When message delivery sequence is important, the `invoke` method should be used instead of `invokeAsync`, even when no response is expected.

2. After the request to the CallStart Integration Point is invoked, a new request is prepared and invoked for CallInfo.

```
// Supply some additional information about the telephone call.

 // Apparently the CrossSell service expects very little here --

 // just the channel again, which it already knows. Hence this
message

 // could be left out with no consequences.

 request = client.createRequest(INLINE_SERVICE_NAME, "CallInfo");

 request.setSessionKey( SESSION_KEY, sCustID );

 request.setArg( "channel", "Call");

 client.invoke(request);
```

2.2.8 Examining the response

When an Advisor is invoked, a number response items, also known as Choices, will be returned. Your application must be prepared to handle this number of response items. See Section 1.2.2, Determining the Response of an Advisor for more information.

In the client application, the selected Choices are accessible through the `IntegrationPointResponseInterface` returned by the invoke method. The `IntegrationPointResponseInterface` provides access to an array of response item objects, `ResponseItemInterface`, where each response item corresponds to a Choice object selected by the Advisor's Decision.

The package `com.sigmadynamics.client` surfaces a Choice as a collection of value strings, keyed by name string.

In our example, the response is examined as such:

1 When invoking a request on an Advisor Integration Point, be prepared to receive a response.

```
// Based on what the server knows about this customer, ask for
some

// product recommendations.

request = client.createRequest(INLINE_SERVICE_NAME,
"OfferRequest");

request.setSessionKey( SESSION_KEY, sCustID );

IntegrationPointResponseInterface response =
client.invoke(request);
```

2 Knowing the number of responses expected allows you handle them accurately. The responses are read from the array and displayed to the customer.

```
if ( response.size() > 0 ){
// Since I know that CrossSell's OfferDecision returns only
// one Choice, I could get that choice from the response with
// response.get(0); Instead, I'll pretend that
// multiple offers could be returned instead of just one.
         System.out.println();
         System.out.println("Here are the deals we've got for you:");
         ResponseItemInterface[] items = response.getResponseItems();
         for ( int i = 0; i < items.length; i++ ){
           System.out.println(" " + (i+1) + ": " + items[i].getId());
           String message = items[i].getValue("message");
           if ( message != null )
             System.out.println("   " + message );
         }
         System.out.println();
         System.out.println("Enter the line number of the offer
         that catches your interest, or zero if none do: " );
```

2.2.9 Closing the loop

Many Inline Services are designed to be self learning. In the CrossSell Inline Service, the OfferResponse Informant reports interest in a cross sell offer back to a Choice Event model.

```
// Tell the server the good news.
request = client.createRequest(INLINE_SERVICE_NAME, "OfferResponse");
request.setSessionKey( SESSION_KEY, sCustID );
request.setArg( "choiceName", prodName );


// "Interested" is one of the Choice Events defined for the choice
group, Offers.
```

To identify the Choice Event model and Choices, see Section 1.2.3, Knowing how to respond to the server.

```
request.setArg( "choiceOutcome", "Interested" );
client.invoke(request);
```

Finally, the session is closed by invoking the CallResolution Informant in the server, which in the CrossSell example has been designed to terminate the session.

```
// Close the server's session.

request = client.createRequest(INLINE_SERVICE_NAME, "CallResolution");
request.setSessionKey( SESSION_KEY, sCustID );
client.invoke(request);
```

2.2.10 Closing the client

When the client application is finished using its `SDClientInterface`, and doesn't intend to use it again, it calls the component's close method, to release any instance-specific information.

```
client.close();
```

Section 3: Using Java Smart Client JSP tags

A convenient way to integrate a Web application with a deployed Inline Service is to use the JSP client integration tags. JSP allows you to generate interactive Web pages that use embedded Java. The JSP tags provided are based on the Java Smart Client discussed in the previous section.

There is negligible overhead when using the JSP tags. In addition, the tags incorporate automatic reuse of Smart Clients for same session to enhance performance. When a Java Smart Client is created using the JSP tag, a check is performed to see if a client already exists with the same name and properties and has not been closed. If it does, it automatically reuses that client; if not it will create a new one.

3.1 Before you get started

You must perform the following tasks first before you can work with the JSP client integration tags:

1. Install a Java Development Kit (JDK), with the *`JAVA_HOME`* environment variable set to its location. To obtain a JDK, go to the Sun Microsystems Web site, http://java.sun.com/products/.

2. Install the Oracle RTD files and deploy Oracle RTD to an application server. See *Installation and Administration of Oracle RTD* for full information.

