Oracle® Real-Time Decisions

Decision Studio
Reference Guide

Version 2.2 Document Revised February 2007

Decision Studio Reference Guide

Preface

Decision Studio is a tool used to define and manage Inline Services. All aspects of Inline Services are exposed in Decision Studio. The target user of Decision Studio is an IT professional with a basic knowledge of Java and a general understanding of application development and lifecycle issues. For more information on using Decision Studio, see *Getting Started with Oracle RTD*.

Decision Studio is a rich-client application that follows an integrated development environment (IDE) paradigm. Decision Studio makes use of an Inline Service Explorer view on the left, and an editor view on the right. The navigator view displays a predefined Inline Service folder structure. Items within each folder are Inline Service metadata elements. Using Decision Studio, metadata elements may be added, edited, and deleted. When a metadata element is double-clicked, the details of the element are shown in the object editor. Each metadata element type has its own editor. The elements are first represented as XML metadata, and then later, Java classes are generated from which the running Inline Service is compiled.

Decision Studio is based on the Eclipse IDE. It combines features that are specific to managing Inline Services with the features of the Eclipse IDE, which include general purpose Java development tools, integration with Software Configuration Management (SCM) systems, and so on.

This guide provides an in-depth look at the concepts, components, and APIs needed to use Decision Studio to develop Inline Services.

About this document

This document acts as a reference to Decision Studio. It identifies each of the elements used to configure Inline Services, including the properties of each Inline Service and the available APIs.

Intended Audience

This document is designed for technical users configuring Inline Services using Decision Studio. Users should have a basic knowledge of Java and the software development lifecycle.

How to use this guide

This document is divided into the following sections:

Section 1: About Decision Studio gives an overview of the platform.

Section 2: About Decision Studio Elements and APIs details each element and the callable APIs of the system.

Section 3: Oracle RTD General APIs serves as a reference for the general APIs of Oracle RTD.

Section 4: Deploying and Testing an Inline Service outlines testing functionality, as well as deploying and redeploying to the Real-Time Decision Server.

Section 5: Troubleshooting and Debugging Inline Services provides tips for debugging your Inline 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.

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

Section 1: About Decision Studio

This section provides an overview of Decision Studio.

This section contains the following topics:

- 1.1 About Inline Services
- 1.2 Decision Studio and Eclipse
- 1.3 About Decision Studio projects
- 1.4 Directory structure of Inline Services
- 1.5 Configuring Inline Services

1.1 About Inline Services

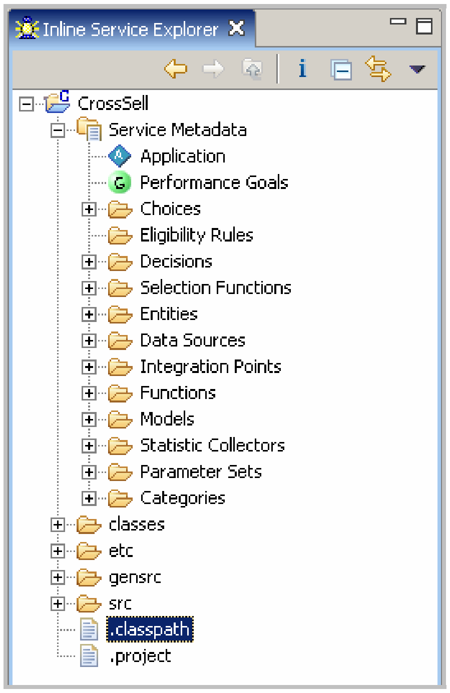
An Inline Service is a deployable application that monitors and advises business processes at key points across the enterprise on a real-time and continuous basis. Inline Services do not follow business processes from end-to-end, but rather focus on specific and identified points within the process. Inline Services are configured and deployed using Decision Studio and analyzed and tuned using Decision Center. Inline Services run on Real-Time Decision Server. Together these components comprise Oracle RTD.

1.2 Decision Studio and Eclipse

Decision Studio is based on Eclipse, an open source Java IDE produced by the Eclipse Foundation. Decision Studio exists as a standard plug-in to the Eclipse environment. If you are using Eclipse, you have the advantage of using the environment for additional development and advanced features. If you are not familiar with Eclipse, it is completely transparent to using Studio.

1.2.1 About the Inline Service Explorer

The Inline Service Explorer gives you access to all aspects of your Inline Service projects. A typical Inline service project is shown below.



The folders contain the following:

Service Metadata	The metadata that forms the Inline Service. The default editor for this type of file is the editor specific to each element. You can also edit metadata in a text editor, however, this is not recommended.
classes	The classes generated by the compile process.
etc	This directory contains various scripts and files which are used for system administration. If a load generator script is built, it is kept in this folder by convention.
gensrc	The generated source code files for the Inline Service.
src	Custom Java code, which may include arbitrary user-provided Java classes. Some of these classes can be used to override the default behavior of the generated Inline Service Java classes.
lib	Optionally, a `lib` folder is created by the user when using outside classes. For instance, assume you want to access a class called `THashMap` in some function, logic, or initialization block. This class exists in the `tcollections.jar` file. To use the class, create the `lib` folder under the project directory in the project workspace, and then put the `tcollections.jar` file in the folder. To use a class from this jar, import using the Advanced button next to description and then use the class in your code.
.classpath	The file containing the Java classpath for the project. There is no need to edit this file.
.project	The Eclipse project file.

1.2.2 Code Generation

In general, as elements are configured for an Inline Service, four files are produced:

- An .sda file that stores the configuration as metadata.
- A .java file that is generated from the metadata and is compiled into a class file.
- A .java file that extends the original generated file and can be used in unusual circumstances to override the actions of the generated file.
- The class file that is first compiled from the generated file and subsequently compiled from any overrides.

The files are named in the following fashion:

- Metadata: *Object_ID*.sda
- Generated: GEN*Object_ID*.java
- Override: *Object_ID*.java
- Class: *Object_ID*.class
- Generated Class: GEN*Object_ID*.class

> Tip: The Object ID is created as you name the object. Object IDs are shown in the text box below the name of the object for which they are created. The Object ID may be different from the label to conform to Java standards. To look up an Object ID, toggle between showing labels and showing Object IDs using the ID toolbar button.

For instance, consider an element named Customer account. An Object ID is formed, CustomerAccount, that conforms to Java naming standards.

The files created are:

- Metadata: `CustomerAccount.sda`
- Generated: `GENCustomerAccount.java`
- Override: `CustomerAccount.java`
- Class: `CustomerAccount.class`
- Generated Class: `GENCustomerAccount.class`

1.2.3 About Decision Studio perspectives and views

Decision Studio lets you work with an Inline Service from several *perspectives*. A perspective defines the initial set and layout of *views* and *editors* for the perspective. Each perspective provides a set of functionality aimed at accomplishing a specific type of task, or works with specific types of resources. Perspectives control what appears in certain menus and toolbars.

The default Inline Service perspective contains four views:

- Inline Service Explorer view: shows the project and elements in tree form; by default, it is located on the left-hand side of the screen.

- Problem view: shows errors and exceptions with your project; by default, it is located at the bottom of the screen as a tabbed selection, along with Test view.
- Test view: provides an area for testing your Inline Service; by default, it is located at the bottom of the screen as a tabbed selection, along with Problem view.
- Cheat Sheets view: provides step-by-step instructions for common tasks; by default, it is located on the right-hand side of the screen.

The center area of the Inline Service perspective is the *editor area,* and shows an editor that is specific to the node on the project tree you have selected. To change to a new editor, double-click the element you want to edit.

To edit a Java file, change to the Java perspective and double-click the Java file you want to edit.

The Inline Service perspective is the default perspective, and is the main work area for configuring and deploying Inline Services. Oracle RTD has a number of features for working with Inline Service metadata. These are documented in the following sections. If there is a feature you do not see here, it is part of the core Eclipse platform. For information about these features, see the Eclipse online help document *Workbench User Guide*.

The Inline Service perspective has the following menu and toolbar items.

File > New Inline Service Project		Creates a new Inline Service project in the workspace you choose.
Project > Download		Downloads an already deployed Inline Service from a Real-Time Decision Server to make changes.
Project > Deploy		Deploys an Inline Service to a Real-Time Decision Server.
Window > Open Perspective > Inline Service Perspective		Opens an Inline Service perspective.
Window > Show View > Inline Service Explorer View		Shows the current Inline Service View.
Window > Dispay Object IDs		Toggles between showing labels and Object IDs.
Help > About		Displays version information about Decision Studio.
	Deploy	Deploys an Inline Service to a Real-Time Decision Server.
	Download	Downloads an already deployed Inline Service from a Real-Time Decision Server to make changes.
ID	Object ID toggle	Toggles between showing labels and Object IDs.

The Inline Service Explorer View has the following toolbar items.

	Metadata toggle	Toggles between showing the entire project tree, or just Inline Service metadata.
	Collapse All	Collapses the project tree.
	Link with editor	Finds the proper editor for the selected element type and links so that when you select an element, the editor adjusts accordingly.

▼	Menu	Provides access to **Link with editor** and **Always Show Object IDs**.
	Always Show Object IDs	Shows both the Object ID and label of elements in the Inline Service Explorer and the editors.

The Java perspective combines views that you would commonly use while editing Java source files, while the Debug perspective contains the views that you would use while debugging Java programs.

To work directly with the generated Java code, use the Java perspective. To debug an Inline Service at the Java code level, use the Debug perspective.

1.2.4 Arranging views and resizing editors

Tabs in the editor area indicate the names of resources that are currently open for editing. An asterisk (*) indicates that an editor has unsaved changes. Tabs on views indicate the name of the view, and have a toolbar that provides functionality specific to that view.

You may drag and drop the views and editors of a perspective to any space on the screen. Views and editors will re-size themselves to fit the area in which they are placed. Occasionally portions of an editor (where you do your main work) or view will become covered by other views, or resized to an area that is not convenient to use. To resize the editor or view, either close some other open views and the remaining will automatically resize, or maximize the editor or view.

Both editors and views can be toggled between Maximize and Minimize by double-clicking the tab, or by using the right-click menu item. For more information on perspectives, editors, and views, see the online documentation provided in the *Workbench User Guide*, contained in the Eclipse online help.

1.2.5 About element logic

Java code is added to the logic panels of elements within Decision Studio. This code is then inserted into the proper methods of the `GEN`*`Object_ID`*`.java` file. To add logic to an element, or to update it, select the element, and use the editor to change the code in the logic panel.

Sometimes it is more convenient to insert larger code fragments directly within the generated code. You may edit these files directly through the Java perspective of Decision Studio. It is very important to note that the generated code can only be manually edited in specific places. Also, note that when you choose **Project** > **Clean**, Decision Studio regenerates the generated code, overwriting any code changes made directly to the generated Java code.

Any method that can be edited through the Java perspective in Decision Studio is clearly marked with a Start and End marker. For instance, the Application object has a method to initialize the Inline Service, `init()`.

Code for this method can be added through the Decision Studio interface, using the **Initialization Logic** panel on the **Logic** tab of the Application element.

If you choose, instead, to add your initialization code directly into the Application class using Eclipse, add it only to the method marked as such:

```
public void init() {

    // SDCUSTOMCODESTART.Application.InitBody.java.0

    // SDCUSTOMCODEEND.Application.InitBody.java.0

  }
```

Your code must fall between the start and end comments of the method. Any code that falls outside of the commented areas risks being overwritten. The code added directly to a generated Java file will be lost when the file is regenerated. To preserve the code, it has to be copied back to the corresponding metadata element.

1.2.6 Overriding generated code

The generated class `Object_ID.java` extends the class `GENObject_ID.java`. If for any reason you need to override the code contained in `GENObject_ID.java`, add your overriding code to the file `Object_ID.java`. This file should be moved from the `gensrc` directory to the `src` directory.

1.3 About Decision Studio projects

Studio Inline Services are built as *Projects* within Decision Studio.

1.3.1 Starting a new project

To start a new Inline Service, use the **File** > **New Inline Service Project** menu to start your project. Choose a template from the list, name your project, and click **Finish** to create a project.

The list of templates contains templates supplied by the Oracle RTD installation, as well as any user-defined templates.

1.3.2 Importing a project

If you are opening an existing project, use the **File** > **Import** menu to import the project. If the metadata needs to be updated from a previous version, you will be prompted to upgrade.

1.3.3 Creating a user-defined template

To create a template from an Inline Service, use **File** > **Export** to export the project to a template. Choose the export type **Inline Service Template**. Templates are stored in the location defined by Inline Services Preferences. To access preferences, use **Window** > **Preferences** and choose **Inline Services**. The directory entered is where your templates are stored on the file system.

1.3.4 Downloading a deployed Inline Service

To download a deployed Inline Service, use **Project** > **Download**. You can also download it from a Real-Time Decision Server using the download icon on the toolbar. If you are going to make changes to a deployed Inline Service, it is important to follow these practices in order to preserve both your changes and the potential changes that have been made by business users. Use the following method:

1. Make sure that no business users are editing the deployed Inline Service.
2. You should always lock an Inline Service when you download, so that additional changes cannot be made by business users while you are enhancing it.
3. Make enhancements in Decision Studio.
4. Redeploy the Inline Service, releasing the locks.

During the period that you have the Inline Service locked, business users will be able to view, but not edit, the deployed Inline Service.

1.3.5 About deployment states

When an Inline Service is deployed from Decision Studio, you chose a deployment state from the deploy dialog. Three deployment states are packaged with Decision Studio: Development, QA, and Deployment. Your system administrator may add additional deployment states through JConsole.

When you test your Inline Service through the test view, the last deployment state is tested.

1.3.6 Example projects

A sample project is available to import in the *`RTD_HOME`*`\examples` directory. This directory includes the Cross Sell Inline Service that is used as an example in *Integration with Oracle RTD.*

The Cross Sell 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 presented 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 Cross Sell Inline Service highlights many features of Oracle RTD, including: driving the decisioning process through Key Performance Indicators (KPIs); optimizing competing KPIs, such as reducing cost and increasing revenue; using graphical rules-based scoring for making the right decision; and using analytical self-learning models to predict the best decision.

It should be noted that some features displayed in the Cross Sell Project are for simulation purposes only. These are clearly marked in the example and should not be used for production Inline Services.

The Cross Sell example can be viewed by importing the project. The project is located at *`RTD_HOME`*`\examples\CrossSell`.

After importing the project, you can view the following features by double clicking each of the elements and viewing the editor for that element.

Feature	Element Name	Description
Multiple KPIs	Performance Goals	The Cross Sell Inline Service is designed to optimize both the maximization of revenue and the reduction of costs of the organization.

Dynamic customer data	Data Source/CustomerDataSource Entity/Customer	The combination of a Data Source and an Entity give access to customer data that will assist us in making a decision of the type of offer to present to the customer. An *Entity* is an object that provides a means to map one or more Data Sources together into an object that represents a significant unit in the Inline Service. The data accessed through the Entity is session specific.
Cross Selling and Customer Retention Offers	Choices	The offers that are available to be extended are organized under Choices. Some of these offers are designed as cross selling offers, while others are designed to boost customer retention rates. By viewing the Score tab of each offer, you can see that offers are assigned a score for evaluation. A Score is provided for each performance goal, Revenue and Retention. Some offers (for instance all Credit Cards) inherit their scoring from the parent Choice Group. this indicates that all offers in this group are scored in the same manner. In this case the score is calculated by the formula 'Profit Margin multiplied by Likelihood of Acceptance'. Other offers (such as Reduced Interest Rate) calculate the score using a rule. Note that the Revenue Goal on Reduced Interest rate is actually scored negatively, as it represents a loss of Revenue to the organization.
Scoring Rules	Scoring Rules/Reduced Interest Rate	Scoring Rules are a way to use session data, such as information about the customer, to dynamically score the offer.

Population segment	Filtering Rules/Segment to Retain	The population can be segmented by using Filtering Rules. The outcome of this rule is two groups, a group that is eligible for customer retention offers and the remaining group that we will cross sell to. If a customer has abandoned 6 or more calls and has been a customer for over 2 years they are filtered into a group for retention offers.
Weighting decisions by population segment	Decisions/OfferDecision	The Decision element allows you to weight the decision process across the competing performance metrics. In this case, we give priority to the offers that score high on Customer Retention a heavier weight for the population segment that fits the customer retention profile.
Integration to organizational processes	Integration Points	Integration Points are the sites that touch outside systems and processes, either by gathering information (such as CallStart, which gathers information from the IVR about the customer) or provides information to an outside system (such as OfferRequest, which provides the CRM system with the highest scored offer for the customer). It should be noted that the OfferResponse Integration Point has code in the `else` branch for simulation purposes. In a production situation this would be feedback from the service center operator on whether the offer was accepted or not.

⚠ Warning: In order to simulate the passage of time when the Inline Service load generation script is run, the method `currentTimeMillis` has been overridden in `Application.java`. If you plan on using CrossSell as a basis for a production Inline Service, you need to remove the override file *RTD_HOME*`\examples\CrossSell\src\com\sigmadynamics\sdo\Application.java`.

For more about overriding generated code, see Section 1.3.5, Overriding generated code.

The Cross Sell Inline Service is ready to be deployed and loaded with data. After you deploy the Inline Service, open Load Generator by running `RTD_HOME\scripts\loadgen.cmd`. Then, choose **Open an existing Load Generator script** and browse to `RTD_HOME\examples\CrossSell\etc\LoadGen.xml`. Finally, run the script.

This script takes simulated customer data and runs the Inline Service. The data and correlations found can then be viewed in Decision Center.

1.3.7 Opening Decision Studio version 1.2 files.

If you are opening an Inline Service from a previous version of Decision Studio, it was not created as a project. To open it as a project, start a **New Inline Service Project** and then use **Create project at external location** to locate the files on your file system.

This will convert the previous version Inline Service to a Decision Studio 2.2 project.

1.4 Directory structure of Inline Services

When you create your Inline Service, you are can create your project anywhere on your file system. It is recommended that you keep all of your projects in one directory for ease of use. The default workspace is `C:\Documents and Settings\User_name\Decision Studio`.

When saving an Inline Service, the directory name is the same as your Inline Service name. The following directory structure is created for your Inline Service.

`..\Inline_Service\classes`	The compiled classes of your Inline Service.
`..\Inline_Service\dc`	A folder for custom JSPs for Decision Center.
`..\Inline_Service\etc`	Load generator scripts.
`..\Inline_Service\gensrc`	Location of the generated source code for your Inline Service.
`..\Inline_Service\meta`	The metadata of your Inline Service.
`..\Inline_Service\src`	The source code for overriding the generated code of your Inline Service.

1.5 Configuring Inline Services

Inline Services are configured and deployed using Decision Studio. Elements are added to a project and Java scriptlets with functional logic are added to certain elements. When the Inline Service is saved, XML metadata is created and Java code is generated and deployed to the Real-Time Decision Server, where the Inline Services runs. For more information about configuring Inline Services, see *Getting Started with Oracle RTD.*

1.5.1 Observer Inline Services

In monitoring, Inline Services focus on collection points: points where data about the business can be gathered. Insights and discoveries of trends and correlations in this data are made by a self-learning model that predicts future behavior and anticipates the consequences of change. This type of Inline Service is known as an Observer.

These discoveries are published to a thin client, Decision Center, where business users use these insights to make decisions. Business users also manage and optimize the Inline Service through Decision Center.

1.5.2 Advisor Inline Services

In advising a business process, Inline Services connect at key decision points as well as collection points. Decision points are places in the overall business process where key business decisions are made, such as product recommendations or retention offers. Data is first gathered at collection points, discoveries are made through the self-learning model, and then choices are scored according to performance metrics the organization wants to achieve. The highest scored choice is presented by the Inline Service at the decision point in the business process. As the success level of these choices is returned to the Inline Service through feedback, the model increases its capability of providing better and better choices. Inline Services of this type are called Advisors.

Section 2: About Decision Studio Elements and APIs

Decision Studio elements are configured graphically within Decision Studio, and the logic is added in the form of Java scriptlets. The following sections describe the properties of each element, and Java scriptlets contained by the element (if any), with examples.

This section contains the following topics:

- 2.1 About element Display Labels and Object IDs
- 2.2 About the Application element
- 2.3 Accessing data
- 2.4 Forming entities
- 2.5 The decisioning process
- 2.6 Performance goals
- 2.7 Choice Groups and Choices
- 2.8 Filtering rules
- 2.9 Scoring rules
- 2.10 Using rule editors
- 2.11 About the Decision Process
- 2.12 About selection functions
- 2.13 About analytic models
- 2.14 About Integration Points
- 2.15 About external systems
- 2.16 About the categories object
- 2.17 About functions
- 2.18 About statistic collectors
- 2.19 About Decision Center perspectives

2.1 About element Display Labels and Object IDs

As you create elements, you enter a Display Label for the element. An Object ID is automatically generated as you type the Display Label. Object IDs are automatically made to conform to Java naming conventions: variables are mixed case with a lowercase first letter; classes are mixed case with an uppercase first letter. If you have spaces in your label name, they will be removed when forming the Object ID. If you choose to manually enter an Object ID, you can overwrite the Object ID that was created for you.