3. The Java Smart Client example works with the sample CrossSell Inline Service. Because of this, you must first populate the Oracle RTD Database with the CrossSell example data, then deploy the CrossSell Inline Service using Decision Studio. For information about populating the Oracle RTD Database with the CrossSell example data, see *Installation and Administration of Oracle RTD*. For information about deploying Inline Services, see *Decision Studio Reference Guide*.

4. Start the Real-Time Decision Server. For more information, see *Installation and Administration of Oracle RTD.*

3.2 Integrating with an Inline Service using Java Smart Client JSP tags

In general, integration using the Java Smart Client includes the following steps:

1. Prepare a properties file.

2. Use an Invoke or AsyncInvoke tag to create a request to the server.

3. Gather and parse any response information from Advisors.

4. Close the connection.

3.2.1 JSP Smart Client Tag Reference

For full information about the JSP Smart Client tags, see the Decision Studio online help.

3.2.2 Sample JSP code

A working example of using the Smart Client JSP tags for integration can be found at *`RTD_HOME`*`\client\Client Examples\JSP Client Example\example.jsp`.

3.2.3 Deploying the JSP Smart Client example

For this example, the CrossSell Inline Service has been integrated to a simple command-line application to demonstrate how to use the Smart Client for integration. You need to deploy the JSP Smart Client example to your application server, as described in the following sections.

3.2.3.1 Deploying the JSP Smart Client example to OC4J

1 Log in to the Application Server Control as the `oc4jadmin` user. You can access the Application Server Control at `http://`*`oc4j_host`*`:`*`port`*`/em`. For the standalone version of OC4J, the port number is typically 8888.

2 On the OC4J home page, click the **Applications** tab.

  Note: If you are using OC4J as part of Oracle Application Server, first click **home** under the Groups heading, then proceed to the **Applications** tab.

3 Click **Deploy**. On the Deploy: Select Archive page, under the Archive heading, browse to specify the archive location *`RTD_HOME`*`/client/Client Examples/JSP Client Example/sdclient-test.war`. Then, click **Next**.

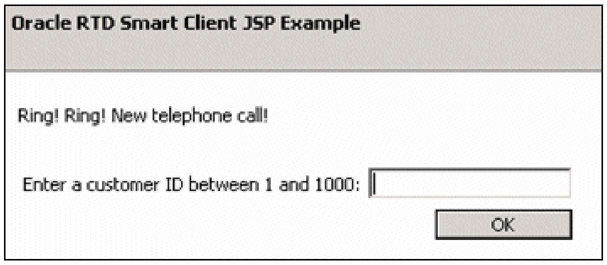
4 On the Deploy: Application Attributes page, enter `JSPClientExample` for **Application Name**, then choose `rtd-web-site` for **Bind Web Module to Site**. Then, click **Next**.

5 On the Deploy: Deployment Settings page, click **Deploy**.

6 To access the application, open a Web browser and go to:

  `http://`*`ocj4_host`*`:8080/sdclient-test/example.jsp`

  A Web page appears that simulates a service call. For example:

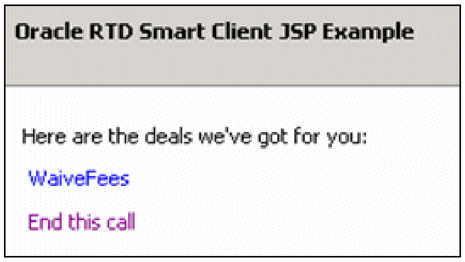


Oracle RTD Smart Client JSP Example

Ring! Ring! New telephone call!

Enter a customer ID between 1 and 1000:

7 Enter a customer ID (such as 5) and click **OK**. A response page appears, displaying an offer and an option to end the call:



Oracle RTD Smart Client JSP Example

Here are the deals we've got for you:

WaiveFees

End this call

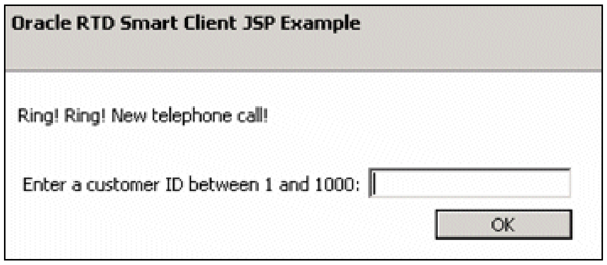
8 Click the offer link, or click **End this call**.

3.2.3.2 Deploying the JSP Smart Client example to WebSphere

1. Access the Integrated Solutions Console at the URL `http://`*`websphere_host`*`:`*`port`*`/ibm/console`. At the login prompt, enter the administrator user name and password. On Windows, you can also access the Integrated Solutions Console through **Start > Programs**.
2. In the tree on the left, expand **Applications**, then choose **Enterprise Applications.**
3. Click **Install.**
4. In the Path to the new application section, enter or browse to the path *`RTD_HOME`*`/client/Client Examples/JSP Client Example/sdclient-test.war`.
5. For **Context root**, enter `sdclient-test`.
6. Click **Next**, then click **Next** again, then click **Next** again.
7. Click **Finish**, then click **Save.**
8. On the Enterprise Applications page, select the **sdclient-test** application and click **Start.**
9. To access the application, open a Web browser and go to:

   `http://`*`websphere_host`*`:8080/sdclient-test/example.jsp`

   A Web page appears that simulates a service call. For example:

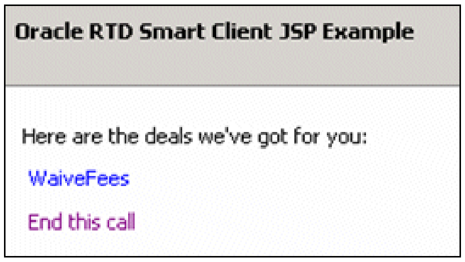


Oracle RTD Smart Client JSP Example

Ring! Ring! New telephone call!

Enter a customer ID between 1 and 1000:

10. Enter a customer ID (such as 5) and click **OK.** A response page appears, displaying an offer and an option to end the call:



Oracle RTD Smart Client JSP Example

Here are the deals we've got for you:

11. Click the offer link, or click **End this call.**

3.2.3.3 Deploying the JSP Smart Client example to WebLogic

1. Access the WebLogic Server Administration Console for your Oracle RTD domain at the URL `http://`*`weblogic_host`*`:`*`port`*`/console`. At the login prompt, enter the administrator user name and password. On Windows, you can also access the WebLogic Server Administration Console through **Start > Programs > BEA Products > User Projects >** ***domain_name*** **> Admin Server Console.**
2. In the tree on the left, click **Deployments.**

3 Click **Install**. You may need to click **Lock & Edit** first to enable the **Install** button.

4 Go to *RTD_HOME*/client/Client Examples and select **JSP Client Example**, then click **Next**.

5 Select **Install this deployment as an application**, then click **Next**.

6 On the Optional Settings page, enter `JSPClientExample` for **Name**. Then, click **Next**.

7 Review your settings and click **Finish**.

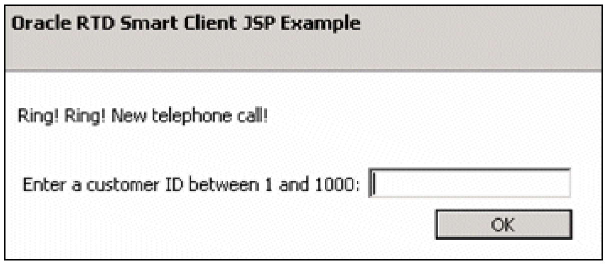
8 Click **Save**, then click **Activate Changes**.

9 Start the application by selecting **JSPClientExample** application in the **Deployments** table, then clicking **Start > Servicing all Requests**. When prompted, click **Yes**. The application is now running.

10 To access the application, open a Web browser and go to:

`http://`*`weblogic_host`*`:8080/JSP%20Client%20Example/example.jsp`

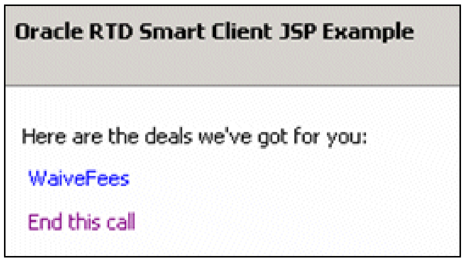
A Web page appears that simulates a service call. For example:


Oracle RTD Smart Client JSP Example

Ring! Ring! New telephone call!

Enter a customer ID between 1 and 1000:

11 Enter a customer ID (such as 5) and click **OK**. A response page appears, displaying an offer and an option to end the call:


Oracle RTD Smart Client JSP Example

Here are the deals we've got for you:

WaiveFees

End this call

12 Click the offer link, or click **End this call**.

Section 4: Using the .NET Smart Client

The .NET Smart Client provides a very similar client to the Java API to make calls from your application. With the current implementation, the .NET Client does not have some of the advanced features of the Java Smart Client, including session affinity management and default response handling..

The .NET Smart Client is located at *`RTD_HOME`*`\client\Client Examples\Dot Net Client Example\sdclient.dll`. This file should be co-located with your application in order to be accessible.

4.1 Before you get started

You must perform the following tasks first before you can work with the .Net Smart Client:

5. Install a Java Development Kit (JDK), with the *`JAVA_HOME`* environment variable set to its location. To obtain a JDK, go to the Sun Microsystems Web site, http://java.sun.com/products/.