You can use the ID icon on the Inline Explorer task bar to toggle between the Display Label of the object and its Object ID.

2.2 About the Application element

When a new project is started in Decision Studio, an Application object is created. Properties of the Application object are defined with the following characteristics:

- Application Parameters
- Control Group
- Model Defaults
- Logic
- Permissions

All of these values are defined through the Decision Studio interface.

2.2.1 Application parameters

Using Debugging Options

If you are testing a deployed Inline Service against a production database, and you do not want to contaminate the model data, you can use the debugging options to keep data from being written. Debugging options are:

- Disable learning: this option maintains the model's current state so that testing does not introduce additional learnings.
- Disable database writes: this option keeps data from being written to the database.

Parameters	Any parameters that the Inline Service requires. Parameters have a name, data type, default value, and can be made an array.

Adding Application Parameters

Application parameters are global-level parameters that can be defined and stored across all sessions. A session is defined by the session key you specify in the Application object. For instance, if you specify the CustomerId as the session key, whenever a new CustomorId is identified, a new session is created. Sessions are closed either explicitly by an Integration Point, or when the session times out.

Click **Add** on the **Application Parameter** tab to add a parameter. When adding parameters, you supply the Name, Type, Array, and Default Value.

Click **Remove** to remove parameters.

2.2.2 Application APIs

```
public String getSDOLabel();

public String getSDOId();
```

Returns the Object label and Id, respectively.

Application Parameter Methods

When global parameters are set, getters and setters are generated in the code. To access the application parameter, for instance `myApplicationParameter` of type `string`, use the following:

```
String param =  Application.getApp().getMyApplicationParameter();
```

Conversely, to set an application parameter:

```
Application.getApp().setMyApplicationParameter("my parameter");
```

2.2.3 Configuring the control group

The *control group* acts as a baseline so that the business user can compare the results of the predictive model against the preexisting business process. It is important to define the control group decision correctly to truly reflect the decision as it would have been made if Oracle RTD was not installed. For example, in a cross-selling Inline Service for a call center, if no cross selling was being done before Oracle RTD was introduced, then the Control Group Decision should return no choices.

No Control Group	If checked, control group will not be used, the Selection Value will be set to 0, and Use value literally will be checked and disabled. Deselect No Control Group to turn on control group and to change the settings.
Selection value	A reference to an attribute value. This value is used to seed the Random selection of requests for the control group . If Use value literally is deselected, then the selection value for the control group should refer to a session key or an attribute uniquely identifying a customer. Control group participation is determined using a pseudo-random hash of the selection value. The result of the calculation is deterministic, and depends only on the selection value and the specified size of the control group. The actual size of the control group may differ slightly from the specified size. If Use value literally option is selected, then the selection value directly determines the control group participation. The selection value in this case can be either Boolean (participation in control group is indicated by the true value) or integer (participation in control group is indicated by a non zero value).
Use value literally	Used when assignment of customers to control group is done outside of Oracle RTD. The attribute used as the control group selection value has to indicate this assignment..
Percent of Population	The percentage of the total number of customers that should be assigned to the control group.
Use for analysis	Controls whether the control group participation should be tracked by analytic.
Name for Analysis	Name that the analytic models should use for tracking the control group participation.

2.2.4 Setting model defaults

Model defaults control which model is used and how the model is set up. Most model defaults should not be changed unless you are advised to do so by Oracle Support Services.

Study name	The name of the study used by the Inline Service. Typically, each Inline Service has its own separate study. This can be achieved by keeping the name of the study the same as the name of the Inline Service. However, an existing study may be used when testing an Inline Service. In that case, learning should be disabled to preserve production data.
Persistence Interval	The interval at which model data is saved to a database.

Time Window Duration	The default value is Quarter. This feature should not be adjusted without assistance from Oracle Support Services.
First Day of Week	The default value is locale dependent. In the United States, the default value is Sunday. This feature should not be adjusted without assistance from Oracle Support Services.
First Month of Year	The default value is January. This feature should not be adjusted without assistance from Oracle Support Services.
Build when data changes by	The default value is 20%. This feature should not be adjusted without assistance from Oracle Support Services.
Significance threshold	The default value is 25. This feature should not be adjusted without assistance from Oracle Support Services.
Correlation threshold	The default value is 0. This feature should not be adjusted without assistance from Oracle Support Services.
Max Input Cardinality	The default value is 500. This feature should not be adjusted without assistance from Oracle Support Services.
Max Input Buckets	The default value is 200. This feature should not be adjusted without assistance from Oracle Support Services.

2.2.5 Writing application logic

Scriptlets to initialize and clean up an Inline Service are added through Decision Studio, using the **Initialization Logic** and **Cleanup Logic** panels on the **Logic** tab of the application element.

These scriptlets are inserted into the `init` and `cleanUp` methods of the Application class. The `init` method is called when the Inline Service is being loaded. The method `cleanUp` is called when the Inline Service is unloaded. An active copy of the Inline Service is unloaded, and the service undeployed or redeployed. If the application is redeployed, `init` will be called again.

2.2.6 Adding imported Java classes

If the `init` or `cleanUp` method refers to user-provided Java classes, these classes may have to be imported. To add additional import statements, click **Advanced** next to the description.

2.2.7 Setting Inline Service permissions

Inline Service permissions are set for users and groups who will be working with the Inline Service during its lifecycle. The following permissions are available for each Inline Service, and are appropriate for Inline Service developers and Decision Center business users.

Open Service for Reading	Allows Decision Center to open the Inline Service in a read-only mode. This mode is appropriate for a business user who will use Decision Center to view reports.

Open Service	Allows the Decision Center user to edit and redeploy an Inline Service to a Real-Time Decision Server. Allows Decision Center to open the Inline Service. Open Service implicitly gives permission for the user to Open Service for Reading. This mode is appropriate for a business user who will use Decision Center to view reports and update Inline Services.
Deploy Service From Studio	Allows Studio user to deploy the Inline Service. To redeploy an existing Inline Service, the user has to have Deploy permission for both the existing and the new service. This mode is appropriate for the Inline Service developer who will use Decision Studio to deploy Inline Services.
Download Service	Allows the Decision Studio user to download an existing, deployed Inline Service from a server. This mode is appropriate for the Inline Service developer who will use Decision Studio to deploy Inline Services.

Inline Service permissions work in conjunction with server-side Cluster Permissions to secure the Inline Service from being changed or redeployed by an unauthorized user. For more information on setting Cluster Permissions, see *Installation and Administration of Oracle RTD*. In addition, you can set permissions on Decision Center perspectives. See Section 2.19, About Decision Center perspectives for more information.

Use the Permissions tab of the Application element to set Inline Service permissions. Use Add to add users or groups to the Inline Service. To retrieve users and groups from the server, click Get Names to retrieve groups, then select Show Users to see all the users.

You can choose a user or group from the list, or you can enter a name in the User or Group field. After you have added users or groups, select the permissions you would like to apply under Granted.

To remove users or groups from the Inline Service, select the user or group and click Remove.

Note: When Windows Authentication is enabled, you cannot retrieve a list of users and groups by clicking Get Names. Instead, you must enter the name of the user or group to which you want to assign permission in the User or Group field.

2.3 Accessing data

To access data within your Inline Service, use the *entity*, *data source*, and *session* elements.

Entities provide an abstract way to bring together data from multiple sources to form an object that is of use to the overall Inline Service. Entities are comprised of a number of *attributes* that describe the contents of the Entity. Entities also provide methods to access their attributes.

An Entity, such as a Customer, may combine incoming data from different sources, such as an account number entered through an IVR and customer history retrieved from a corporate database. One of the entity's attributes may be a *key,* or unique identifier.

Data sources act as *suppliers* of data. They provide a way to manage the connection to relational database tables and stored procedures. Data sources identify columns in a database table, or result sets from stored procedures as attributes. These attributes are mapped to entity attributes.

The session object is a specialized entity that identifies which attributes are available in memory to the Inline Service. Those attributes can be comprised of a entity, such as Customer described above, as well as attributes that are set by some other source, such as calculation. A session object is used to store information for a user session. Attributes stored in session are available throughout the Inline Service, and are destroyed when the session is closed.

To access data, you typically follow these steps:

1. Create a data source, based on a SQL table or stored procedure.
2. Create an entity with attributes from one or more data sources.
3. Add a key value to the entity.
4. Add the entity to the session as an attribute, and assign a session key.
5. Map the entity attributes to data source columns or output values.
6. Map the entity key to a session key or function.

2.3.1 Accessing Siebel Analytics data

Siebel Analytics Server exposes an ODBC client interface for accessing data stored in OLTP and OLAP databases. RTD Decision Service uses the JDBC-ODBC bridge included in Java Runtime Environment (JRE) to connect to the ODBC driver provided by Siebel Analytics Server.

From the RTD Decision Service point of view, Siebel Analytics Server is a SQL data source similar to a regular database. Subject areas in Siebel Analytics Server are treated as database tables by the Inline Service. Column names are compound, combining two levels of the presentation object hierarchy.

For details on adding a JDBC data source that can be accessed by a Decision Studio data source, see *Installation and Administration of Oracle RTD*.

2.3.2 About data sources

Data is accessed within Inline Services using the elements *data source* and *entity*. A *data source* is an abstract provider of data. Data sources act as *suppliers* of data to the Inline Service.

Data sources are created entirely within Decision Studio. There is no need to access data sources through the Java API. There are two types of data sources: SQL data sources, and Stored Procedure data sources.

2.3.3 Creating SQL data sources

SQL data sources are defined with the following characteristics:

Description	Description of the data source.
Data Source	The JNDI name of a JDBC data source. To create a new data source, see *Installation and Administration of Oracle RTD.*
Table Name	The name of the table. This name is always case insensitive, even when the database itself is case sensitive.
Outputs Column Name	The columns to select from the data source.

Outputs Type	Data type of the output column.
Inputs Column Name	The columns used in the WHERE clause of the query to the data source. This is the column or columns you will match on in order to select data from the data source.
Inputs Type	Data type of the input column.
Allow multiple results	Allows multiple rows to be returned. If this option is not selected and multiple rows are returned, only the first one is used.
Advanced	The Advanced button lets you choose to show the element in Decision Center, and to change the label of the element. Changing the label of the element does *not* change the Object ID.

Adding columns to the data source

Click Add or Remove to add or remove columns from the data source. If you expect more than one row, select Allow Multiple Results. If you do not select this option and multiple rows are returned, only the first will be used.

Importing database column names

Click Import to connect directly to the data source. All of the database tables for the specified data source will be shown. If no data source is specified, the default data source SDDS is used.

Select Include objects from all schemas to display tables not defined in the data source schema. Tables from all accessible schemas will be shown, with the table schema displayed in a separate column.

Choose the table you want to import, and the column names and data types of those columns are imported. If there are columns you do not need, use Remove to remove them.

Setting the input column

The Input column is the column you will match on the database table to retrieve the rows needed for the session. Most likely, this will be a value of the primary key or a unique index column to return a single record. Otherwise, if you need larger result sets, it may be a non-unique indexed column. Choose the attribute on which you want to match using Add.

2.3.4 Creating Stored Procedure data sources

Stored Procedure data sources are defined with the following characteristics:

Description	Description of the data source.
Data Source	The JNDI name of a JDBC data source
Procedure Name	The name of the Stored Procedure. This name is always case insensitive, even when the database itself is case sensitive.
Inputs and Outputs	Input and output parameters for the stored procedure. Each Input and Output has a Name, a Type, and a Direction.
Result Sets	The result sets from the stored procedure.

Allow multiple results	Allows multiple results to be returned. If this option is not selected, and multiple results are returned, only the first is used.
Result Set Details	The column names and type of the results expected.
Advanced	The Advanced button lets you choose to show the element in Decision Center and to change the label of the element. Changing the label of the element does *not* change the Object ID.

Adding attributes to the data source

Click Add or Remove to add or remove attributes from the data source. If you expect more than one result, select Allow Multiple Results. Choose whether the attribute is an Input, Output, or Input/Output.

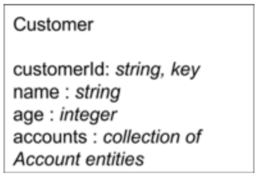
Attributes must be ordered. Use Up or Down to order the attributes.

Adding result sets to the data source

If the Stored Procedure has a result set, use the Result Set Add button to add a result set. Use the Result Set Detail Add button to add the column names and types of the result set. The result set columns must be defined in the same order they are returned by the stored procedure.

2.4 Forming entities

An *entity* is a set of named attributes and methods to access those attributes. One attribute is usually designated as the entity's key. For example, a simple customer entity might look as follows:



Customer

customerId: *string, key*
name : *string*
age : *integer*
accounts : *collection of Account entities*

In this entity, the *customerId* is the key of the entity, *name* and *age* are simple attributes, and *accounts* is a collection of Account entities.

2.4.1 About the Session entity

One specific kind of entity is the Session entity.

As an example of using sessions, consider a client Web application, where each request supplies the HTTP session-ID of the Web application as the session key. When the first request arrives with a new HTTP session-ID, the Real-Time Decision Server notices that this session key is new, and consequently creates a new Session object and makes it available to the Integration Point as it executes with the request. Any subsequent requests using the same HTTP session-ID will access the same Session object.

The Session entity is automatically created for every Inline Service.

About session keys

A session key is a field passed in the request that identifies an instance of a Real-Time Decision Server server-resident Session object that will be available to the Integration Point. The processing of the Integration Point may implicitly or explicitly save information in the session, so that it will be available to subsequently invoked Integration Points.

2.4.2 Creating entities

Entities are defined using Decision Studio. Entity names must begin with an uppercase letter. Entities are defined with the following characteristics:

Description	Description of the entity as entered in Decision Studio.
Key	A unique ID for the entity. Click **Add Key** to add a key to an entity.
Each entity attribute is defined with the following data:	
Description	Description of the attribute as entered in Decision Studio.
Type	Attribute types are either primitive types, Java classes, entity types, Choices, or Choice Groups.
Array	Whether single-valued, or a collection.
Default value	The default value, which can be a constant, a function, or a reference to an attribute.
Add attribute/key options	
Use for analysis	Select this option to use this attribute for analysis within the predictive model.
Category	The category of the attribute. Categories help organize the display of attributes in Decision Center.
Analysis options	Additional analysis options are available for Date attributes. To use Dates for analysis, specify the pattern you are interested in analyzing. The effect of month, day of month, day of week, and time of day can be analyzed separately, or in any combination.

2.4.3 Adding attributes and keys to the entity

Click **Add Attribute** or **Add Key** to add attributes to the entity. Attribute names must begin with a lowercase letter (for example, `aSampleAttributeName`). If the attribute is a collection, select the **Array** column.

> ⚠ **Warning:** When adding a Key attribute, the data type will automatically be String. If the data type of your data source column or output parameter is a type other than String, use a transformation function when you set the input on the data source.

2.4.4 Importing attributes from a data source

To automatically add all of the attributes of a data source, click **Import**, then choose a data source from which to import. If you would like to import from more than one data source, repeat the procedure. Click **Remove** to remove any unwanted attributes.

When using **Import**, select **Build data mappings for selected data source** to automatically map the attributes to the data source. If the entity is nested (for example, in a one-to-many relationship) and the attributes are mapped indirectly, deselect this option.

2.4.5 Using attributes for analysis

Select Use for Analysis to have the attribute added to the analytical model.

2.4.6 Decision Center display

The option Show in Decision Center is selected by default. Deselect this option if you want the attribute to be hidden from Decision Center users. Choose a Category to control the display of the attribute in Decision Center.

2.4.7 Adding a session key

If the session key value that you choose to use is an attribute of an entity, first add the entity to the session. To do this, click Add attribute to add a new attribute.

For instance, assume you want to make the session key the `customerID` attribute from the Customer entity. Click Add Attribute, then add an attribute to the session called `customer`. The type of this attribute is an entity type, namely `Customer`.

To access the entity types, use the drop-down list on the Type column and choose Others. The Type window appears. Choose the Entity Type for this attribute.

To add the session key, click Select from Session Keys from Dependent Entities. All key values from entities that are attributes of the session are available to be chosen as a key value for the session. Choose the key on which you want to base the session, in this instance `customerID`.

Note: The keys of cached entities cannot be used as session keys.

To add a session key that is not from a dependent entity, use Add Key. Once you have added the key, click Default value to map the key to a attribute, constant or function call.

2.4.8 Adding attributes to the session

Use Add Attribute to add an attribute that you want to make available for the entire session. Session attributes have getters and setters generated for them, as do other entity attributes.

2.4.9 Mapping attributes to data sources

After creating an entity using Decision Studio, map the attributes of the entity to values that are either constant, calculated, a reference to an attribute of a data source, or to the session key.

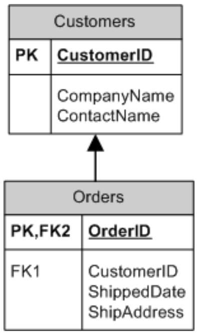
Mapping is done through the Mapping tab of the entity object. To map the attributes of an entity to a data source, use the Source column to choose the path to the data source column or output parameter that will supply the attribute with data.

To map the key value, click Input Value from Data Source Input Values. Your key value will appear here when you map the attributes to data source values. You can map the key to a session key attribute, to another entity key value, or to a function. The input type must be of type String. If it is not, use a function to transform the non-string value.

2.4.10 One-to-many relationships

To access data in an entity in a one-to-many foreign key relationship, make the related entity an attribute of the first entity. For example, take the following relationship:

The Customers table has a key, `CustomerID`. Customers have many Orders, which are identified by `OrderID` and a foreign key `CustomerID`.



1. In Decision Studio, define a data source for each of these tables.
2. Create an entity for Customers and Orders.
3. Add Customer to the session, as that is the key to retrieving the next level of data.
4. Choose `CustomerID` as the session key.
5. To associate the one-to-many relationship between Orders and Customers, add an attribute to Customer called Orders, of entity type `Orders`. Since there are many Orders for one Customer, make it an array.
6. You can map all of the attribute values through the Customer entity mapping tab.

2.4.11 Adding imported Java classes

To add imported Java classes to your Inline Service, click Advanced next to the description.

2.4.12 Session logic

The session element can accept scriptlets that are executed on initialization and exit of the session. The cleanup scriptlet is executed either when the session is forced closed, or when it times out.

2.4.13 Session APIs

```
public String getSDOLabel();

public String getSDOId();
```

Returns the Object label and ID, respectively.

You can use `session()`to access the other entities of the Inline Service. For example:

```
session().getCustomer().getCustomerId();
```

where `Customer` is an entity and `customerId` is an attribute of that entity.

Use `session()`to access the instance of the application session. Session has the following APIs available:

```
public boolean isTemporary();
```

If no session keys have been passed in, the session is considered temporary.

```
  public IntegrationPointRequestInterface getRequest();
```

Used to access an Integration Point request outside of the Integration Point. Returns the current request.

```
  boolean isClosed();
```

Returns whether the current instance of the session has been closed.

```
  Set getKeys();
```

Returns any known session keys. This may or may not be the complete set, depending on where it is called.

```
   public void close();
```

Close the current session instance.

```
   public ApplicationInterface getApp();
```

Gets the application object of the current session.

2.4.14 Entity APIs

```
public String getSDOLabel();

public String getSDOId();
```

Returns the Object label and Id, respectively.

2.4.15 About entity classes

In addition to the normal classes generated, an array class is also generated for each entity. The generated classes have a property, getter, and setter for each attribute. Hence, the definition of entities such as Customer, Account, and Call will result in classes with these names, as well as another class, representing a collection of that class.

For instance, for the Account entity, the following two classes are generated:

```
Account

SDAccountArray
```

The second class represents a collection of Accounts. So, when our Customer entity has an attribute named accounts of type Account (with multiplicity set to multiple), then the following gets generated for Customer:

```
 Customer  {

 SDAccountArray getAccounts() {

   }

 void setAccounts(SDAccountArray accounts) {

   }

void addToAccounts(Account account) {

   }

 }
```

2.4.16 Creating entities

Because a class is generated for each entity type, you create an entity with the *new* operator as you would any Java object. For example:

```
Customer cust = new Customer();
```

Optionally, if the entity is the child of another entity, pass in the name of the parent entity. Session can be a parent entity of any other entity.

```
Customer cust = new Customer(entityParent);
```

2.4.17 Adding entity keys

Most entities are not very useful unless they have a value for the key attribute. The key attribute, as with any attribute, is set using a generated setter:

```
Customer cust = new Customer();

cust.setCustomerId(newKey);
```

2.4.18 Accessing entity attributes

As mentioned previously, getters are generated for each attribute. The form of the getter depends on whether the attribute has one value or more than one value. The sample Customer entity would have the following getters:

```
String id = cust.getCustomerId();

String name = cust.getName();

double age = cust.getAge();

Collection accounts = cust.getAccounts();
```

Corresponding setters are also generated. We have already seen the setter for `customerId`, but here are the others for our Customer example:

```
cust.setName("Fred Johnson");

cust.setAge(42);

cust.setAccounts(newCollection);
```

And, because Accounts is an attribute that has multiple values, you can also add to the collection:

```
cust.addToAccounts(anotherAccountObject);
```

An array can be added using

```
cust.addAllAccounts(anotherAccountArray);
```

2.4.19 Resetting and filling an entity

Three special methods are provided to reset and to fill an entity.

```
cust.reset();
```

Resets all keys except session keys and all attributes.

```
cust.resetAttributes();
```

Resets all attributes, but does not reset keys.

```
cust.fill();
```

`Fill` recursively fills the values of the attributes according to the entity mapping, forcing it to refresh the data through the data source, calculated or constant value. Any attributes of entity type are also filled.