6. Install the Oracle RTD files and deploy Oracle RTD to an application server. See *Installation and Administration of Oracle RTD* for full information.

7. The .NET Smart Client example works with the sample CrossSell Inline Service. Because of this, you must first populate the Oracle RTD Database with the CrossSell example data, then deploy the CrossSell Inline Service using Decision Studio. For information about populating the Oracle RTD Database with the CrossSell example data, see *Installation and Administration of Oracle RTD*. For information about deploying Inline Services, see *Decision Studio Reference Guide*.

8. Start the Real-Time Decision Server. For more information, see *Installation and Administration of Oracle RTD.*

4.2 Integrating with an Inline Service using the .NET Smart Client

The following example outlines how to use the .NET Smart Client for integration to deployed Inline Services on Oracle RTD.

In general, the following are the steps for integration:

1. Create the RTD Smart Client within your application code.
2. Create a request directed at an Inline Service and an Integration Point.
3. Populate the request with arguments and session keys.
4. Invoke the request using the Smart Client.
5. If the request is invoked on an Advisor, examine the response.
6. Close the Smart Client when finished.

4.2.1 .NET API Reference

For full information about the .NET Smart Client API, see the Decision Studio online help.

4.2.2 .NET integration example

You can find an example of a .NET integration client in `*RTD_HOME*\client\Client Examples\Dot Net Client Example\DotNetSmartClientExample.sln`. You can open this example in Microsoft Visual Studio, then run or debug the example.

In this example, Informant and Advisor Integration Points are invoked on the CrossSell Inline Service. To familiarize yourself with this Inline Service, see Section 1.2, About the CrossSell Inline Service. In the example, the Integration Points are invoked and the return values from the Advisor are written to the console.

Follow these steps to run the example in Microsoft Visual Studio:

1 Start Microsoft Visual Studio .NET.

2 From the menu bar, select File > Open > Project.

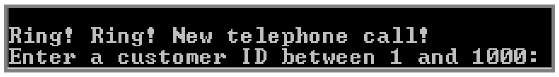
3 For File Name, select *RTD_HOME*`\client\Client Examples\Dot NET Client Example\DotNetSmartClientExample.sln` and click Open.

4 If Real-Time Decision Server is running on a different computer, in the right-hand Solution Explorer window, double-click DotNetSmartClientExample.cs. Locate the following line:

```
SDClient client = new SDClient("http://localhost:8080");
```

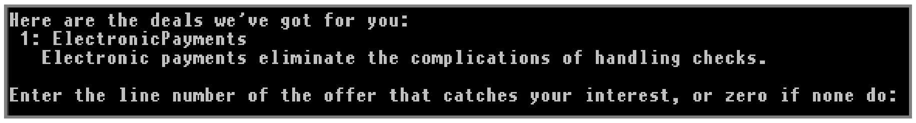
Change `localhost` to the name of the computer where Real-Time Decision Server is running. Then, save and close the file.

5 From the menu bar, select Debug > Start. In the Console window, the following text appears:



```
Ring! Ring! New telephone call!
Enter a customer ID between 1 and 1000:
```

6 Place the cursor after the colon, then enter a customer ID (such as 5) and press Enter. The response appears similar to the following:



```
Here are the deals we've got for you:
 1: ElectronicPayments
   Electronic payments eliminate the complications of handling checks.

Enter the line number of the offer that catches your interest, or zero if none do:
```

7 Place the cursor after the final colon, then enter 1 to select the offer. The server responds with a final message.

8 The process repeats. Press Enter at the Customer ID prompt, without entering a number, to stop the program.

Section 5: Zero Client Integration

Real-Time Decision Server Integration Points are available through a Zero Client approach. Integration Points on a deployed Inline Service are exposed through a Web Services definition.

It is recommended that you work through the tutorial outlined in Section 2: Using the Java Smart Client to understand the process of invoking Integration Points.

5.1 About Web Services

The ability to invoke and asynchronously invoke a deployed Integration Point is exposed as a Web Service by the Real-Time Decision Server. The definition of these operations are available in a WSDL file, located at `RTD_HOME\deploy\DecisionService\DecisionService.wsdl`. The WSDL file defines all complex types and operations available.

Some slight structural changes were introduced in Version 2.2 to bring the Decision Service up to the WS-I Basic level of compliance. The previous version of the WSDL file is named `RTD_HOME\deploy\DecisionService\DecisionServiceLegacy.wsdl`. Although implementors should develop new clients using the new WSDL, the server still understands the protocol defined by `DecisionServiceLegacy.wsdl`, and existing clients should experience no loss of functionality.