Reset and fill should not be called on cached entities.

2.4.20 About cached entities

Entities can be cached on the server so that they are easily accessible. To cache entities, click the Cache tab of the entity.

Cache has the following characteristics:

Enable caching for this entity type	Select this option to enable caching. Cached entities are treated exactly like non-cached entities and have the same API, except that cached entity keys may not be used as session keys.
Max number of items to cache	The maximum number of items to cache. Items are flushed in a first in/first out manner.
Caching strategy	
Use fixed lifetime	Number of seconds each object stays in cache before being refreshed.
Use fixed period	Number of seconds before the entire cache is refreshed.
Never refresh cache	Cached items stay in cache until the maximum number is reached.

If an entity is marked for caching, use the following to set the attributes. Once you create the entity, set the key values and then get the attribute values from the cache. Cached entity attributes (other than the key) do not have setters. This keeps the entity in sync with the cached version.

```
Customer cust = new Customer();

cust.setCustomerId(newKey);

cust.getCustomerId();

cust.getName();

cust.getAge();

cust.getAccounts();
```

2.5 The decisioning process

The decisioning process is based on a framework that takes into account the overall performance goals with which an organization is concerned, the performance metrics that measure those goals, the action required to score each of the available choices, and a weighting of that score based on segments of the population.

The following elements are part of this framework:

1. Performance Goals
2. Decisions
3. Choice Groups
4. Choices
5. Filtering Rules
6. Scoring Rules
7. Predictive Models

2.6 Performance goals

In designing a decision process for an organization, first consider the overall performance goals of the organization. Performance goals are comprised of the specific metrics with which the organization has chosen to measure their success. Some common performance metrics are:

- Revenue
- Costs
- Number of products per customer
- Customer satisfaction

The performance metrics are configured with an optimization direction (maximize or minimize) and a normalization quotient.

Performance goals have the following characteristics:

Performance metric	Metrics with which the organization has chosen to measure their success that relate to the overall goals of the organization.
Optimization	A value, Minimize or Maximize, that indicates the direction in which to optimize the performance metric.
Required	Check if scoring for the performance metric is required. If a metric is not marked required, and a score is not available through lack of data, Oracle RTD can provide a score by examining other scores. If it is marked required, a general score will not be provided and the metric is marked not available and dropped from the scoring process.
Normalization factor	The relative value to the organization of this performance metric.

2.6.1 Adding a performance metric

Click Add to add performance metrics. Add a metric (for example, revenue), an optimization direction (maximize), and whether the metric is required to have scores available for a decision to be made.

Once you have added all of your metrics, you must decide on the normalization factor.

2.6.2 Calculating a normalization factor

The normalization factor represents the relative value to the organization. Many times, the the performance metrics are not measured on the same scale. For instance, churn is measured by number of customers who turn over, while revenue is measured in dollars. If a churn metric and revenue metric are part of the same performance goal, you must find some leveling factor for them. If, for example, you know that the loss of one customer costs your organization $500, the you would use 500 as a normalization factor, and mark the the churn metric as Minimize.

2.7 Choice Groups and Choices

Choice Groups and Choices are elements that allow the selection of one or more choices among a number of possible choices. The Choice is either returned to a Decision, so that it can be return by an Advisor, or it is registered with a model by an Informant. The process of selecting choices is divided into the following computations:

1. Eligibility: a set of rules that determines whether or not a choice is available to be selected for the decision.

2. Scoring: the computation of a score along each Key Performance Indicator (KPI) as defined by the performance metrics. A score assigns a numerical value to the expected effect a choice will have for a given performance metric, if selected.

3. Normalization: brings the scores along the different performance metrics to a common scale that enables the comparison of scores.

4. Totaling: produces a single number for each Choice. This number can be used to compare the relative benefit of each Choice.

5. Selection: determines which Choices are to be used. Selection usually applies to the totals computed in Totaling.

2.7.1 About Choice Groups and Choices

Choice Groups and Choices have the following:

Attribute values	The attributes that make up the choice. These can be inherited from the parent Choice Group, or assigned at the Choice level.
Scores	Each choice will be scored according to the definition in the Scores tab. Choices are score against all of the performance metrics that are contained in performance goals.
Choice Events	Choice events are only described at the Group level. These events are defined about the lifecycle surrounding a Choice. For instance, a Cross Selling Offer made may have events such as Offered, Accepted, and Product First Used.
Rules	Rules affect whether a Choice is able to be considered for Scoring and an ultimate Decision. A rule may be altered by Decision Center users.
Advanced	The Advanced button lets you choose to show the element in Decision Center, and to change the label of the element. Changing the label of the element does *not* change the Object ID.

2.7.2 About Choice attributes

Choice attributes have the following characteristics:

Name	The name of the attribute.
Category	The category to which the attribute belongs. Categories are defined by the Category element.
Type	Data type of the attribute.
Array	Whether the attribute is a collection.
Inherited Value	The value, if any, that the Choice Group or Choice attribute has inherited from its parent.
Value	The value of the attribute. This value always overrides an inherited value.
Show in Decision Center	Select this option to make the attribute visible to business users in Decision Center. Deselect for internally used attributes.
Use for indexing	Select this option if you want to be able to look up the Choice by the attribute specified. For example, assume you have a choice attribute called `name`. A static method is generated on the Choice Group called: `getChoiceWithName(String name)` This method returns a choice.
Send to client	Select this option if the attribute will be sent to the outside client calling the Inline Service that returns this choice.

2.7.3 Adding Choice attributes

To add or remove Choice attributes, click Add or Remove. To edit a Choice attribute, right-click the attribute and choose Properties. You can only edit Choice attributes at the highest level that they are defined.

2.7.4 About Choice Group attributes

Choice Group attributes are used to allow flexible group level rules. Group attributes apply only at the Choice Group level, and so are not assignable for individual Choices.

2.7.5 About Choice attributes

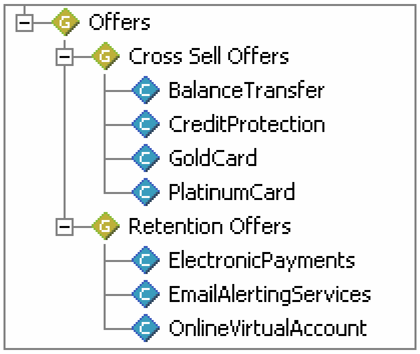
Choice attributes define the Choice. Choice attributes are set at the Choice Group level so that each Choice in a Group has the same set of attributes, and then they are individually defined at the Choice level. Choice attributes may have default values that can be set and overridden at lower levels.

Choice Groups and Choices are defined hierarchically. The hierarchy should follow the logical taxonomy of the Choices. At the top level, it is necessary to consider the definition of Choice attributes that make sense for a whole subtree. At lower levels, the shape of the hierarchy is typically determined by organizational issues.

Choice attributes are typically defined at the higher levels of the hierarchy. Some attributes have a default value that may be marked as nonoverrideable, which means that the value provided by default is the value that will be used. This is typically done when computations are involved. This is useful when you do not want a business user to update the attribute after deployment.

Choice values can be constant, attribute or variable, a function call or predictive. A predictive attribute is calculated by the model at run time and used as a scoring device to select the proper Choice by the Decision or other logic. Predictive attributes have no default value.

As an example, examine this choice group:



The Choice attributes set at the Offers level are:

Attribute	Type	Value
Message	String	No default value. Values assigned at Choice level for each Choice.
ShouldRespondPositively	Boolean	The function `ShouldRespondPositively()` that returns a Boolean value about whether a customer will respond positively to a given Choice.
likelihood	Predictive/Double	A placeholder attribute that is assigned by the model. Used for likelihood that the choice will be accepted. There is no default value as this is calculated at run time.
Profit Margin	Double	Default value of 0.5. Values assigned at Choice level for each Choice.

Each Choice overrides the Profit Margin and Message values with a value that is indicative of that Choice. However, the default value will be available at run time in case the server cannot respond in an effective time period.

No Choices override the `ShouldRespondPositively` attribute, as they all use the same function to determine that value. The likelihood is calculated by the model for each Choice at runtime.

There is another attribute at the Group level. It is a Group Attribute.

Attribute	Type	Value
averageLikelihood	Predictive/Double	An attribute used by the model as an average of likelihoods *across all choices.* Used as a likelihood if a likelihood for a given Choice is not available. There is no default value, as this is calculated at run time.

2.7.6 About Choice scoring

Choice Groups and Choices inherit scoring from their parents. In scoring the Choice or Choice Group, you identify the performance metrics that apply to that Choice and then apply a scoring method to it. Scoring methods can be a scoring rule, function, constant, or the likelihood of an event occurring on a Choice Event Model.

For instance, assuming a Choice Group structure described above, some of the Choices may have scoring similar to:

Mileage Plus Card	
Performance Metric	Score
Increase Revenue	A function that uses the likelihood of the customer to accept the offer and the expected profit margin of the card to calculate the revenue potential of the offer. The likelihood is computed by a model.

Gold Card	
Performance Metric	Score
Increase Revenue	An inherited constant from the choice group level.

Credit analysis	
Performance Metric	Score
Increase customer retention	A scoring rule that uses customer data to assign a score.

Scoring must return a double data type, but more importantly, must conceptually match the normalization rate given to the performance metric in the performance goal. For example, if that leveling factor was between dollars and dollars, the functions must return a value representing dollars.

2.7.7 About eligibility rules

Choices and Choice Groups have rules that determine their eligibility to participate in the decision. The rule determines whether the Choice is eligible to participate in the selection function or rule of the Decision or logic that makes the

Choice selection. The Group-level rules are located on the **Choice Eligibility** and **Group Eligibility** tabs of the Choice Group editor. Choice-level rules are located on the **Eligibility Rule** tab of the Choice editor.

The rule determines whether the subtree headed by a Choice Group or a Choice is eligible for a decision. Note that even if Choices themselves are eligible, they will not be eligible unless all their ancestors are eligible.

For information about how to use the editors for these rules, see Section 2.10, Using rule editors.

2.7.8 Evaluating Choice Group rules and Choice eligibility rules

Choice Group rules and Choice rules are inherited and additive. That is, if there are rules at the Choice Group (Group and Choice rule) and rules at the Choice level, it is as if there is a logical AND extending the rules. The inherited rules are shown in an expandable section at the top of the rule labeled **Inherited eligibility conditions**. Use the buttons ⏫and ⏬ to expand and collapse the sections.

Take the following instance:

Group$_1$ has rules GroupRule$_1$ and ChoiceRule$_1$

Group2 is a child of Group1 and has rules GroupRule2 and ChoiceRule2

Group$_2$ has a Choice, Choice$_1$, and it has a rule, Rule$_1$

In evaluating the rules for Choice$_1$, the following rules will be invoked:

GroupRule1 AND GroupRule2 AND ChoiceRule1 AND ChoiceRule2 AND Rule1

2.7.9 Determining eligibility

When determining eligibility for a Dhoice, the Dhoice rule is first tested on the Choice. Then, if the Choice is eligible, the parent rules are tested using `super.isEligible()`. It is important to note that we do not test with `this.getParent().isEligible()` because that would test the parent for eligibility, not the Choice.

Eligibility for the following Choice:

Group$_1$ has rule GroupRule$_1$

Group2 is a child of Group1 and has rule GroupRule2

Group$_2$ has a Choice, Choice$_1$, and it has a rule, Rule$_1$

would be determined in the following manner:

If Choice$_1$ is eligible with Rule$_1$ test with GroupRule$_2$

if eligible test with GroupRule$_1$

2.7.10 Choice Group APIs

```
public String getSDOLabel();

public String getSDOId();
```

Returns the Object label and Id, respectively.

```
public Choice getChoice(String internalNameOfChoice);
```

Returns a Choice object from the Choice Group.

```
public Choice getChoiceWithAttributeID(AttributeType val);
```

When a Choice attribute is marked for indexing, this method is used to return the Choice as referenced by the indexed attribute.

2.7.11 Choice APIs

```
public String getSDOLabel();

public String getSDOId();
```

Returns the Object label and Id, respectively.

To get the Choice Group in which the Choice is contained, use:

```
public ChoiceGroup getGroup();
```

Choice event tracking API consists of two methods defined on choices:

```
void recordEvent(String eventName);

void recordEvent(String eventName, String channel);
```

Typical code for an Integration Point recording a choice event would be:

```
String choiceName = request.getChoiceName();

String choiceOutcome = request.getChoiceOutcome();

ChoiceGroup.getChoice(choiceName).recordEvent(choiceOutcome);
```

Tracking of extended and accepted offers is required by many Inline Services for eligibility rules that depend on previous offers extended to, or accepted by, the same customer in the past.

Two kinds of questions, both related to a particular customer, can be answered by the Choice event history:

- How long ago was an offer extended or accepted?
- How many times was an offer extended or accepted during a given recent time period?

The answers to these questions are provided by API methods defined on Choices and Choice Groups:

```
int daysSinceLastEvent(String eventName);

int daysSinceLastEvent(String eventName, String channel);

int numberOfEventsDuringLastNDays(String eventName, int numberOfDays);

int numberOfEventsDuringLastNDays(String eventName, int numberOfDays,
String channel);
```

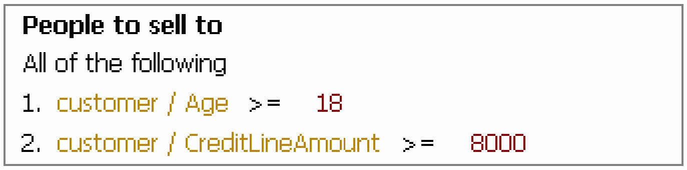
2.8 Filtering rules

Filtering rules are available as standalone rules; they are identical in usage to eligibility rules that are associated with Choices or Choice Groups. Their main function is to segment the population for which Decisions are being made. For more about eligibility rules, see Section 2.7.7, About eligibility rules.

For information on editing rules, see Section 2.10, Using rule editors.

2.8.1 Using filtering rules to segment population

Filtering rules are used to segment the population. This, in turn, is used by the Decision to apply performance metrics to different segments of the population. A typical rule used to segment population may be as shown below:



People to sell to

All of the following

1. customer / Age >= 18
2. customer / CreditLineAmount >= 8000

This rule targets customers over the age of 18 with a credit line amount over $8000. Filtering rules are used by Decisions.

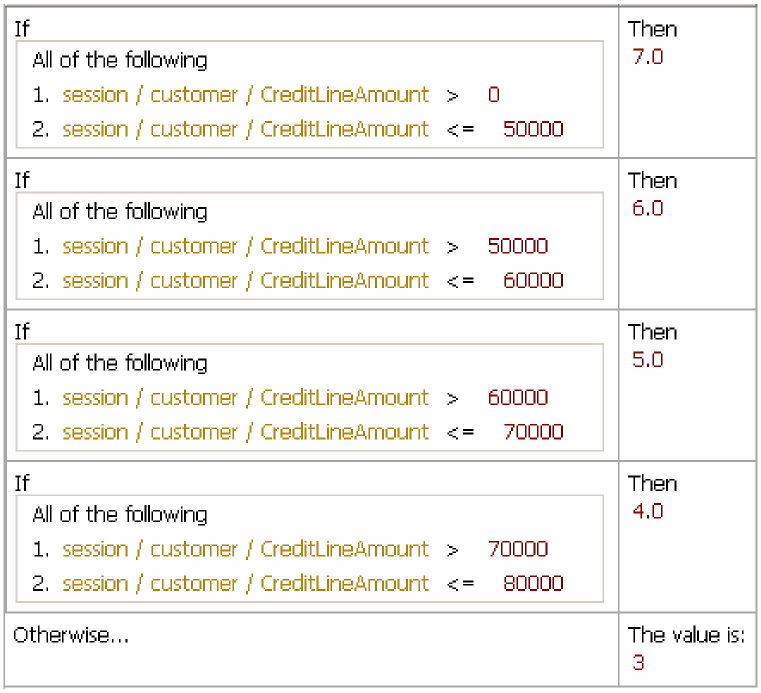
2.9 Scoring rules

Scoring rules are available as standalone rules to be used by a Choice to assign a Score. Scoring rules are very similar in function to eligibility rules on Choices. For more on using the rule editor, see Section 2.10, Using rule editors.

In addition to the functionality that filtering rules have, scoring rules evaluate to a numeric score, in the form of a double. Scoring rules have a default value if none of the rule segments evaluate to true.

To add a value, click under Then or The value is in the Value column. Edit Value appears. Then, on the ellipsis ..., edit the value as you would any other rule value.

For instance, the scoring rule below assigns scores based on the credit line amount of a customer. If they do not fit into any of the credit line range categories, the score defaults to 3.



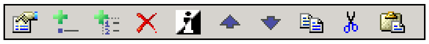
If	Then
All of the following 1. session / customer / CreditLineAmount > 0 2. session / customer / CreditLineAmount <= 50000	7.0
All of the following 1. session / customer / CreditLineAmount > 50000 2. session / customer / CreditLineAmount <= 60000	6.0
All of the following 1. session / customer / CreditLineAmount > 60000 2. session / customer / CreditLineAmount <= 70000	5.0
All of the following 1. session / customer / CreditLineAmount > 70000 2. session / customer / CreditLineAmount <= 80000	4.0
Otherwise...	The value is: 3

Description	Scoring Rules can be adjusted by Decision Center users, so it is very important to describe your scoring rule adequately. It is suggested that you include the range that the score is to work over.
Advanced	The Advanced button lets you choose to show the element in Decision Center, and to change the label of the element. Changing the label of the element does *not* change the Object ID.

2.10 Using rule editors

Rules are used for several purposes within Decision Studio and Decision Center: for determining the eligibility of Choice Groups and Choices to take part in a Decision, as standalone, reusable rules for filtering population segments, and as standalone, reusable rules for scoring Choices.

The Rule editor toolbar provides access to features used to edit rules. This toolbar is sensitive to the context of the task you are performing. A right-click context sensitive menu is also available with these functions.



The functions from left to right are:

- Edit rule properties
- Add Rule
- Add Ruleset
- Delete
- Invert
- Move up

- Move down
- Copy selection to clipboard
- Cut selection to clipboard
- Paste from clipboard

The editors that are used to create rules are very similar. The following section describes how to create rules using these editors.

2.10.1 Adding rule segments

Add a rule using the Add Rule button. There are two types of rule segments to add: Rules and Rule Sets. By default, rules are two-operand rules. To change to a single-operand rule, click the rule's number to select it. Click the in the corner of the operand to choose between single and double operands. Single operands always evaluate as a Boolean.

About logical operators

There are four logical operators for a set of rules:

- All of the following (logical and). The Logical Expression will be true when all of its child expressions are satisfied.
- Any of the following (logical or). The Logical Expression will be true when any one of its children is true.
- None of the following (logical not and). The Logical Expression will be true when all of its child expressions are false.
- Not all of the following (logical not or). The Logical Expression will be true when any one of its child expressions are false.

You can change the value of the Logical Operator by clicking it, then selecting another one from the popup menu that appears.

Editing rule properties

Both Filtering and Scoring rules have rule properties that can be set. To edit rule properties, click the Rule properties button. Edit rule properties appears.

Rule properties include call templates and negative call templates. Call templates provide a user-friendly way to describe how to call a rule from another rule.

To define a call template, add the number of parameters for the rule using the Add button under Parameters. Using {0}, {1}, and so on as arguments, and phrasing to describe the rule, define the template for call. It is important to use good phrasing, as this is what will be shown when using the rule.

For instance, a rule that checks how many calls have come in from a user in the last *x* number of days could be phrased as follows:

```
{0} calls in last {1} days
```

The negative call template is used when a rule is inverted, and should express the opposite:

```
        Not {0} calls in the last {1} days
```

Rule properties also let you assign which Choice Group to use with the rule. By selecting **Use with choice group**, you can specify which Choice Group or its Choices will provide the Choice attributes for use by parameters. These attributes will be available when you edit the value of an operand.

Selecting an operator

Click the operator = ◢, then click the lower right corner to select an operator.

The following operators are available:

none	A Simple Expression that only has one operand
=	Left is equal to Right
≠	Left is not equal to Right
<	Left is less than Right
<=	Left is less than or equal to Right
>	Left is greater than Right
>=	Left is greater than or equal to Right
in	Left value is contained in a List on the Right side
not in	Left value is not contained in a List on the Right side
includes all of	Left list includes all of the values of the Right list
excludes all of	Left list contains none of the values of the Right list
includes any of	Left list includes any one of the values of the Right list
does not include all of	Left list does not include all of the values of the list on the Right

Editing the value of a rule element

To edit the elements of the rule, click the left side, then click the ellipsis .... **Edit Value** appears. You can choose from a constant, attribute, or function call. Select **Array** at the top of the page to specify an array value.

If you choose **Constant**, provide:

Data type	The data type of the item.
Value	A constant value.

If you selected **Array**, add as many items to the array as needed, then choose a data type and provide a value for each.

If you choose **Attribute**, provide one of the following:

Group attribute	Attributes that are part of the Choice Group or its Choices that is selected in the Properties of the rule.
Session attribute	Attributes that are part of the Session entity.
Application attribute	Attributes that are a member of the Application element.

Optionally, select **Apply filter type** and choose a **Data type** to filter the attributes by type. If you have selected **Array**, add as many items to the array as needed, then assign an attribute value for each.

If you choose **Function call**, provide one of the following:

Filtering rules	Standalone filtering rules defined for the Inline Service.
Scoring rules	Standalone scoring rules defined for the Inline Service.
Function calls	Standalone functions defined for the Inline Service.

Optionally, select **Apply filter type** and choose a **Data type** to filter the attributes by type. If you have selected **Array**, add as many items to the array as needed, then assign a function or rule for each.

Adding a Ruleset

Click the **Add Ruleset** button. A new group of rules appears in the expression. By default, it is an **All of the following** expression. To change the expression, click the right corner.

Inverting a rule

Using the **Invert** button, you can invert different elements of the rule. By selecting the number of a rule segment, you can invert the operator of a rule. For instance, if the rule operand was =, it will be inverted to <>.

Logical operator for a rule can also be inverted. Select the logical operator and use **Invert**. For instance, **All of the following** becomes **Not all of the following**.

The final use for **Invert** is to invert a Boolean, or single operand, rule. When this type of rule is inverted, it is transformed to the negative call template of the function that defines the rule.

2.11 About the Decision Process

Decisions are called by Advisors to score Choices according to their functions or rules, and return one or more Choices from a Choice Group. The set up of a decision must include at least one Choice Group from which choices are selected, and a function or rule to score the choices. At run time, the Decision collects all the eligible choices that are in the subtrees of the configured Choice Groups. Then, the choices are scored to finally select a number of choices.

Examples of scores include:

- Likelihood of being interested
- Expected business benefit in dollars
- Expected time savings

Alternatively, a custom selection function can be written to select the choice.

A Decision is typically used to select one or more Choices out of a group of eligible Choices. The most common use is within an Advisor that refers to two decisions, one for the regular processing and one for the control group.

Selection Criteria	
Select Choice from	Used to assign the Choice Group or Groups that will be considered by the Decision.
Number of Choices to Select	Indicates the number of Choices that will be selected by the decision. The default and most commonly used number is 1. This is the maximum number; the actual number of Choices returned at run time may be smaller or equal to this number.
Select at random	Assigns random selection of a Choice from the Choice Groups. This is used primarily for a Control Group Decision.
Target Segments	Segment of the population that have been segmented using filtering rules. The default segment is everyone.
Priorities	Used to set the priorities for different segments by weighting performance metrics that apply to those segments.

To add a Choice Group, click **Select**, then select the Choice Group or Groups to use.

2.11.1 Segmenting population and weighting goals

Decisions can also target segments of the population and weight the performance metrics attached to that Decision for each segment. To add or remove a Segment, click **Add** and **Remove**. Click **Add** to add performance metrics to Priorities.

For instance, assume the Select Best Offer decision has two performance goals: Customer Retention and Revenue. We have also defined a segment of the population: People to retain that we have defined through filtering rules. The default remainder is the segment to which we we will cross sell.

The weighting is for each performance goal and for each segment:

People to sell to	Customer Retention Revenue	20% 80%
default	Customer Retention	90%
	Revenue	10%

When the Decision is invoked, the performance metric scoring (whether function, scoring rule, function, and so on) is applied to all of the eligible Choices. Scores are leveled using the normalization factor of the performance metrics. Scores are then weighted according to performance metric weighting contained in the decision. A total score is achieved, and the Choice with the highest score is selected.

2.11.2 Using a custom selection function

If, instead of scoring, you would like to use a custom selection function, select the **Custom Selection Function** option on the **Custom Selection Function** tab. Choose the selection function from the list, and add any parameters that the function requires.

2.11.3 Pre/post-selection logic

Scriptlets in the Pre and Post Selection tab are executed before or after the scoring is done and the Decision is made. These scriptlets typically modify the Choices in some way, or record facts.

Pre-selection logic is executed after collecting all the eligible Choices, but before the selection happens. Post-selection logic is executed after the selection, but before the selected Choices are returned. Post-selection logic is more common. You can use this to record the Decision made, or to further process the Decision.

The logic here can use the variables defined for the computation of the choices. For example, the name of the Choice array, which contains eligible Choices before and selected Choices after the selection, is set in the Pre/Post Selection tab (by default, `choices`).

Decision returns a `choiceArray`. To access the individual elements, use an index into the array. The following example reads the choiceArray, and records the base event `Delivered` to the Choice Event Model. The method `choice.recordEvent` calls the model `recordEvent`, passing in the Choice to be recorded.

```
    // SDCUSTOMCODESTART.<Classname>.PostSelectionBody.java.0

    for (int i = 0; i < outputChoiceArray.size(); i++) {

      Choice choice = outputChoiceArray.get(i);

      choice.recordEvent("Delivered");

    }

    session().addAllToPresentedOffers(outputChoiceArray); /* Store
presented offers for future reference */

    // SDCUSTOMCODEEND.<Classname>.PostSelectionBody.java.0
```

2.11.4 Selection function APIs

The type of weights parameter is GoalValues. The GoalValues class has a `getValue` method for each of the goals defined in the Decision. For example, given the goals CustomerRetention and Revenue, it has the following methods:

```
public double getValueForCustomerRetention();

public double getValueForRevenue();
```

2.11.5 Adding imported Java classes and changing the Decision Center display

To add imported Java classes to your Inline Service, click **Advanced**, next to the description. You can also change the display label for Decision Center, and choose whether the element is displayed in the Decision Center Navigator. Changing the display label does not affect the Object ID.

2.12 About selection functions

As an alternative, a *selection function* can be used by a Decision. Selection functions are like functions in that they are completely user defined. However, selection functions have a well-defined signature. They take a Choice array as input, and return a Choice array.

Selection functions have the following characteristics:

Description	Description of the selection function.
Primary Parameters	
Input Choice Array	The input parameter to the selection function. The data type of this variable is `SDChoiceArray`.
Output Choice Array	The return variable, which specifies the name of the variable that contains the selected choices and should be returned to the caller of this selection function. The return variable can be the Input Choices Array that is passed in to this selection function, or it can be another variable defined locally within the Logic panel. The data type of this variable is `SDChoiceArray`.
Number of Choices Parameter	The name of the function argument that represents the number of choices that the selection function should return. The default name of the parameter is `numChoices`. The data type of this argument is `int`.
Weights	If goals are defined for the Decision that uses this selection function, those goals are passed to the Selection function under the parameter named in Weights. The type is a `GoalValue`. For more about `GoalValue`, see the section on Decisions.
Extra Parameters	Any extra parameters the selection function needs.

Selection function scriptlets

Selection functions are used as a custom function for selection criteria. Many standard priority functions are available through templates. Priorities or selection functions are defined in Java. A set of these are predefined in the template, and usually either fill in the need, or provide an advanced prototype to modify.

Java code that does the actual selection of Choices from the list passed in as an Input Choices Array is entered in the Logic pane. Often, the Java code in the Logic section will want to refer to other classes. For the Java code and the function to compile correctly, the classes need to be imported into the function.

The `execute` method invokes the selection function.

A simple example of a selection function is shown below:

```
double maxL = -1.0;

Choice ch = null;

for (int i = 0; i < eligibleChoices.size(); i++) {

  Cross_Selling_OfferChoice cso =
(Cross_Selling_OfferChoice)eligibleChoices.get(i);

  double likelihood = cso.getLikelihood();

  if (ch == null || (!Double.isNaN(likelihood) && likelihood > maxL)) {

    maxL = likelihood;

    ch = cso;

  }
}

SDChoiceArray selectedChoices = new SDChoiceArray(1);

if (ch != null)

  selectedChoices.add(ch);
```

2.12.1 Adding imported Java classes and changing the Decision Center display

To add imported Java classes to your Inline Service, click **Advanced** next to the description. You can also change the display label for Decision Center, and choose whether the element is displayed in the Decision Center Navigator. Changing the display label does not affect the Object ID.

2.13 About analytic models

Analytic models serve two primary goals -- online prediction of business process parameters, and offline analysis of data.

For online prediction of business process parameters, models are attached to Choices and are used to predict the result of selecting a Choice with respect to a specific Key Performance Indicator (KPI).

For offline analysis, models are used to analyze the Choice targets that provide analytical data to Decision Center reports.

The main partitioning attribute for a model is typically the Choice. Additional partitioning attributes can be used to achieve strong differentiation between separate situations or populations. For example, the channel used in the interaction in a customer contact, or the country of the customer, can be used as partitioning attributes. Partitioning attributes should be used judiciously, as they multiply the amount of memory required by the model.

2.13.1 Type of models

There are three types of models:

Choice Models: Choice Models are used for data analysis on a set of mutually exclusive options. For instance, analyzing the reason for calls into a call center uses a Choice Model.

Choice Event Models: Choice Event Models are specifically designed to track choices for self learning. Choice events are events that relate to a particular choice that is either internal or external to an Inline Service.

Models: A generic, non-specific model. This model should only be used for special purposes.

2.13.2 Model algorithm

The **Algorithm** drop-down list defines the type of prediction algorithm the model will use. The *Regression* algorithm is better suited for large populations, while the *Bayesian* algorithm is better suited for small populations.

2.13.3 Model attributes

The **Choice** tab and the **Attributes** tab of Choice and Choice Event Models are different.

Attributes of the **Choice** tab and the **Attributes** tab of the Choice Model are:

Choice	
Choice Group	The name of the Choice Group to which the model applies.
Label for Choice	The label for the Choice Group.
Mutually Exclusive	Select this option if the Choices are mutually exclusive.
Attributes	
Partitioning attributes	Each analytic model consists of a number of sub-models, each corresponding to a separate combination of values of partitioning attributes. A decision to make an attribute a partitioning one has to be based on a radical difference in predicted likelihood depending on the value of the attribute in question. Attributes that have a moderate effect on predicted results should be treated as regular input, not as partitioning. The model size is very sensitive to the cardinality of partitioning attributes. Two partitioning attributes with cardinality 10 would each result in a model 25 times larger than the one that would be produced if each of the partitioning attributes had cardinality 2.
Excluded attributes	By default, all session and entity attributes are used as model inputs. To prohibit the model from using some attributes as inputs, you can add them to the Excluded Attributes list. Note that the easiest way to exclude an entity attribute from all models is to turn off the Use for analysis option in the properties of the attribute itself.
Aggregate by	By choosing one of the partitioning attributes, you can create additional sub-models for learning on the attribute. As with partitioning, a decision to aggregate on an attribute has to be based on a radical difference in predicted likelihood depending on the value of the attribute in question.

Attributes of a Choice tab and the Attributes tab of the Choice Event Model are:

Choice	
Choice Group	The name of the Choice Group to which the model applies.
Label for Choice	The label for the Choice Group.
Base Event	The base event is the Choice event used as a base for analysis.
Base Event Label	The label for the base event shown in Decision Center.
Positive Outcome events	Positive outcome events are those events that indicate a successful prediction.
Attributes	

Partitioning attributes	Each analytic model consists of a number of sub-models, each corresponding to a separate combination of values of partitioning attributes. A decision to make an attribute a partitioning one has to be based on a radical difference in predicted likelihood, depending on the value of the attribute in question. Attributes that have a moderate effect on predicted results should be treated as regular input, not as partitioning. The model size is very sensitive to the cardinality of partitioning attributes. Two partitioning attributes with cardinality 10 would each result in a model 25 times larger than the one that would be produced if each of the partitioning attributes had cardinality 2.
Excluded attributes	By default, all session and entity attributes are used as model inputs. To prohibit the model from using some attributes as inputs, you can add them to the Excluded Attributes list. Note that the easiest way to exclude an entity attribute from all models is to turn off the Use for analysis option in the properties of the attribute itself.

The Learn Location tab and the Temporary Data Storage tab are the same:

Learn location	
Learn location	By default, all models learn at the session close time. Alternatively, you can make learning happen when individual Informant or Advisor Integration Points are called. Take into account that it is usually more efficient to store all needed data in the session and pass it to the models once at the end of the session.
Temporary data storage	In business processes that take a considerable amount of time, a Choice outcome may become known a few weeks or even months after the Choice is made. Original values of the entity attributes may not be available when the Choice outcome is learned. It is desirable, however, to provide the learning model with exactly the same values of all input and partitioning attributes as they had at the moment of selecting the Choice. When the Use temporary data storage option is enabled, the model stores the values of all input and partitioning attributes whenever it receives a value of its target attribute that corresponds to the base event. The model creates one or more nearly identical database records, one for each session key included in the Keys list. When temporary data storage is enabled, it is possible to provide the model with a Choice outcome without worrying about the values of the input attributes. The values of all input model attributes are retrieved from the previously stored record.
Days to Keep	Specifies the number of days to keep temporary data stored on the server.
Keys	The data stored in temporary data storage is available for retrieval by having ANY ONE of the keys defined in the data storage tab.
Advanced	The Advanced button lets you choose to show the element in Decision Center, and to change the label of the element. Changing the label of the element does *not* change the Object ID.

2.13.4 Additional model attributes

The Use explicit base option tells the model to expect explicit notifications of decisions (base events), and assume that it will not receive notifications for one of the decision outcomes.

Choice event models are always defined with an explicit base event, so the Use explicit base option is only displayed for general-purpose models.

The target attribute is the subject of predictions. The model is able to predict likelihoods of different values of the target attribute. For model performance, it is highly recommended that you specify all possible values of the target attribute in the Possible Values field.

2.13.5 About partitioning and aggregation

Each analytic model consists of a number of sub-models, each corresponding to a separate combination of values of partitioning attributes. A decision to make an attribute a partitioning one has to be based on a radical difference in predicted likelihood, depending on the value of the attribute in question. Attributes that have a moderate effect on predicted results should be treated as regular input, not as partitioning. Model size is very sensitive to the cardinality of partitioning attributes. Two partitioning attributes with cardinality 10 each would result in a model 25 times larger than the one that would be produced if each of the partitioning attributes had cardinality 2.

By default, all session and entity attributes are used as model inputs. To prohibit the model from using some attributes as inputs, you can add them to the Excluded Attributes list. Note that if you need to exclude an entity attribute from all models, the easiest way to do so is to turn off the Use for Analysis option in the properties of the attribute itself.

One of the partitioning attributes can be marked as aggregated. This results in the creation of additional sub-models, which exclude the aggregated attribute. For example, marking the `choice` attribute as aggregated lets the model be used to predict a likelihood of positive response to any Choice.

2.13.6 Model APIs

```
public String getSDOLabel();

public String getSDOId();
```

Returns the Object label and Id, respectively.

Querying the model

The model can be queried using any of the `getChoiceEventLikelihood` methods shown below. This will return the likelihood of a Choice being chosen by the model.

```
public static SDDoubleArray getChoiceEventLikelihoods(GENOffersChoice
choice);

public static SDDoubleArray getChoiceEventLikelihoods(GENOffers
choiceGroup) ;

public static double getChoiceEventLikelihoods(GENOffersChoice choice,
String eventName);

public static double getChoiceEventLikelihoods(GENOffers choiceGroup,
String eventName);
```

Recording the Choice with the model

For the Choice Event model, the model method `recordEvent` is executed when a call to the Choice method `recordEvent` is made. Therefore, it is not necessary to directly invoke this method on the model. This method is usually called from within the Integration Point where the Choice was extended to the calling application.

For instance, in an Advisor Intgeration Point:

```
if (choices.size() > 0) {

  Choice ch = choices.get(0);

  ch.recordEvent("Presented");

  session().setOfferExtended(ch.getSDOId());

}
```

For the Choice model, the following APIs are available:

```
public static SDStringArray getChoice()

public static void setChoice(SDStringArray _v)

public static void addToChoice(String _a)

public static void addAllToChoice(SDStringArray _c)
```

The Informant usually records a Choice with the model. For instance, in a case where we are recording the Choice of a call reason code with the Model Reason Analysis:

```
if (code == 17)

  ReasonAnalysis.addToChoice("BalanceInquiry");

else if (code == 18)

  ReasonAnalysis.addToChoice("MakePayment");

else if (code == 19)

  ReasonAnalysis.addToChoice("RateInquiry");

else

  ReasonAnalysis.addToChoice("Other");
```

If the Choices were not marked mutually exclusive, this call must include a call to `getModelData()` before recording the Choice:

```
if (code == 17)

  ReasonAnalysis.getModelData().addToChoice("BalanceInquiry");

else if (code == 18)
```

```
  ReasonAnalysis.getModelData().addToChoice("MakePayment");

else if (code == 19)

  ReasonAnalysis.getModelData().addToChoice("RateInquiry");

else

  ReasonAnalysis.getModelData().addToChoice("Other");
```

If you are working with a Choice Array, you should send an empty string to the model first:

```
ReasonAnalysis.getModelData().addToChoice("");
```

2.14 About Integration Points

Integration Points perform within Oracle RTD from two points of view: data and process.

From the data point of view, Integration Points provide values to entity attributes. The Integration Point definition includes a mapping for assigning incoming values to session or entity attributes.

From the process point of view, Integration Points are defined to follow a unit as it passes through the different systems that implement the process. In general, it is best to identify the earliest point at which a unit can be identified. At that point in the process, an Informant is called by the enterprise operational systems. The system sends a request to the Informant, which enables the Inline Service to begin forming the Session and, if desired, pre-fetch information from some of the data sources.

Next, other points in the process where interesting information or measurements are known are identified, and Informants are defined for them.

Sometimes, the Inline Service runs in an observation-only mode, where there are no Advisors, just Informants. This mode is useful for gathering information about the process and measuring the non-optimized performance. In this case, Informants record observations with models so that the models can find correlations and trends in the data.

Advisors are defined for each point in the process where an Inline Intelligence Decision is to be supplied to an operational system.

An *external system* and an *order number* are also defined for each Integration Point. These are used to generate the process map presented in Decision Center. The system determines the swim-lane and the order the position, from left to right. The order can be any number, not just integers, allowing for the introduction of new Integration Points without modifying existing ones.

For information about operational systems accessing Integration Points, see *Integration with Oracle RTD.*

2.14.1 About Informants

An *Informant* publishes reports from the data and analysis gathered by its model. The targets for that analysis are Choices in one or more Choice Groups. Informants contain the logic needed to process and publish to the model. Entities act to organize data into objects for decision making and analysis.

2.14.2 About Informant functionality

Informants act in concert with Choice Groups as targets for analysis and an analytical model to perform the analysis. In general, to add an Informant to the Inline Service, do the following:

1. Create an external system to identify which system accesses the Integration Point.

2. Create a Choice Group to represent the targets for your analysis. For instance, a Choice Group may represent the reasons for calls to the service center.

3. Create an Informant that receives the session key information and gathers and processes data based on the session.

4. Create an analytical model that is the repository for the data and analyzes it.

Informants have the following characteristics:

Description	Description of the Informant.
Request	
Session Keys	One or more session keys used to identify the beginning and end of a session. Any of the session keys within the message are sufficient for identifying a session, and therefore cause the message to be dispatched to an existing session, if any, already containing information related to this message.
External System	Identifies the external system that will be sending the Informant a request. Associating the Informant with an external system allows the Informant to be displayed among other Informants and Advisors in Decision Center's process map.
Order	This number identifies the position of the Informant in the sequence of Integration Points displayed in Decision Center's process map. An Integration Point with an order less than another Integration Point's order will be displayed before the other Integration Point. The order can be a decimal number; for example, 2.1 will be displayed before 2.2.
Force Session Close	When selected, causes the Inline Service to automatically terminate the session of the Informant after all of the Informant's asynchronous logic has executed. The same effect can be achieved by placing the following Java statement anywhere in a subtab of the Informant's Logic tab: `session().close();`

Adding a session key

On the **Request** tab, click **Select** to select a session key for the Integration Point. This is one of the values that the operational system will supply to the Integration Point.

Identifying the external system and order

On the **Request** tab, use the drop-down list to choose the external system that accesses the Integration Point. This menu is populated by creating external system identifiers using the external system element.

The order in which the Integration Points are accessed is represented by **Order**. This number and the **External System** determine how the end-to-end process is displayed in Decision Center.

Adding request data

On the **Request** tab, click **Add** to add request data. Assignments are the values that the operational system will supply to the Integration Point. Assignments have the following characteristics:

Incoming Parameter	The name of the field in the request sent to the Informant whose value will be copied from the request to the session attribute. This name does not have to be the same as the session attribute; however, it generally is named the same. After the session key is created, incoming parameters are assigned to the session key attributes.
Type	This is the data type of the session attribute into which the incoming argument will be copied. The valid types are: integer, string, date, or double. Note: If the type of the request field and the session key attribute do not match, you should use a transform method.
Array	Marked if the type is a collection.
Session Attribute	The attribute of the session to which the incoming parameter of a request will be mapped.

2.14.3 Adding imported Java classes and changing the Decision Center display

To add imported Java classes to your Inline Service, click Advanced next to the description. You can also change the display label for Decision Center, and choose whether the element is displayed in the Decision Center Navigator. Changing the display label does not affect the Object ID.

2.14.4 Informant APIs

```
public String getSDOLabel();

public String getSDOId();
```

Returns the Object label and Id, respectively.

2.14.5 Informant logic

Logic

This script runs after any Request Data declared in Request Data are executed. If the primary purpose of the Informant is to transfer data from the operational system request fields to the session key and Request Data, logic may be unnecessary, as this happens automatically according to declarations in the Request Data tab.

Logic in Informants is typically used for tracing message reception in the log file, or for pre-populating entities whose keys are supplied by the Informant's message, in order to avoid having to do this later in an Advisor, where response time might be more important. Logic is executed directly following the Request Data.

Logic in the Informant can also be used to record Choices with a Choice Model. See the Choice Model APIs for methods to call.

Asynchronous Logic

This script runs after the script defined in Logic, described above. Any additional processing that needs to be done can be placed in this area. The order of execution of Asynchronous Logic is not guaranteed.

2.14.6 About models and Informants

Sometimes, the Inline Service runs in an observation-only mode, where there are no Advisors, just Informants. This mode is useful for gathering information about the process and for measuring the non-optimized performance. In this case, the logic of the Informant generally uses methods of the Choice to record events to the model.

Accessing request data from the Informant

Request data from an Informant is accessed one of several ways. If the incoming parameter is mapped to a session attribute, there is a `get` method for the parameter.

```
request.get$()
```

where $ is the parameter name with the first letter capitalized.

If the attribute is not mapped, there are methods to achieve the same results using the field name of the parameter.

```
String request.getStringValue(fieldName)

SDStringArray request.getStringArrayValue(fieldName)

boolean request.isArgPresent(fieldName)
```

2.14.7 About Advisors

Advisors are defined for each point in the process where an Inline Intelligence Decision is desired. Typically, each Advisor will make use of two specific Decision objects, one for the optimized Decision and one for the Control Group.

The Control Group decision should be as close to the existing business process as possible, so that the Optimized Decision has a basis for comparison.

In addition to the Decisions, default Choices can be defined for the Advisor. These Choices are used when the computation in the server can not be completed in time, or if the client loses communication with the server.

Choice Groups and Choices have rules. The rules determine the situations in which the Choice or Choice Group and all its descendants are available to be sent as results of the decision back to the caller of the Advisor. These rules can refer to session or entity values, or to Choice settings. For example, a choice may be available only for customers in certain groups, where the current customer group is an entity attribute and the groups is a choice attribute.

2.14.8 About the Advisor decisioning process

Advisors act in concert with Choice Groups as targets for a Decision. Scoring, in the form of functions, scoring rules, analytical models or constants, of the Choices helps decide which Choice is appropriate. The Decision then weights the scores according to Performance Goals that the organization has defined. The resulting highest-scored Choice is the Decision that is given to the Advisor as a response. For more about the Decisioning Process, see Section 2.11, About the Decision Process.

Advisors have the following characteristics:

Description	Description of the Advisor.
Request	

Session Keys	One or more session keys used to identify the beginning and end of a session. Any of the session keys within the message are sufficient for identifying a session, and hence cause the message to be dispatched to an existing session, if any, already containing information related to this message. When the Advisor is called, the session key creation is the first thing executed.
External System	Identifies the external system that will be sending the Advisor a request. Associating the Advisor with an external system allows the Advisor to be displayed among other Informants and Advisors in Decision Center's process map.
Order	This number identifies the position of the Advisor in the sequence of Integration Points displayed in Decision Center's process map. An Integration Point with an order less than another Integration Point's order will be displayed before the other Integration Point. The order can be a decimal number; for example, 2.1 will be displayed before 2.2.
Force Session Close	When selected, this option causes the Inline Service to automatically terminate the Advisor's session after all of the Advisor's asynchronous logic has executed. The same effect can be achieved by placing the following Java statement anywhere in any subtab of the Advisor's Logic tab: `session().close();`

2.14.9 Adding imported Java classes and changing the Decision Center display

To add imported Java classes to your Inline Service, click Advanced next to the description. You can also change the display label for Decision Center, and choose whether the element is displayed in the Decision Center Navigator. Changing the display label does not affect the Object ID.

2.14.10 Adding a session key

On the Request tab, click Select to select a session key for the Integration Point. This is one of the values that the operational system will supply to the Integration Point

2.14.11 Identifying the external system and order

On the Request tab, use the drop-down list to choose the external system that accesses the Integration Point. This list is populated by creating external system identifiers using the external system element.

The order in which the Integration Points are accessed is represented by Order. This number and the External System determine how the end-to-end process is displayed in Decision Center.

2.14.12 Adding request data

On the Request tab, click Add to add request data. Request data is the values that the operational system will supply to the Integration Point. Request data has the following characteristics:

Request Data	
Incoming Parameter	The name of the field in the request sent to the Advisor whose value will be copied from the request to the session attribute. This name does not have to be the same as the Session attribute; however it generally is named the same. After the session key is created, the assignment of incoming parameters to session attributes is made.
Type	This is the data type of the session attribute into which the incoming argument will be copied. The valid types are: integer, string, date or double. Note: If the type of the request field and the session attribute do not match, you should use a transform method.
Array	Select this option if the type is a collection.
Session Attribute	The attribute of the session to which the incoming parameter of a request is mapped.

2.14.13 Adding response data

On the Response tab, click Add to add response data. Response data is the values that the operational system will send back to the Integration Point after a request is invoked. Response data has the following characteristics:

Response	The response contains an array of selected Choice objects, with each Choice containing a collection of named attribute values. The Choice selection process is governed by one of two Decision objects referenced by the Advisor. One Decision is given to the calling application.
Decision to Use	The name of the Decision object to use for normal sessions, as opposed to control-group sessions. This Decision becomes the Advisor's response to the calling system.
Control Group Decision to Use	Control Group Decision is used for only a small percentage of sessions as a way to assess the effectiveness of the other Decisions by providing a baseline. The percentage of sessions that use the control-group decision is specified in the Application element of the Inline Service. The Control Group Decision should be designed to select choices using "business-as-usual" logic, meaning whatever rules the enterprise previously used before introducing the Inline Service. Reports are available through the Decision Center console that compare the business effectiveness of the Advisor's normal Decision object with its Control Group Decision.
Parameters	Input parameter defined by the decision. The Name and Type columns are descriptive only, surfaced here from the Decision object.
Default number of choices returned	Default number of choices returned by the decision. This is the number of choices defined by the Decision.
Override default with	The Advisor can override or accept the number specified by the referenced Decision. This area specifies the maximum number of qualified choices to be included in the Advisor's response.

Default Choices	A list of Choices that are returned to the calling client application whenever it tries to invoke this Advisor, and the Advisor is not able to deliver its response within the server's guaranteed response time. Note that default Choices do not have to be specified for each Advisor. The Inline Service may also declare default Choices, which are used for Advisors that don't declare their own. Also note that the default Choice configuration is propagated to the client application and stored in the local file system by the Smart Client component. Hence, it is subsequently available to client applications that cannot connect to the server.

2.14.14 Logic in Advisors

Logic

This script runs after any request data declared in the request data tab is executed, and before the response is sent back to the client.

Advisor logic is generally not needed. You may want to use it for preprocessing data coming in with the request, or for debugging purposes.

Asynchronous Logic

This script runs after the response has been handed off to the server-side mechanism that sends it back to the client. Depending on the type of endpoint used by the client, the client may be able to start processing the result before this script finishes, thus improving the effective response time by increasing parallelism.

2.14.15 Accessing request data from the Advisor

Request data from an Advisor is accessed in one of several ways. If the incoming parameter is mapped to a session attribute, there is a `get` method for the parameter.

```
request.get$()
```

where $ is the parameter name with the first letter capitalized.

If the attribute is not mapped, there are several methods to achieve the same results.

```
   String request.getStringValue(fieldName)

   SDStringArray request.getStringArrayValue(fieldName)

   boolean request.isArgPresent(fieldName)
```

2.15 About external systems

External systems are only identified within Decision Studio. The external system represents the operational systems within the enterprise that integrate to the Inline Service. The external system is not accessible through the API. The external system is used by an Integration Point to identify which external system will access that Integration Point. External systems are used for display on the Integration Map in Decision Center.

External systems have the following characteristics:

Description	Description of the system.
Display Label	Lets you change the Display Label. This does *not* affect the Object ID.

2.16 About the categories object

Categories are used to organize Choices. All Choices of the same category appear together in Decision Center. No classes are generated for categories. They are only used by Decision Center for grouping and organizing Choices.

Categories have the following characteristics:

Name	Name of the category, as entered in Decision Studio.
Description	Description of the category, as entered in Decision Studio.
Display Label	Lets you change the Display Label. This does *not* affect the Object ID.

2.17 About functions

Functions can be used for calculation or for other processing that you want to make reusable. Functions are defined using Decision Studio. Functions are defined with the following characteristics:

Description	Description of the function.
Return value	Specifies whether the function returns a value.
Data Type	Type of the returned value.
Array	Select this option if the return type is an array.
Call Template	The definition of how the function will be called. Using `{0}`, `{1}`, and so on as arguments, and phrasing to describe the function, define the template for call. It is important to use good phrasing, as this is what will be shown when using the function. For instance, a call template for multiply is `{0} multiplied by {1}`.
Parameters	Named parameters that will be used in the logic of the function. This number must match the number of arguments in the call template. For instance, multiply has the following parameters: `a, type Double; b, type Double`
Logic	Java code for the function. The code for multiply is: `return a * b;`

2.17.1 Adding imported Java classes and changing the Decision Center display label

To add imported Java classes to your Inline Service, click **Advanced** next to the description. You can also change the display label for Decision Center, and choose whether the element is displayed in the Decision Center Navigator. Changing the display label does not affect the Object ID.

Functions are called from other elements using the call template. For instance, if you wanted to use the `multiply` function described above, you would choose the function from the **Edit Value** dialog. The call template `{0} multiplied by {1}` provides the editor with the position and number of arguments.

2.18 About statistic collectors

Statistic collectors manage the collection and lifecycle of Inline Service event statistics. A Choice Event Statistics Collector is created for each Inline Service. Choice Event Statistics Collectors automatically collect statistics for the events defined by your Inline Service. Statistics collectors have the following properties:

Description	The description of the statistics collector.
Collect Statistics On	Statistics can be collected either for each object, such as Choice or Choice Group, individually, or aggregated for all objects of the same type.
Aggregation	Either record individual events, or record aggregated data. Care should be used in recording individual events, as high transactional systems may suffer from performance issues.
Aggregation Interval	Amount of time in seconds to aggregate data before recording it.
Expiration	Choose either **Keep forever** or **Purge old statistics**. Care should be used in choosing **Keep forever**, as data size can be an issue.
Keep in database for	Amount of time in days that data is kept before purging.

All parameters are configurable through the Decision Studio editor. Choice Event Statistics are displayed as a report in Decision Center.

2.18.1 Creating a custom statistics collector

Using Decision Studio, you can create a custom statistics collector to record additional statistics about objects or classes. For instance, you can create a statistics collector to record statistics about Choices. Configure the parameters as described in the previous section.

In code in your Inline Service (for instance, in an Informant or through a Function Call), create a Statistics Collector Factory, passing in the Collector Name (String) or the statistic type (String):

```
StatisticCollectorFactory factory = Application.getCollectorFactory(<stat
collector name | statistic type>);
```

Using the factory, create a collector, passing in the event name on which you want to collect statistics (String) or the statistic name (String):

```
StatCollectorInterface collector = factory.getCollector(<event name |
statistic name> );
```

The event name or statistic name is an arbitrary string that represents what you want to collect.

Then, finally, using the collector, record the event passing in the object_type (String), object_id (String), event value (double), and extra_data (string) to record:

```
Collector.recordEvent(<object_type>, <object_id>, event value, extra
data);
```

The object_type must be a valid Object type, such as Choice, Choice Group, Entity, and so on. The object_id is the internal name of the object.

2.19 About Decision Center perspectives

Like Decision Studio, Decision Center lets you work with an Inline Service from several *perspectives*. A perspective defines the initial layout in Decision Center. Each perspective provides a set of functionality aimed at accomplishing a specific type of task, and works with specific types of resources. Perspectives control what appears in certain menus and toolbars.

Decision Center has three default perspectives: Explore, Design, and At a Glance. The Inline Service Navigator changes according to the perspective you are using. Your system administrator may have added additional perspectives.

You can control access to Decision Center perspectives by assigning permissions to groups and users. See *Installation and Administration of Oracle RTD* for information about managing groups and users.

To assign permissions for perspectives, go to the Inline Service Navigator in Decision Studio and right-click the perspective for which you want to set access. Choose Properties, then click Add. Select the group to which you want to assign permissions, then select Use perspective under Permissions. Click OK to finish.

Section 3: Oracle RTD General APIs

This section describes the general APIs for Oracle RTD.

This section contains the following topics:

3.1 com.sigmadynmics.util Class Null

A utility class called Null that tests for Null.

isNull

public static boolean isNull(String val)

Parameters:

val – The String value to be tested for null

Returns:

true or false depending on whether the value is null or not.

isNull

public static boolean isNull(boolean val)

Parameters:

val – The boolean value to be tested for null

Returns:

true or false depending on whether the value is null or not.

isNull

public static boolean isNull(long val)

Parameters:

val – The long value to be tested for null

Returns:

true or false depending on whether the value is null or not.

isNull

public static boolean isNull(float val)

Parameters:

val – The float value to be tested for null

Returns:

true or false depending on whether the value is null or not.

isNull

public static boolean isNull(int val)

Parameters:

val – The int value to be tested for null

Returns:

true or false depending on whether the value is null or not.

Object

3.2 com.sigmadynamics.support Class SDOBase

Methods for logging are defined in the SDO base class. These methods are available for use throughout any Inline Service.

Three levels of logging are available: Info, Warning and Debug. By default the Info level is enabled and available in the Inline Service and goes to the Error Log. To enable the other levels, use JConsole. For more about using JConsole, see *Installation and Administration of Oracle RTD*.

---

logDebug

public static void logDebug(String msg)

Logs a String message at the debugging level.

Parameters:

msg – the String value to log.

---

logDebug

public static void logDebug(String msg, Object[] args)

Logs a String message and an array of Objects at the debugging level.

Parameters:

msg – the String value to log.

args – the Object array to log. Object arrays are first converted to string values using toString.

---

logDebug

public static void logDebug(String msg, Throwable t)

Logs a String message and an exception at the debugging level.

Parameters:

msg – the String value to log.

t – the exception to be logged.

---

logDebug

public static void logDebug(String msg, Object[] args, Throwable t)

Logs a String message, an array of Objects and an exception at the debugging level.

Parameters:

msg – the String value to log.

args – the Object array to log. Object arrays are first converted to string values using toString.

t - the exception to be logged.

---

logInfo

public static void logInfo (String msg)

Logs a String message at the informational level.

Parameters:

msg – the String value to log.

---

logInfo

public static void logInfo(String msg, Object[] args)

Logs a String message and an array of Objects at the informational level.

Parameters:

msg – the String value to log.

args – the Object array to log. Object arrays are first converted to string values using toString.

---

logInfo

public static void logInfo(String msg, Throwable t)

Logs a String message and an exception at the informational level.

Parameters:

msg – the String value to log.

t – the exception to be logged.

---

logInfo

public static void logInfo(String msg, Object[] args, Throwable t)

Logs a String message , an array of Objects and an excepton at the informational level.

Parameters:

msg – the String value to log.

args – the Object array to log. Object arrays are first converted to string values using toString.

t - the exception to be logged.

logWarning

public static void logWarning(String msg)

Logs a String message at the warning level.

Parameters:

msg – the String value to log.

logWarning

public static void logWarning(String msg, Object[] args)

Logs a String message and an array of Objects at the warning level.

Parameters:

msg – the String value to log.

args – the Object array to log. Object arrays are first converted to string values using toString.

logWarning

public static void logWarning(String msg, Throwable t)

Logs a String message and an exception at the warning level.

Parameters:

msg – the String value to log.

t – the exception to be logged.

logWarning

public static void logDebug(String msg, Object[] args, Throwable t)

Logs a String message, an array of Objects and an exception at the warning level.

Parameters:

msg – the String value to log.

args – the Object array to log. Object arrays are first converted to string values using toString.

t - the exception to be logged.

---

logError

public static void logError(String msg)

Logs a String message at the error level.

Parameters:

msg – the String value to log.

---

logError

public static void logError(String msg, Object[] args)

Logs a String message and an array of Objects at the warning level.

Parameters:

msg – the String value to log.

args – the Object array to log. Object arrays are first converted to string values using toString.

---

logError

public static void logError(String msg, Throwable t)

Logs a String message and an exception at the error level.

Parameters:

msg – the String value to log.

t – the exception to be logged.

---

logError

public static void logError (String msg, Object[] args, Throwable t)

Logs a String message, an array of Objects and an exception at the error level.

Parameters:

msg – the String value to log.

args – the Object array to log. Object arrays are first converted to string values using toString.

t - the exception to be logged.

---

logError

public static void logError (Throwable t)

Logs an exception at the error level.

Parameters:

t - the exception to be logged.

3.3 com.sigmadynamics.util Class StringUtil

The StringUtil class contains methods that return various types in the form of a readable string. This is useful for debugging.

3.4 com.sigmadynamics.util Class DateUtil

The DateUtil class contains methods that return a date formatted in various ways.

---

getCalendar

public static Calendar **getCalendar**(long datetime)

Converts date and time in milliseconds since January 1, 1970 UTC to Calendar object.

Parameters:

datetime - Date and time in milliseconds since January 1, 1970 UTC.

Returns:

A Calendar object representing the date.

See Also: Calendar

---

formatDate

public static String formatDate(long date)

Converts date in milliseconds since January 1, 1970 UTC to a string representation.

Parameters:

date - Date in milliseconds since January 1, 1970 UTC.

Returns:

The date is short string form, e.g. "4/21/03" in the US locale.

---

formatTime

public static String formatTime(long time)

Converts time in milliseconds since January 1, 1970 UTC to a string representation.

Parameters:

time - Time in milliseconds since January 1, 1970 UTC.

Returns:

The time value is short string form, e.g. "8:35 pm" in the US locale.

---

formatDateTime

public static String formatDateTime(long datetime)

Converts date and time in milliseconds since January 1, 1970 UTC to a string representation.

Parameters:

datetime - Date and time in milliseconds since January 1, 1970 UTC.

Returns:

The date and time is short string form, e.g. "4/21/03 8:35 pm" in the US locale.

---

format

public static String format(long datetime, DateFormat format)

Formats date and time in milliseconds since January 1, 1970 UTC according to a supplied format specification.

Parameters:

datetime - Date and time in milliseconds since January 1, 1970 UTC.

format - The format specification.

Returns: The formatted date/time string.

See Also: DateFormat

3.5 com.sigmadynamics.util Class SDArray classes

Various array classes are provided. The base class for these is SDArray.

---

public abstract class SDArray

extends Object

implements Serializable

This abstract class serves as a base for specialized SDArrayType classes.

Constant Field Values

protected static final int ALLOC_UNIT

protected static final int DEFAULT_INITIAL_SIZE

protected int size

Constructor Detail

SDArray

public SDArray()

Default constructor.

---

size

public final int size()Returns the number of elements in the array.

Returns:

the number of elements.

---

isEmpty

public final boolean isEmpty()

Returns true if the array does not contain any elements, or false otherwise.

Returns:

true or false depending on whether the array is empty or not.

---

capacity

public abstract int capacity()

Returns:

buffer capacity.

setSize

public abstract void setSize(int size)

Adds or deletes elements at the end of the array.

Parameters:

size - new number or elements.

clear

public void clear()

Deletes all elements from the array.

trimToSize

public abstract void trimToSize()

Removes any excess buffer capacity above the actual number of elements.

ensureCapacity

protected abstract void ensureCapacity(int capacity)

Grows the internal buffer as needed to accommodate the specified number of elements.

Parameters:

capacity - number of elements to accommodate.

3.6 com.sigmadynamics.util Class SDBooleanArray

A type safe implementation of a resizable array containing elements of type boolean.

public class SDBooleanArray

extends SDArray

Field Detail

buf

protected boolean[] buf

Constructor Detail

SDBooleanArray

public SDBooleanArray()

Constructs an empty array.

SDBooleanArray

public SDBooleanArray(int capacity)

Constructs an empty array with the specified initial capacity.

Parameters:

capacity - the initial capacity of the array.

capacity

public int capacity()

Returns:

buffer capacity.

Specified by:

capacity in class SDArray

Returns:

buffer capacity.

get

public boolean get(int index)

Returns:

the element at the specified position in this array.

Parameters:

index - index of the element to return.

set

public void set(int index, boolean val)

Replaces the element at the specified position in this array with the specified element.

Parameters:

index - index of the element to replace.

add

public void add(boolean element)

Appends the specified element to the end of this array.

Parameters:

element - element to be appended to this array.

addAll

public void addAll(SDBooleanArray array)

Appends all of the elements in the specified array to the end of this array.

Parameters:

array - array containing elements to be appended to this array.

fill

public void fill(boolean element)

Replaces all elements of this array with the same value.

Parameters:

element - element that replaces all elements of this array.

---

fill

public void fill(int fromIndex, int toIndex, boolean element)

Replaces a group of consecutive elements of this array with the same value. The group of elements to be replaced consists of elements whose index is between fromIndex, inclusive, and toIndex, exclusive.

Parameters:

fromIndex - index of the first element to be replaced.

toIndex - index after the last element to be replaced.

element - element that replaces the elements from fromIndex to toIndex.

---

contains

public boolean contains(boolean element)

Returns true if this array contains the specified element.

Parameters:

element - element whose presence in this array is to be tested.

Returns:

true if the specified element is present; false otherwise.

---

containsAll

public boolean containsAll(SDBooleanArray elements)

Returns true if this array contains all elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if all specified elements is present in this array; false otherwise.

---

containsAny

public boolean containsAny(SDBooleanArray elements)

Returns true if this array contains any one of the elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if any specified elements are present in this array; false otherwise.

---

sort

public void sort()

Sorts the array in ascending order.

---

equals

public boolean equals(Object anObject)

Element by element comparison.

Parameters:

anObject - an object to compare this array with.

Returns:

true if the arrays are equal, false otherwise.

---

toArray

public boolean[] toArray()

Returns an array of boolean containing all elements of this one.

Returns:

an array of boolean containing all elements of this one.

---

toArray

public void toArray(boolean[] array)

Copies elements to a given array. If the supplied array is shorter than the number of elements in this one, only the elements that fit the supplied array are copied. If the supplied array is longer than the number of elements in this one, the rear part of the supplied array remains unchanged.

Parameters:

array - the array to which the elements of this one are to be copied.

Throws:

ArrayStoreException - if the runtime type of the supplied array is not a supertype of the runtime type of every element in this one.

---

setSize

public void setSize(int size)

Adds or deletes elements at the end of the array.

Specified by:

setSize in class SDArray

Parameters:

size - new number or elements.

Throws:

IllegalArgumentException - if size < 0.

---

trimToSize

public void trimToSize()

Removes any excess buffer capacity above the actual number of elements.

Specified by:

trimToSize in class SDArray

---

toString

public String toString()

Returns a string representation of this array. The string representation consists of elements of the array enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (comma and space). Elements are converted to strings by String.valueOf(boolean).

Returns:

a string representation of this array.

---

indexOf

public static int indexOf(boolean element, boolean[] array)

Returns the index in an array of the first occurrence of the specified element, or -1 if the array does not contain such element. This method uses linear search.

---

ensureCapacity

protected void ensureCapacity(int capacity)

Grows the internal buffer as needed to accommodate the specified number of elements.

Specified by:

ensureCapacity in class SDArray

Parameters:

capacity - number of elements to accommodate.

3.7 com.sigmadynamics.util Class SDDoubleArray

Type safe implementation of resizable array containing elements of type double.

---

public class SDDoubleArray

extends SDArray

Field Detail

buf

protected double[] buf

Constructor Detail

SDDoubleArray

public SDDoubleArray()

Constructs an empty array.

SDDoubleArray

public SDDoubleArray(int capacity)

Constructs an empty array with the specified initial capacity.

Parameters:

capacity - the initial capacity of the array.

---

capacity

public int capacity()

Returns buffer capacity.

Specified by:

capacity in class SDArray

Returns:

buffer capacity.

---

get

public double get(int index)

Returns the element at the specified position in this array.

Parameters:

index - index of the element to return.

Returns:

The element at the specified position in this array.

---

set

public void set(int index, double val)

Replaces the element at the specified position in this array with the specified element.

Parameters:

index - index of the element to replace.

---

add

public void add(double element)

Appends the specified element to the end of this array.

Parameters:

element - element to be appended to this array.

---

addAll

public void addAll(SDDoubleArray array)

Appends all of the elements in the specified array to the end of this array.

Parameters:

array - array containing elements to be appended to this array.

---

fill

public void fill(double element)

Replaces all elements of this array with the same value.

Parameters:

element - element that replaces all elements of this array.

---

fill

public void fill(int fromIndex, int toIndex, double element)

Replaces a group of consecutive elements of this array with the same value. The group of elements to be replaced consists of elements whose index is between fromIndex, inclusive, and toIndex, exclusive.

Parameters:

fromIndex - index of the first element to be replaced.

toIndex - index after the last element to be replaced.

element - element that replaces the elements from fromIndex to toIndex.

contains

public boolean contains(double element)

Returns true if this array contains the specified element.

Parameters:

element - element whose presence in this array is to be tested.

Returns:

true if the specified element is present; false otherwise.

containsAll

public boolean containsAll(SDDoubleArray elements)

Returns true if this array contains all elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if all specified elements is present in this array; false otherwise.

containsAny

public boolean containsAny(SDDoubleArray elements)

Returns true if this array contains any one of the elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if any specified elements are present in this array; false otherwise.

sort

public void sort()

Sorts the array in ascending order.

equals

public boolean equals(Object anObject)

Element by element comparison.

Parameters:

anObject - an object to compare this array with.

Returns:

true if the arrays are equal, false otherwise.

toArray

public double[] toArray()

Returns an array of double containing all elements of this one.

Returns:

an array of double containing all elements of this one.

toArray

public void toArray(double[] array)

Copies elements to a given array. If the supplied array is shorter than the number of elements in this one, only the elements that fit the supplied array are copied. If the supplied array is longer than the number of elements in this one, the rear part of the supplied array remains unchanged.

Parameters:

array - the array to which the elements of this one are to be copied.

Throws:

ArrayStoreException - if the runtime type of the supplied array is not a supertype of the runtime type of every element in this one.

setSize

public void setSize(int size)

Adds or deletes elements at the end of the array.

Specified by:

setSize in class SDArray

Parameters:

size - new number or elements.

Throws:

IllegalArgumentException - if size < 0.

---

trimToSize

public void trimToSize()

Removes any excess buffer capacity above the actual number of elements.

Specified by:

trimToSize in class SDArray

---

toString

public String toString()

Returns a string representation of this array. The string representation consists of elements of the array enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (comma and space). Elements are converted to strings by String.valueOf(double).

Returns:

a string representation of this array.

---

indexOf

public static int indexOf(double element, double[] array)

Returns the index in an array of the first occurrence of the specified element, or -1 if the array does not contain such element. This method uses linear search.

---

ensureCapacity

protected void ensureCapacity(int capacity)

Grows the internal buffer as needed to accommodate the specified number of elements.

Specified by:

ensureCapacity in class SDArray

Parameters:

capacity - number of elements to accommodate.

---

increment

public double increment(int index, double amount)

Increments an element by a given amount.

Parameters:

index - element index.

amount - a value to add to the element.

Returns:

the incremented value.

---

decrement

public double decrement(int index, double amount)

Decrements an element by a given amount.

Parameters:

index - element index.

amount - a value to add to the element.

Returns:

the incremented value.

3.8 com.sigmadynamics.util Class SDIntArray

Type safe implementation of resizable array containing elements of type integer.

---

public class SDIntArray

extends SDArray

Field Detail

buf

protected double[] buf

Constructor Detail

SDIntArray

public SDIntArray ()

Constructs an empty array.

SDIntArray

public SDIntArray (int capacity)

Constructs an empty array with the specified initial capacity.

Parameters:

capacity - the initial capacity of the array.

SDIntArray

public SDIntArray (int[] elements)

This constructor is not a part of the public API and should not be used by Inline Service code.

capacity

public int capacity()

Returns buffer capacity.

Specified by:

capacity in class SDArray

Returns:

buffer capacity.

get

public double get(int index)

Returns the element at the specified position in this array.

Parameters:

index - index of the element to return.

Returns:

The element at the specified position in this array.

---

set

public void set(int index, int val)

Replaces the element at the specified position in this array with the specified element.

Parameters:

index - index of the element to replace.

val - value.

---

add

public void add(int element)

Appends the specified element to the end of this array.

Parameters:

element - element to be appended to this array.

---

addAll

public void addAll(SDIntArray array)

Appends all of the elements in the specified array to the end of this array.

Parameters:

array - array containing elements to be appended to this array.

---

fill

public void fill(int element)

Replaces all elements of this array with the same value.

Parameters:

element - element that replaces all elements of this array.

---

fill

public void fill(int fromIndex, int toIndex, int element)

Replaces a group of consecutive elements of this array with the same value. The group of elements to be replaced consists of elements whose index is between fromIndex, inclusive, and toIndex, exclusive.

Parameters:

fromIndex - index of the first element to be replaced.

toIndex - index after the last element to be replaced.

element - element that replaces the elements from fromIndex to toIndex.

---

contains

public boolean contains(int element)

Returns true if this array contains the specified element.

Parameters:

element - element whose presence in this array is to be tested.

Returns:

true if the specified element is present; false otherwise.

---

containsAll

public boolean containsAll(SDIntArray elements)

Returns true if this array contains all elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if all specified elements is present in this array; false otherwise.

containsAny

public boolean containsAny(SDIntArray elements)

Returns true if this array contains any one of the elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if any specified elements are present in this array; false otherwise.

sort

public void sort()

Sorts the array in ascending order.

equals

public boolean equals(Object anObject)

Element by element comparison.

Parameters:

anObject - an object to compare this array with.

Returns:

true if the arrays are equal, false otherwise.

toArray

public int[] toArray()

Returns an array of double containing all elements of this one.

Returns:

an array of double containing all elements of this one.

toArray

public void toArray(int[] array)

Copies elements to a given array. If the supplied array is shorter than the number of elements in this one, only the elements that fit the supplied array are copied. If the supplied array is longer than the number of elements in this one, the rear part of the supplied array remains unchanged.

Parameters:

array - the array to which the elements of this one are to be copied.

Throws:

ArrayStoreException - if the runtime type of the supplied array is not a supertype of the runtime type of every element in this one.

---

setSize

public void setSize(int size)

Adds or deletes elements at the end of the array.

Specified by:

setSize in class SDArray

Parameters:

size - new number or elements.

Throws:

IllegalArgumentException - if size < 0.

---

trimToSize

public void trimToSize()

Removes any excess buffer capacity above the actual number of elements.

Specified by:

trimToSize in class SDArray

---

toString

public String toString()

Returns a string representation of this array. The string representation consists of elements of the array enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (comma and space). Elements are converted to strings by String.valueOf(int).

Returns:

a string representation of this array.

---

indexOf

public static int indexOf(int element, int[] array)

Returns the index in an array of the first occurrence of the specified element, or -1 if the array does not contain such element. This method uses linear search.

---

ensureCapacity

protected void ensureCapacity(int capacity)

Grows the internal buffer as needed to accommodate the specified number of elements.

Specified by:

ensureCapacity in class SDArray

Parameters:

capacity - number of elements to accommodate.

---

increment

public double increment(int index, int amount)

Increments an element by a given amount.

Parameters:

index - element index.

amount - a value to add to the element.

Returns:

the incremented value.

---

decrement

public double decrement(int index, int amount)

Decrements an element by a given amount.

Parameters:

index - element index.

amount - a value to add to the element.

Returns:

the incremented value.

3.9 com.sigmadynamics.util Class SDLongArray

Type safe implementation of resizable array containing elements of type long.

public class SDLongArray

extends SDArray

Field Detail

buf

protected double[] buf

Constructor Detail

SDLongArray

public SDLongArray ()

Constructs an empty array.

SDLongArray

public SDLongArray (int capacity)

Constructs an empty array with the specified initial capacity.

Parameters:

capacity - the initial capacity of the array.

SDLongArray

public SDLongArray (int[] elements)

This constructor is not a part of public API and should not be used by Inline Service code.

---

capacity

public int capacity()

Returns buffer capacity.

Specified by:

capacity in class SDArray

Returns:

buffer capacity.

---

get

public double get(int index)

Returns the element at the specified position in this array.

Parameters:

index - index of the element to return.

Returns:

The element at the specified position in this array.

---

set

public void set(int index, long val)

Replaces the element at the specified position in this array with the specified element.

Parameters:

index - index of the element to replace.

val - value.

---

add

public void add(long element)

Appends the specified element to the end of this array.

Parameters:

element - element to be appended to this array.

---

addAll

public void addAll(SDLongArray array)

Appends all of the elements in the specified array to the end of this array.

Parameters:

array - array containing elements to be appended to this array.

---

fill

public void fill(long element)

Replaces all elements of this array with the same value.

Parameters:

element - element that replaces all elements of this array.

---

fill

public void fill(int fromIndex, int toIndex, long element)

Replaces a group of consecutive elements of this array with the same value. The group of elements to be replaced consists of elements whose index is between fromIndex, inclusive, and toIndex, exclusive.

Parameters:

fromIndex - index of the first element to be replaced.

toIndex - index after the last element to be replaced.

element - element that replaces the elements from fromIndex to toIndex.

---

contains

public boolean contains(long element)

Returns true if this array contains the specified element.

Parameters:

element - element whose presence in this array is to be tested.

Returns:

true if the specified element is present; false otherwise.

---

containsAll

public boolean containsAll(SDLongArray elements)

Returns true if this array contains all elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if all specified elements is present in this array; false otherwise.

---

containsAny

public boolean containsAny(SDLongArray elements)

Returns true if this array contains any one of the elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if any specified elements are present in this array; false otherwise.

---

sort

public void sort()

Sorts the array in ascending order.

---

equals

public boolean equals(Object anObject)

Element by element comparison.

Parameters:

anObject - an object to compare this array with.

Returns:

true if the arrays are equal, false otherwise.

---

toArray

public long[] toArray()

Returns an array of long containing all elements of this one.

Returns:

an array of double containing all elements of this one.

---

toArray

public void toArray(long[] array)

Copies elements to a given array. If the supplied array is shorter than the number of elements in this one, only the elements that fit the supplied array are copied. If the supplied array is longer than the number of elements in this one, the rear part of the supplied array remains unchanged.

Parameters:

array - the array to which the elements of this one are to be copied.

Throws:

ArrayStoreException - if the runtime type of the supplied array is not a supertype of the runtime type of every element in this one.

---

setSize

public void setSize(int size)

Adds or deletes elements at the end of the array.

Specified by:

setSize in class SDArray

Parameters:

size - new number or elements.

Throws:

IllegalArgumentException - if size < 0.

---

trimToSize

public void trimToSize()

Removes any excess buffer capacity above the actual number of elements.

Specified by:

trimToSize in class SDArray

---

toString

public String toString()

Returns a string representation of this array. The string representation consists of elements of the array enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (comma and space). Elements are converted to strings by String.valueOf(long).

Returns:

a string representation of this array.

---

indexOf

public static int indexOf(long element, long[] array)

Returns the index in an array of the first occurrence of the specified element, or -1 if the array does not contain such element. This method uses linear search.

---

ensureCapacity

protected void ensureCapacity(int capacity)

Grows the internal buffer as needed to accommodate the specified number of elements.

Specified by:

ensureCapacity in class SDArray

Parameters:

capacity - number of elements to accommodate.

3.10 com.sigmadynamics.util Class SDStringArray

Type-safe implementation of resizable array containing elements of type String.

public class SDStringArray

extends SDArray

Field Detail

buf

protected double[] buf

Constructor Detail

SDStringArray

public SDStringArray ()

Constructs an empty array.

SDStringArray

public SDStringArray (int capacity)

Constructs an empty array with the specified initial capacity.

Parameters:

capacity - the initial capacity of the array.

SDLongArray

public SDLongArray (int[] elements)

This constructor is not a part of public API and should not be used by Inline Service code.

capacity

public int capacity()

Returns buffer capacity.

Specified by:

capacity in class SDArray

Returns:

buffer capacity.

---

get

public double get(int index)

Returns the element at the specified position in this array.

Parameters:

index - index of the element to return.

Returns:

The element at the specified position in this array.

---

set

public void set(int index, String val)

Replaces the element at the specified position in this array with the specified element.

Parameters:

index - index of the element to replace.

val - value.

---

add

public void add(String element)

Appends the specified element to the end of this array.

Parameters:

element - element to be appended to this array.

---

addAll

public void addAll(SDStringArray array)

Appends all of the elements in the specified array to the end of this array.

Parameters:

array - array containing elements to be appended to this array.

---

fill

public void fill(String element)

Replaces all elements of this array with the same value.

Parameters:

element - element that replaces all elements of this array.

---

fill

public void fill(int fromIndex, int toIndex, String element)

Replaces a group of consecutive elements of this array with the same value. The group of elements to be replaced consists of elements whose index is between fromIndex, inclusive, and toIndex, exclusive.

Parameters:

fromIndex - index of the first element to be replaced.

toIndex - index after the last element to be replaced.

element - element that replaces the elements from fromIndex to toIndex.

---

contains

public boolean contains(String element)

Returns true if this array contains the specified element.

Parameters:

element - element whose presence in this array is to be tested.

Returns:

true if the specified element is present; false otherwise.

---

containsAll

public boolean containsAll(SDStringArray elements)

Returns true if this array contains all elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if all specified elements is present in this array; false otherwise.

---

containsAny

public boolean containsAny(SDStringArray elements)

Returns true if this array contains any one of the elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if any specified elements are present in this array; false otherwise.

---

sort

public void sort()

Sorts the array in ascending order.

---

equals

public boolean equals(Object anObject)

Element by element comparison.

Parameters:

anObject - an object to compare this array with.

Returns:

true if the arrays are equal, false otherwise.

---

toArray

public String[] toArray()

Returns an array of String containing all elements of this one.

Returns:

an array of double containing all elements of this one.

---

toArray

public void toArray(String[] array)

Copies elements to a given array. If the supplied array is shorter than the number of elements in this one, only the elements that fit the supplied array are copied. If the supplied array is longer than the number of elements in this one, the rear part of the supplied array remains unchanged.

Parameters:

array - the array to which the elements of this one are to be copied.

Throws:

ArrayStoreException - if the runtime type of the supplied array is not a supertype of the runtime type of every element in this one.

---

setSize

public void setSize(int size)

Adds or deletes elements at the end of the array.

Specified by:

setSize in class SDArray

Parameters:

size - new number or elements.

Throws:

IllegalArgumentException - if size < 0.

---

trimToSize

public void trimToSize()

Removes any excess buffer capacity above the actual number of elements.

Specified by:

trimToSize in class SDArray

---

toString

public String toString()

Returns a string representation of this array. The string representation consists of elements of the array enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (comma and space).

Returns:

a string representation of this array.

---

indexOf

public static int indexOf(String element, String[] array)

Returns the index in an array of the first occurrence of the specified element, or -1 if the array does not contain such element. This method uses linear search.

---

ensureCapacity

protected void ensureCapacity(int capacity)

Grows the internal buffer as needed to accommodate the specified number of elements.

Specified by:

ensureCapacity in class SDArray

Parameters:

capacity - number of elements to accommodate.

3.11 com.sigmadynamics.util Class SDStringArray

Type-safe implementation of resizable array containing elements of type Object.

---

public class SDObjectArray

extends SDArray

Field Detail

buf

protected double[] buf

Constructor Detail

SDObjectArray

public SDObjectArray()

Constructs an empty array.

SDObjectArray

public SDObjectArray (int capacity)

Constructs an empty array with the specified initial capacity.

Parameters:

capacity - the initial capacity of the array.

---

capacity

public int capacity()

Returns buffer capacity.

Specified by:

capacity in class SDArray

Returns:

buffer capacity.

---

get

public double get(int index)

Returns the element at the specified position in this array.

Parameters:

index - index of the element to return.

Returns:

The element at the specified position in this array.

set

public void set(int index, Object val)

Replaces the element at the specified position in this array with the specified element.

Parameters:

index - index of the element to replace.

val - value.

add

public void add(Object element)

Appends the specified element to the end of this array.

Parameters:

element - element to be appended to this array.

addAll

public void addAll(SDObjectArray array)

Appends all of the elements in the specified array to the end of this array.

Parameters:

array - array containing elements to be appended to this array.

fill

public void fill(Object element)

Replaces all elements of this array with the same value.

Parameters:

element - element that replaces all elements of this array.

fill

public void fill(int fromIndex, int toIndex, Object element)

Replaces a group of consecutive elements of this array with the same value. The group of elements to be replaced consists of elements whose index is between fromIndex, inclusive, and toIndex, exclusive.

Parameters:

fromIndex - index of the first element to be replaced.

toIndex - index after the last element to be replaced.

element - element that replaces the elements from fromIndex to toIndex.

---

contains

public boolean contains(Object element)

Returns true if this array contains the specified element.

Parameters:

element - element whose presence in this array is to be tested.

Returns:

true if the specified element is present; false otherwise.

---

containsAll

public boolean containsAll(SDObjectArray elements)

Returns true if this array contains all elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if all specified elements is present in this array; false otherwise.

---

containsAny

public boolean containsAny(SDObjectArray elements)

Returns true if this array contains any one of the elements of the specified array.

Parameters:

elements - elements whose presence in this array is to be tested.

Returns:

true if any specified elements are present in this array; false otherwise.

---

sort

public void sort()

Sorts the array in ascending order.

---

sort

public void sort(Comparator c)

Sorts the array of objects according to the order induced by the specified comparator.

Parameters:

c - the comparator to determine the order of the array. A null value indicates that the natural ordering of the elements should be used.

Throws:

ClassCastException - if the array contains elements that are not mutually comparable using the specified comparator.

See Also:

Comparator

---

equals

public boolean equals(Object anObject)

Element by element comparison.

Parameters:

anObject - an object to compare this array with.

Returns:

true if the arrays are equal, false otherwise.

---

toArray

public Object[] toArray()

Returns an array of Object containing all elements of this one.

Returns:

an array of double containing all elements of this one.

---

toArray

public void toArray(Object[] array)

Copies elements to a given array. If the supplied array is shorter than the number of elements in this one, only the elements that fit the supplied array are copied. If the supplied array is longer than the number of elements in this one, the rear part of the supplied array remains unchanged.

Parameters:

array - the array to which the elements of this one are to be copied.

Throws:

ArrayStoreException - if the runtime type of the supplied array is not a supertype of the runtime type of every element in this one.

---

setSize

public void setSize(int size)

Adds or deletes elements at the end of the array.

Specified by:

setSize in class SDArray

Parameters:

size - new number or elements.

Throws:

IllegalArgumentException - if size < 0.

---

trimToSize

public void trimToSize()

Removes any excess buffer capacity above the actual number of elements.

Specified by:

trimToSize in class SDArray

---

toString

public String toString()

Returns a string representation of this array. The string representation consists of elements of the array enclosed in square brackets ("[]"). Adjacent elements are separated by the characters ", " (comma and space). Elements are converted to strings by String.valueOf(Object).

Returns:

a string representation of this array.

---

indexOf

public static int indexOf(Object element, Object[] array)

Returns the index in an array of the first occurrence of the specified element, or -1 if the array does not contain such element. This method uses linear search.

---

ensureCapacity

protected void ensureCapacity(int capacity)

Grows the internal buffer as needed to accommodate the specified number of elements.

Specified by:

ensureCapacity in class SDArray

Parameters:

capacity - number of elements to accommodate

3.12 Data types in Studio

The metadata defined in studio for the following data types result in these Java classes being generated.

Metadata Type	Generated Java Type
String	String
Double	double

Metadata Type	Generated Java Type
Date	long. This number represents the number of milliseconds since midnight January 1, 1970
Boolean	Boolean
Integer	int
Java Class Name	*Java Class Name*
<Choice> (specific)	<Choice Group ID>Choice
<Choice Group> (specific)	<*Choice Group ID*>
Choice (generic)	Choice
Choice Group (generic)	Choice Group
Entity ID	<*Entity ID*>

Section 4: Deploying and Testing an Inline Service

This section describes how to deploy and test an Inline Service.

This section contains the following topics:

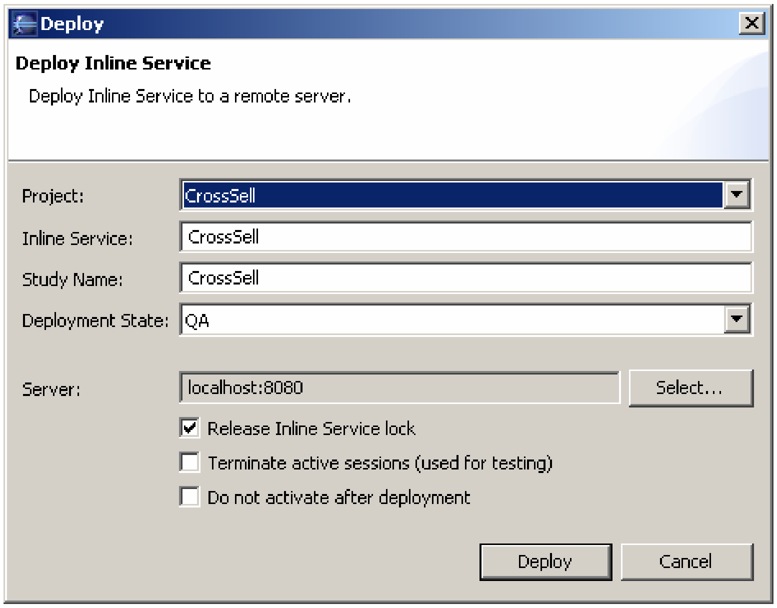
- 4.1 Deploying an Inline Service
- 4.2 Connecting to the Real-Time Decision Server
- 4.3 Redeploying Inline Services
- 4.4 Testing your Inline Service

4.1 Deploying an Inline Service

Once you have configured your Inline Service, you deploy it locally or to a test environment for testing. You can deploy an Inline Service in two different states: QA and Production.

First deploy in the QA state, and then, after testing, into Production state. When you deploy to Production state, select Release Inline Service locks. After the Inline Service is deployed to business users, they can also update and redeploy the Inline Service.

Deploy the Inline Service using the Project > Deploy menu item, or click Deploy on the task bar. The deploy dialog give several options:



Deploy

Deploy Inline Service

Deploy Inline Service to a remote server.

Project: CrossSell

Inline Service: CrossSell

Study Name: CrossSell

Deployment State: QA

Server: localhost:8080

☑ Release Inline Service lock

□ Terminate active sessions (used for testing)

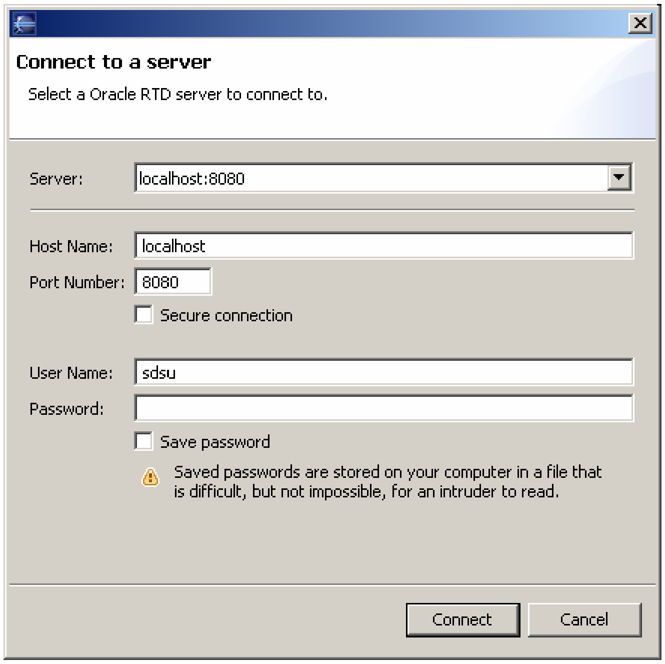
□ Do not activate after deployment

Note: You must have the proper permissions on the server cluster to deploy an Inline Service. For more about cluster permissions, see *Installation and Administration of Oracle RTD*.

Project	Choose the project that you will deploy to the Real-Time Decision Server.
Inline Service	The Inline Service contained in this project.
Study Name	Enter a study name for this Inline Service. Each Inline Service's learnings are associated with a study name. If you want to redeploy an Inline service and restart its learnings, deploy it with a new study name. One study name can be used for QA, and another for Production.
Deployment State	The default deployment states of Inline Services are Development, QA or Production. Deployment state marks an Inline Service that is in development, testing or production so that others are aware of its state. Your system administrator may have created custom deployment states.
Server	Click this option to enter the server and port to which you want to deploy. In the Server dialog box, provide a valid username and password that has deployment authorization on the server cluster to which you are deploying. Cluster authorization is granted through JConsole by your administrator.
Release Application Lock	A deployed Inline Service is automatically locked, and only the user who deployed it is able to redeploy the Inline Service. Once you have completed development and testing and are deploying the Inline Service for production, select Release Application Lock to allow Decision Center users to make changes and redeploy the Inline Service.
Terminate Active Sessions	If the Inline Service you are deploying is in production, there may be active sessions. If a new version of the Inline service is deployed while there are active sessions, the older version will be maintained to service those sessions. Select Terminate Active Sessions to terminate the active sessions if you are in testing. For a production Inline Service, keep this option deselected so that any active sessions will continue to run on the production version of the Inline Service. New sessions will be routed to the new version, and the old version will terminate when all active sessions have completed.
Do not activate after deployment	Use this option to deploy the Inline Service to the server, but not start the process. If you would like to activate the Inline Service at a later date, use JConsole. For more information about JConsole, see *Installation and Administration of Oracle RTD.*

4.2 Connecting to the Real-Time Decision Server

When deploying or downloading Inline Services or importing data sources, you connect to the Real-Time Decision Server. To connect, use the username and password you created on installation, or consult your Administrator for your username and password. To connect in a secure manner using `https`, select Secure connection.



Connect to a server

Select a Oracle RTD server to connect to.

Server: localhost:8080

Host Name: localhost

Port Number: 8080

□ Secure connection

User Name: sdsu

Password:

□ Save password

Saved passwords are stored on your computer in a file that is difficult, but not impossible, for an intruder to read.

4.3 Redeploying Inline Services

If you are going to make changes to a deployed Inline Service, it is important to follow these practices in order to preserve both your changes and the potential changes that have been made by the business user. If you are making changes to a deployed Inline Service, you can download it from a Real-Time Decision Server using the download icon on the toolbar. Use the following method:

1. Make sure that no business users are editing the deployed Inline Service.
2. Always lock an Inline Service when you download it, so that additional changes cannot be made by business users while you are enhancing it.
3. Make enhancements in Decision Studio.
4. Redeploy the Inline Service, releasing the locks.

During the period that you have the Inline Service locked, business users will be able to view, but not edit, the deployed Inline Service.

4.4 Testing your Inline Service

To test your inline Service, you can generate data using Load Generator. You can open Load Generator by running *RTD_HOME*`\scripts\loadgen.cmd`. For a sample Load Generator script, see the `etc` directory of the Cross Sell example.

4.4.1 About Load Generator

Load Generator is a tool used for debugging and benchmarking Inline Services by simulating users. Load Generator is used both for testing the Inline Service, and for performance characterization.

Load Generator has four tabs:

- Run: runs a load generator session and provides feedback on the performance through measurement and graphs.
- General: sets the general settings for Load Generator's operation, including the rate at which data is sent to the server and the location of the client configuration file.
- Variables: used to create script, message, and access variables.
- Edit Script: used to set up the script that specifies the Integration Point requests to be sent to the server.

Using Load Generator for testing

Load Generator is used to generate load on the server to test it for performance and scalability. Intelligently random messages are sent to the Inline Service, allowing the models to learn. The capability of your models can be gauged after running Load Generator for a sufficient period of time.

Using Load Generator for performance characterization

Once an Inline Service is configured, Load Generator is used to evaluate how the service performs under load in order to assess how many servers are needed for specific loads. When you want to stress the server, typically one instance of Load Generator running on one client machine is sufficient, because Load Generator can engage many threads of execution to run multiple scripts concurrently. If additional load is desired and Microsoft Task Manager shows that Load Generator is already consuming the majority of the client's processing power, then several instances of Load Generator can be started on several client computers and pointed to one server. They send messages with some intelligently random generated messages in the context of sessions. The clients measure performance statistics, as well as the server.

4.4.2 Running a Load Generator session

To start a session, first create a new script, or load an existing one. Then, select the Run option from the Run menu, or click the Run icon on the toolbar. You can alter the delay between data samples on the General tab.

Measuring the server load

The Run tab displays real-time information about the session running. It displays the following information:

New Requests	The number of requests that have been closed since the previous data sample was taken.
New Errors	The number of errors, either client or server side, that have occurred since the previous data sample was recorded.
New Default Responses	The number of errors since the last data sample, that occurred for Advisor Integration Point requests (as opposed to Informant Integration Point requests) and a default response was defined by the Inline Service for the Advisor.
Active Scripts	Number of simulated users currently connected to the server from this load generator.
Peak Response Time	The length of time it took to close the oldest request during the current data sample.
Total Requests	The total number of requests that have been closed.
Total Errors	The total number of errors.

Total Default Responses	The total number of default responses.
Total Finished Scripts	The total number of simulated users.
Average Script Duration	The length in milliseconds of an average script's execution, from start to finish.

4.4.3 Viewing performance graphs

By default, the Requests per Second graph is visible. You can hide and show graphs using View > Graphs. To clear the data in the graphs, click Clear Graphs 🗑 on the toolbar, or use View > Clear Graphs.

If you stop a script and restart it, all recorded data will be cleared. However, if you pause a session and then start it again, the data will not be cleared. The following graphs are available:

Average Response Time	A histogram depicting the 40 most recent average response times.
Errors	A line graph depicting the number of errors that occurred within the most recent 12,000 data samples.
Peak Response Time	A line graph depicting the peak response time, in milliseconds, that occurred within each of the most recent 12,000 data samples.
Requests Per Second	A line graph depicting the average number of requests per second that occurred within each of the most recent 12,000 data samples.
Requests Per Second distribution	A histogram depicting the 40 most recent readings for requests per second.

4.4.4 About the General tab

The General tab contains variables about Load Generator's configuration, timing, and which Inline Service is being specified. The General tab has the following variables:

Load Generator	
Client Configuration	Describes which endpoints Load Generator should use to contact the server.
Graphs Refresh Interval in Seconds	Sets the delay between graph and counter updates. Click Apply for settings to take effect while a script is already running.
Details	
Inline Service	The name of the Inline Service to which this script will send requests.
Random Number Generator Seed	If your script has any random elements in it, this gives you the ability to reproduce, to some extent, the random behavior. Repeatable randomness is not possible when running more than one concurrent script (see Number of Concurrent Scripts to Run later in this table).

Think Time	
Fixed Global Think Time	The number of seconds that all simulated users will wait between requests.
Ranged Global Think Time	A variable time that simulated users wait between requests. The think time changes by either a random number, or a sequentially increasing number from a set number range.
Minimum	A nonzero number of seconds to wait at a minimum.
Maximum	A nonzero number of seconds to wait at a maximum (must be greater than minimum).
Access Type Sequential	At each access, increase the think time by one until you reach the maximum, when it will reset to the minimum.
Access Type Random	At each access, choose a value between minimum and maximum, inclusive of each.
Scripts	
Number of Concurrent Scripts to Run	The number of simultaneous users to simulate.
Maximum Number of Scripts to Run	A positive number in this field causes Load Generator to stop running after that number of sessions have completed. Zero means unlimited.
Logging	
Enable Logging	When this option is selected, Load Generator statistics data is periodically written to a file.
Append to Existing File	When this option is selected, and logging is enabled, Load Generator will append new statistics data onto the end of an existing log file, if any, or else it will create a new file.
Log File	The full path to the log file, a tab-separated file whose contents is described below.
Logging Interval in Seconds	The number of seconds to wait after appending values onto the log file before writing the next set of values.

4.4.5 About variables

Variables allow a load simulation to draw its input from many different sources. Session variables are generated once per session. Subsequent accesses to a session variable use the same value. Message variables are held constant for a single request. Access variables may vary every time they are read. Variables are used in Message Actions.

Using variables

To use a variable in a message for a value to a parameter, you can select it from the drop-down list. However, if you want to use it as part of a larger string value, you can surround the variable name with braces (for example, `{customerNum}`).

4.4.6 Types

There are five types of variables.

Constant Value	A constant value.
Integer Range	Select an integer from a range. Example: Minimum: 0, Maximum: 50000, Access Type: Random Minimum: 0, Maximum: 1, Access Type: Sequential
String Array	Select a string from the specified array. Example: List: [A, B, C], Access Type: Random List: [Male, Female], Access Type: Sequential
Weighted String Array	Select from the specified array a string with some likelihood [0,1] Example: List: [[0.3, Interested], [0.3, Accepted], [0.4, Rejected]] List: [[0.999, Interested], [0.001, Accepted]]
Text File	Select a line of text from a file. Examples: c:/data.txt, Access Type: Sequential -- an absolute reference to a file on the C: drive. inbox/data.txt, Access Type: Random -- a relative reference to a file in the inbox directory, under the directory containing the script file.

4.4.7 About actions

In order to easily simulate multiple clients supplying realistic loads to the server, messages can be generated from patterns specified in metadata that are interpreted by Load Generator at run time. The patterns specify message sequences, with fixed or random inter-message delays (think times), as well as patterns for generating values for message fields. Message field values can be literal strings, with optional embedded random characters, or they can be randomly selected from a set of values associated with the field. Sessions are supported, allowing certain fields to remain constant across messages of the session, suitable for representing session keys (for example, a customer ID, call ID, or account number). The patterns allow some flexibility in the sequencing of messages. For example, in a typical session, certain messages will come before others, or a predetermined number of messages of certain kind need to happen, and so on.

Types of actions

There are two types of actions: Message and Loop.

Message has the following attributes:

Integration Point name	The name of the Integration Point that will be sent the message.
Session Keys and values	The values sent to the Integration Point request. Session keys have to be separated from other message fields because the server uses them for routing.

Loop has the following attributes:

Number of times to execute	Can be a constant value, or a range value. A range value executes either sequentially, or randomly within the range.

4.4.8 Load Generator CSV log file contents

This section describes the fields of the comma-separated value (CSV) file containing Load Generator statistics.

Date/Time	The time of day at which the current row of counters was appended to the file. Millisecond precision is available to facilitate correlations with messages in the server's log file.
Thread Pool Size	The number of threads engaged or available to run scripts. This is an implementation detail of little to interest to most people.
New Requests	The number of requests that have been closed since the previous data sample was taken.
Total Requests	The total number of requests that have been closed.
New Errors	The number of errors, either client or server side, that have occurred since the previous data sample was recorded.
Total Errors	The total number of errors.
New Default Responses	The number of errors since the last data sample, that occurred for Advisor Integration Point requests (as opposed to Informant Integration Point requests) and a default response was defined by the Inline Service for the Advisor.
Total Default Responses	The total number of default responses.
Active Scripts	Number of simulated users currently connected to the server from this Load Generator.
Total Scripts	The total number of simulated users.
Average Response Time (ms)	The average length of time it took to close the oldest request during the current data sample.
Max Response Time (ms)	The maximum length of time it took to close the oldest request during the current data sample.

Average Script Duration (ms)	The length in milliseconds of an average script's execution, from start to finish.
Snapshot Period (ms)	The number of milliseconds during which the current counter values were accumulated.

4.4.9 XLS file contents

This section describes the contents of the Microsoft Excel file, `lg_perf.xls`, included in the `etc` directory of the installation for the purpose of rendering the Load Generator counters written to `log/loadgen.csv`.

At the top, cell A1 contains a comment describing how to link `lg_perf.xls` to the tab-separated counter file as a datasource:

> To specify the path to the Load Generator performance log, place cursor the in cell A2 and select "Import External Data" > "Edit Text Import" from the "Data" menu, and navigate to the path specified in your loadgen configuration, typically {$install_directory}\log\loadgen.csv. Use default parsing settings when prompted. Data will then be automatically refreshed every 3 minutes. To change interval and other settings, select from the "Data" menu the selection "Import External Data" > "Data Range Properties".

In row 2 are the headers containing the names of each counter. All of the headers from the CSV file, described above, appear here, with values below them.

Section 5: Troubleshooting and Debugging Inline Services

Oracle RTD provides services to help troubleshoot and debug your Inline Service. While developing your Inline Service, you can use several methods to debug, such as interactive error location and validation, and a built-in test bed for your Inline Service.

Once deployed, the Real-Time Decision Server can be run in debug mode to set breakpoints in your Inline Service.

This section contains the following topics:

- 5.1 About the Problems view
- 5.2 Using the Test view
- 5.3 Using system logs
- 5.4 Using performance monitoring
- 5.5 Error messages and exceptions

5.1 About the Problems view

The Problems view identifies compilation errors and validation errors as the Inline service is built. Double-click a compilation error to display the Java perspective with the error highlighted.

Double-click a validation error to display the Inline Service perspective with the element editor for the element that has validation errors.

5.2 Using the Test view

Decision Studio includes a Test view where you can test individual Integration Points. The Test view allows you to simulate the operational systems that will call the Integration Points. The Test view has a drop-down list of all Integration Points in the Inline Service. To test the Integration Point, insert values for the session key and request data and click the run icon to run. Three subtabs provide information about the Integration Point: Results, Trace, and Log.

The Results tab shows the results of calling an Advisor Integration Point. Only Advisors return results. For testing Informants and for debugging both kinds of Integration Points, use `logInfo()`.

You can use the statement `logInfo()` at various points in your code as a debugging device. This statement is helpful to use in elements such as Advisors or Informants, Decisions, functions, and so on. Insert the statement into the logic pane of the element and use it as a device to display in the log data at different stages.

5.2.1 Using logInfo( )

The Log tab gives a view of all `logInfo()` statements.

The `logInfo` method is part of the logging API described in Section 3.2, com.sigmadynamics.support Class SDOBase. This class contains methods for logging messages at the informational, debug, warning, and error levels. These logging methods generally accept a string and another argument as parameters.

Two examples of using `logInfo` are shown below:

```
logInfo("Installation date = " + DateUtil.toString(session().
getCustomer().getInstallationDate());
```

```
logInfo("Customer age = " + session().getCustomer().getAge() );
```

5.2.2 Testing for incoming request data

When testing an Integration Point, you can check for the incoming request data using the following methods.

If the incoming parameter is mapped to a session attribute, there is a `get` method for the parameter.

```
request.get$()
```

where $ is the parameter name with the first letter capitalized.

If the attribute is not mapped, there are methods to achieve the same results using the field name of the parameter.

```
String request.getStringValue(fieldName)

SDStringArray request.getStringArrayValue(fieldName)

boolean request.isArgPresent(fieldName)
```

Outgoing response data is always stored in a SDChoiceArray:

```
    SDChoiceArray choices = null;
```

The Decision is executed by the Integration Point, and the Choice is stored:

```
    if (session().isControlGroup()) {

      choices = RandomChoice.execute();

    } else {

      choices = SelectOffer.execute();

    }
```

To find out what the Choice is, you can get them from the array and use `getSDOId` or `getSDOLabel`.

```
    if (choices.size() > 0) {

      Choice ch = choices.get(0);

      ch.getSDOId();

    }
```

The best place to do this is in the Post Selection Logic of the Decision. After the Decision executes, the post selection logic will run.

5.3 Using system logs

Oracle RTD has two logging locations.

Log	Default Location
Real-Time Decision Server log; this can be viewed from Eclipse using the Error Log view from the Window menu.	*`RTD_HOME`*`\log\server.log`
Eclipse log	*`RTD_HOME`*`\eclipse\workspace\.metadata\.log`

5.3.1 Setting logging levels

To set the logging levels for Eclipse, make changes to the file *`RTD_HOME`*`\eclipse\plugins\com.sigmadynamics.studio_2.2\etc\eclipse-log.properties`.

To adjust logging levels, set the following values to `true` or `false`. The default settings are show below.

- debug=false
- info=true
- warn=true
- error=true
- fatal=true
- trace=false

To change logging options for the Real-Time Decision Server log, use JConsole. For more information on JConsole, see *Installation and Administration of Oracle RTD*.

5.4 Using performance monitoring

Oracle RTD includes a robust performance monitoring system for observing the behavior of Inline Services. Performance Monitoring parameters are set, and a snapshot view of some of the common counters can be observed, through JConsole. A chronological view can be obtained by enabling the performance monitor. Once enabled, a comma-separated value (CSV) file is produced that can be used to observe behavior over time.

> ⚠ Warning: This file grows without limit, and should be enabled only for active troubleshooting.

5.4.1 Setting performance monitoring parameters

The performance monitoring parameters are set using the SDManagementCluster > Members > Properties > PerformanceMonitoring MBean. The following table describes the properties governing performance monitoring.

For more information on JConsole, see *Installation and Administration of Oracle RTD*.

DSPerfCounterEnabled	Enables the writing of DS performance counters. Should not be enabled indefinitely, because the file grows without limit.

DSPerfCounterAppend	If true, performance data is appended to an existing file, if any, otherwise any existing file is overwritten when the server restarts.
DSPerfCounterLogFile	The tab-separated CSV file into which DS performance counts are periodically appended. If MS Excel is available, `ds_perf.xls`, supplied in the `etc` directory of the installation, provides a convenient view. See the first row of `ds_perf.xls` for instructions on linking `ds_perf.xls` to `ds_perf.csv` as a datasource.
DSPerfCounterLogInterval	The update interval in milliseconds for DS performance counts.

5.4.2 Viewing common performance monitoring snapshot values

A snapshot of some of the performance counters is available for viewing through the SDManagementCluster > Members > Decision Service MBean. Use the F5 key to refresh the values.

Performance monitoring does *not* have to be enabled to use this view.

5.4.3 CSV file contents

This section describes the fields of the CSV file containing performance counters.

<table>
<tr><td>Date/Time</td><td colspan="2">The time of day at which the current row of counters was appended to the file. Millisecond precision is available to facilitate correlations with messages in the server’s log file.</td></tr>
<tr><td rowspan="3">Max Allowable Running Requests</td><td colspan="2">The maximum number of Inline Service requests that can run concurrently.
The value is derived from configuration settings. It should be chosen to minimize the operating system’s thread scheduling overhead, and hence provide maximum throughput for a busy system.
The value can be set manually, by setting a non-zero value in either the cluster-wide configuration property, SDManagementCluster > Properties > Misc > IntegrationPointMaxConcurrentJobs, or in the server-specific property, SDManagementCluster > Members > Properties > Misc > IntegrationPointMaxConcurrentJobs.
The preferred value is chosen by setting the property to zero, in which case the value is calculated according to the following formula.
NumCPUs * Math.ceil(1/(1-DSRequestIOFactor)) + 5
The formula uses these terms:</td></tr>
<tr><td>NumCPUs</td><td>Server-specific configuration property SDManagementCluster > Members > Properties > Misc > NumCPUs.
Use the number of physical CPUs in the machine.</td></tr>
<tr><td>Math.ceil</td><td>Means “round up to the next higher integer value.”</td></tr>
</table>

	DSRequestIOFactor	Server-specific configuration property SDManagementCluster > Members > Properties > Misc > IntegrationPointRequestIOFactor. The fraction of time Integration Point requests spend doing input/output operations, or otherwise waiting for systems external to this virtual machine. The default value is 0.5.
Peak Requests Running	The largest number of requests that have been running at the same time since the server was started.	
Max Requests Running	The largest number of requests that have been running at the same time during the current logging interval.	
Requests Running	The number of Inline Service requests that are currently running. This value will always be less than or equal to the Max Allowable Running Requests value.	
Request Queue Capacity	The configured maximum number of requests that can wait at the same time in this server to run. This is the value of the cluster-wide property SDManagement-Cluster > Properties > Misc > IntegrationPointQueueSize, or the server-specific property, SDManagement-Cluster > Members > Properties > Misc > IntegrationPointQueueSize. When a request arrives and the request queue is full, the request is rejected and a Server Too Busy error is logged in the server. The property should be set to a value slightly less than the number of concurrent HTTP requests (threads) supported by the Web server; otherwise, the request queue could never fill up, because the requests would be rejected first by the Web server.	
Peak Queue Length	The largest number of Inline Service requests that have been waiting at the same time to run in this server since the server started. This will always be less than or equal to Request Queue Capacity.	
Max Queue Length	The largest number of Inline Service requests that have been waiting at the same time to run in this server during the current logging interval. This will always be less than or equal to Request Queue Capacity.	
Requests Waiting (Queue Length)	The number of Inline Service requests that are currently waiting to run.	
Requests When Queue Full, Total	The total number of requests that have arrived while the server's request queue was full. Each of these requests was rejected with a Server Too Busy error.	
Requests Queued, Total	The total number of Inline Service requests that were required to wait to run until other requests finished running. If all requests are being queued, the system is very busy.	
Requests Seen, Total	The total number of Inline Service requests for this server.	
Requests In System	The current number of Inline Service requests being processed by this server. The number includes those waiting to run, and those already running.	

Timed Out Requests, Total	The total number of requests that have failed to finish running before their guaranteed service level timeout, as specified by cluster-wide property SDManagementCluster > Properties > Misc > IntegrationPointGuaranteedRequestTimeout. This count includes all timed-out requests since the server was started. If this number is growing but the number of queued requests is not growing, this is an indication that the Inline Service logic handling the request is too slow to satisfy the response time guarantee, even on an idle system. One or more Integration Point requests must be optimized, or the response time guarantee must be increased.
Timed Out Requests	The number of requests that failed to finish running before their guaranteed service level.
Timed Out While Running, Total	The total number of requests, observed since the server started, to have started running and not finish within their response time guarantee. The server's processing power consumed by these requests is largely wasted, because the clients will ignore their late responses. When the system is very busy, it sometimes times out requests that are still waiting to run, thus avoiding wasting resources on them.
Timed Out While Running	The number of requests, observed during the current logging interval started, to have started running and not finish within their response time guarantee. The server's processing power consumed by these requests is largely wasted, because the clients will ignore their late responses. When the system is very busy, it sometimes times out requests that are still waiting to run, thus avoiding wasting resources on them.
Timed Out Requests Still Running	The number of requests that have started running, timed out, and are still running. A non-zero value could be an indication of a programming problem in one or more Integration Points.
Request Run Time, Average (ms)	The average time, in milliseconds, during the current logging interval that requests ran. Excludes wait time, if any.
Request Run Time, Max (ms)	The largest amount of time, in milliseconds, during the current logging interval, that any single request ran. Excludes wait time, if any.
Run Times < [0.1 GRT]	The number of requests that finished running during the current logging interval and ran less than 10% of the configured guaranteed response time. There are nine similarly formatted columns, showing the run time distribution for 0.10, 0.25, 0.50, 0.75, 1.00, 1.25, 1.50, and 2.0 times the guaranteed response time.
Run Times < N and >= M	The number of requests that finished running during the current logging interval and ran less than N milliseconds and greater than or equal to M milliseconds.
Run Times >= [2.0 GRT]	The number of requests that finished running during the current logging interval and ran two or more times the configured guaranteed response time.
Request Wait Time, Average (ms)	The average time, in milliseconds, that requests waited on the request queue prior to running or timing out. Includes only those requests that finished running, or timed out before running, during the current logging interval.

Request Wait Time, Max (ms)	The largest amount of time, in milliseconds, during the current logging interval, that any single request waited on the request queue. Includes only those requests that finished running, or timed out before running, during the current logging interval.
Wait Times < [0.1 GRT]	The number of requests that finished running during the current logging interval, and were placed on the request queue before running, but waited there less than 10% of the configured guaranteed response time. There are nine similarly formatted columns, showing the wait time distribution for 0.10, 0.25, 0.50, 0.75, 1.00, 1.25, 1.50, and 2.0 times the guaranteed response time.
Wait Times < N and >= M	The number of requests that finished running during the current logging interval and waited on the request queue less than N milliseconds and greater than or equal to M milliseconds before running.
Wait Times >= [2.0 GRT]	The number of requests that finished running during the current logging interval and waited two or more times the configured guaranteed response time before timing out.
Sessions, Current	The number of Decision Server sessions still open in this server.
Sessions, Total	The total number of Decision Server sessions created by this server.
Stale Sessions Closed Asynchronously	The total number of Decision Server sessions that have been closed by kernel jobs, instead of by request threads. This is usually unimportant. In a busy system, most stale sessions are closed by request threads and the kernel jobs are engaged only as the system winds down. It could be of interest to someone observing a lot of kernel-job activity (see Kernel Jobs Running, Current).
Stale Sessions Closed by Requests	The total number of Decision Server sessions that have timed out and been closed by request threads. Most sessions will be closed this way, especially on a busy server. After processing an Inline Service request, the calling thread will be asked to close at most one stale session before returning to the caller.
Requests Forwarded, Total	The total number of Inline Service requests that this server has forwarded to a different server because a session key in the request is currently being hosted by a different server. A non-zero number is an indication that the application server or external load balancer is not perfectly routing requests to servers in a way that assures session affinity. This is OK, but performance can be improved by tuning the application server's session affinity parameters, or acquiring an external load balancing system.
Remote Session Keys, Current	The current number of session keys that this server knows reference sessions hosted by other servers. If a request arrives with one of these keys, it will be forwarded to the other server.
Remote Session Keys, Total	The total number of times that session keys were registered in this server for sessions hosted by other servers. This is an aggregation of "Remote Sessions Keys, Current".

Kernel Jobs Running, Current	The number of maintenance activities currently running in the server. Maintenance activities include model maintenance, session timing, and timed-out request processing.
Kernel Jobs Running, Peak	The largest number of maintenance activities that have run at the same time in this server. This value will always be less than or equal to the cluster-wide property SDManagement-Cluster > Properties > Misc > WorkerThreadPoolSize, or the server-specific property, SDManagement-Cluster > Members > Properties > Misc > WorkerThreadPoolSize.
Snapshot Period (ms)	The period of time, in milliseconds, over which the server collected data before logging this row of counters.

5.4.4 XLS file contents

This section describes the contents of the Microsoft Excel file, `ds_perf.xls`, included in the `etc` directory of the installation.

At the top, cell B1 contains a comment describing how to link `ds_perf.xls` to the tab-separated counter file as a datasource:

> To specify path to the ds_perf.csv file, place cursor in cell B2 and select "Import External Data" > "Edit Text Import" from the "Data" menu, and navigate to your {$install_directory}\log\ folder and select the ds_perf.csv file. Use default parsing settings when prompted. Data will then be automatically refreshed every 3 minutes. To change interval and other settings, select from the "Data" menu the selection "Import External Data" > "Data Range Properties"

In row 2 are the headers containing the names of each counter. All of the headers from the CSV file appear here, with values below them.

The following columns appear after the values from the CSV file, with formulas showing values calculated from the CSV values:

Gross Throuhput (req/sec)	The average rate of requests finishing during the current logging interval, in requests per second. The formula is: `RequestsFinished / SnapshotPeriod * 1000`
Net Throughput (req/sec)	The average rate of requests finishing during the current logging interval, excluding requests that timed out. The formula is: `(RequestsFinished - Timeouts) / SnapshotPeriod * 1000`
Utilization (%)	The percentage of the server's capacity utilized during the current logging interval. The formula is: `(RunTimeAverage * RequestsFinished) / (MaxAllowableRunningRequests * SnapshotPeriod) * 100` This value can be briefly larger than 100 when requests are finishing that started running in previous logging intervals.

5.5 Error messages and exceptions

The following exceptions may appear in your log. Suggested actions are shown with each. If these actions do not resolve your problem, contact Oracle Support Services.

Exception	Action
Error reading file *file_name*	Check that the file exists and has read permissions.
Error writing file *file_name*	Check that the file exists and has write permissions.
Cannot load Inline Service with id *id*. No Decision Service is present.	Check that the server where the Decision Service is deployed is running. Use JConsole to find where the Decision Service is deployed. For information about JConsole, see *Installation and Administration of Oracle RTD*.
Failure setting up Smart Client's default responses from file, *file_name*	Check the Smart Client properties file for the location of the file and check to make sure it exists.
Error connecting to server	Check your server connectivity and that your network is working properly.
There were compiler errors\: *errors*	Compiler errors may occur on the server if you have configured an Inline Service in a newer version of Decision Studio than the version of the Server. Upgrade your Server version to correct the problem.
Internal Error.	Contact Oracle Support Services.
No default choice is defined for Inline Service *Inline_Service_name*, or its Integration Point *Integration_Point_name*.	Default choices are choices used by the calling application when the server is unavailable. Default choices are defined at the Integration Point level.
Could not create a backup of {0} in {1}	If the backup location is on your network, check the network connectivity. If the backup location is local, check that you have enough disk space for the backup.
Could not find JMX Server *server_name*	If the JMX Server is unavailable, check the database connectivity and make sure the server is running.
Unknown Decision Service message type received by HTTP endpoint\: *endpoint_name*	Refer to *Integration with Oracle RTD* for examples and specifications for HTTP queries.
Malformed SOAP query	Refer to *Integration with Oracle RTD* for examples and specifications for SOAP queries.
Session merging not implemented, but found two keys in same request referencing different sessions	Session merging is not yet implemented. In your Inline Service, use only one session key.
Input column at location *table_or_stored_procedure* and name *column_name* has a null value which is not supported in where clauses	The Input column of a where clause cannot be null. Go to the data source indicated and provide a value for the column.

Exception	Action
The current server *server* has a newer version of the metadata so *Inline_Service* cannot be loaded	This error can occur if you are using a newer version of Decision Studio with an older version of Real-Time Decision Server. Upgrade your server to correct the error.
The current server *server_name* supports metadata versions up to *version_number,* but the metadata of *Inline_Server* is at version *version_number*	This error can occur if you are using a newer version of Decision Studio with an older version of Real-Time Decision Server. Upgrade your server to correct the error.
Generation of Inline Service "*Inline_Service_name*" failed	This error can occur if you are using a newer version of Decision Studio with an older version of Real-Time Decision Server. Upgrade your server to correct the error.
Unable to read a study definition created by a newer version of the software.	This error can occur if you are using a newer version of Decision Studio with an older version of Real-Time Decision Server. Upgrade your server to correct the error.
Unable to read a prediction model created by a newer version of the software.	This error can occur if you are using a newer version of Decision Studio with an older version of Real-Time Decision Server. Upgrade your server to correct the error.
Encountered a database record created by a newer version of the software.	This error can occur if you are using a newer version of Decision Studio with an older version of Real-Time Decision Server. Upgrade your server to correct the error.
Unable to read a learning model created by a newer version of the software.	This error can occur if you are using an newer version of Decision Studio with an older version of Real-Time Decision Server. Upgrade your server to correct the error.
No result set found while getting result set of procedure "*stored_procedure_name*".	This error can occur if you have defined a result set for your data source, but there is none on the stored procedure. Check the stored procedure definition in the database.
Generic Exception caught while setting blob for procedure "*stored_procedure_name*".	This error can occur for several reasons. Check your database connectivity. Check that the database server is running.
Exception during output of batched statements\: database *insert/update/select/delete* operation for *table_or_stored_procedure* took *duration*. Batch size is *number_of_results*.	This error can occur for several reasons. Check your database connectivity. Check that the database server is running.
The stored procedure "*stored_procedure_name*" was not found in database.	This error can occur for several reasons. Check your database connectivity. Check that the database server is running. Finally, check that the stored procedure named in the data source is named correctly and resident on the database.
Failed to find column "*column_name*" in table "*table_name*".	This error can occur for several reasons. Check your database connectivity. Check that the database server is running. Finally, check that the table named in the data source is named correctly and resident on the database.

Exception	Action
Error setting up Smart Client properties.	This error can occur if you have not properly configured your Smart Client properties file. Refer to *Integration with Oracle RTD* for information on using the Smart Client.
Failed to load Inline Service\: *Inline_Service_name*	This error can occur on start-up of the Real-Time Decision Server. Redeploy the Inline Services that did not start up. If the error reoccurs, contact Oracle Support Services.

The following errors may occur during development.

Error	Explanation
Internal error in code generator. See error log for details.	Check the Problems pane in Decision Studio for errors. If none are apparent, contact Oracle Support Services.
Cannot get *Inline_Service* from the database. Will mark it invalid	Check to make sure that the database server is running, that you have the proper drivers, and that you have connectivity to the database server.
Error loading Inline Service *Inline_Service_Name*. Will mark it as invalid in the database.	If you get an error from the server on loading your Inline Service, check to see if the logic in your Application Initialization logic and Session Initialization logic is correct.
Response for an asynchronous request to Advisor "*Advisor_name*" will not be sent	If you want a response from an Advisor, you must use the `invoke()` method, not `invokeAsync()`.
Invoke failed	Make sure that you have connectivity to the Real-Time Decision Server. Check that your properties file is properly configured. For more information on using Invoke, refer to *Integration with Oracle RTD*.
Internal error	Contact Oracle Support Services.
Internal error while generating code for *Inline_Service*.	Contact Oracle Support Services.
Error in Application Session Cleanup.	This error can be caused by incorrect logic in the Application and Session elements. Check to see if the logic in your Application Cleanup logic and Session Cleanup logic is correct.
Failed to load study "*study_name*"	This error can be caused by database connection issues. Check to make sure that the database server is running, that you have the proper drivers, and that you have connectivity to the database server.
Failed to save study "*study_name*".	This error can be caused by database connection issues. Check to make sure that the database server is running, that you have the proper drivers, and that you have connectivity to the database server.

Error	Explanation
Failed to load prediction model "*model_name*"	This error can be caused by database connection issues. Check to make sure that the database server is running, that you have the proper drivers, and that you have connectivity to the database server.
Error delivering response message: *error_details*	Your `invoke()` on the Integration Point may have timed out. Check the timeout setting in the properties file. For more information on using `invoke()`, refer to *Integration with Oracle RTD*.
Product sdstudio could not be found.	This error message is informational and can be ignored.