Working with Business Rules in My webMethods

Version 9.6

April 2014

This document applies to webMethods Business Rules Version 9.6 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: MWM-WBR-96-20140415

Table of Contents

About this Guide

Working with Business Rules in My webMethods Help is for users of My webMethods who want to modify rule projects that were exported from Software AG Designer to the My webMethods Server repository.

Business rules are created with the Rules Development feature in Software AG Designer. For more information, see *webMethods Rules Development Help*.

Working with Business Rules in My webMethods Help contains supporting documentation on the following main topics:

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service* .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).

Convention	Description
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Documentation Installation

You can download the product documentation using the Software AG Installer. The documentation is downloaded to a central directory named _documentation in the main installation directory (SoftwareAG by default).

Online Information

You can find additional information about Software AG products at the locations listed below.

If you want to...	Go to...
Access the latest version of product documentation.	**Software AG Documentation website** http://documentation.softwareag.com

If you want to...	Go to...
Find information about product releases and tools that you can use to resolve problems. See the Knowledge Center to: ■ Read technical articles and papers. ■ Download fixes and service packs (9.0 SP1 and earlier). ■ Learn about critical alerts. See the Products area to: ■ Download products. ■ Download certified samples. ■ Get information about product availability. ■ Access older versions of product documentation. ■ Submit feature/enhancement requests.	**Empower Product Support website** https://empower.softwareag.com
■ Access additional articles, demos, and tutorials. ■ Obtain technical information, useful resources, and online discussion forums, moderated by Software AG professionals, to help you do more with Software AG technology. ■ Use the online discussion forums to exchange best practices and chat with other experts. ■ Expand your knowledge about product documentation, code samples, articles, online seminars, and tutorials. ■ Link to external websites that discuss open standards and many web technology topics. ■ See how other customers are streamlining their operations with technology from Software AG.	**Software AG Developer Community for webMethods** http://communities.softwareag.com/

1 Getting Started

Before you can get started, you must have a My webMethods user account, with full permissions (read and write) for the application page you want to modify.

My webMethods provides an extremely flexible framework for granting or restricting user access to virtually every aspect of the My webMethods interface. Administrators assign permissions known as access privileges and functional privileges. Access privileges define the application pages you can display. Functional privileges define the actions you can perform in My webMethods. If this guide lists pages or menu items that you cannot access, it is likely because you do not have the access privileges to view the page. If this guide lists user interface controls (e.g. buttons) that are greyed out, it is likely because you do not have the functional privileges to perform the actions associated with the user interface controls. If you have any questions about your access privileges, consult with your My webMethods administrator.

For information about permissions management and customizing the tools, see *Administering My webMethods Server*.

Installation and Configuration Information for My webMethods System Administrators

You must install and configure several Software AG products, before you can modify rule projects that were exported from Software AG Designer to My webMethods. For complete information about installation, see *Software AG Installation Guide*.

To exchange rule projects with the Rules Development feature of Software AG Designer:

- The Business Rules User Interface must be installed on the My webMethods Server. This creates the folder in which the rule projects are stored (My webMethods Applications\webMethods Application Data\Rule Projects).
- There must exist at least one My webMethods Server user (other than "sysadmin") with full write access to this folder. The permissions for the folder can be set by the "sysadmin" user using the tools in the folder view.

To set up business rules user accounts in My webMethods:

- The My webMethods Server administrator must create a role for business rules users and assign **ALL** access rights to this role for the Rule Projects folder.
- The My webMethods Server administrator must add the My webMethods Server user who will be accessing the rule projects to this role. For more information about creating and managing user accounts and roles, see *Administering My webMethods Server*.
- The business rules user will not see the newly exported rule project until an administrator gives him permission to do so (see *Administering My webMethods Server, Managing Permissions in My webMethods*). This has to be done once for each rule project.

2 Rules Development Terminology

The following terminology applies to Working with Business Rules in My webMethods Help:

Term	Explanation
Business Rule	A business rule is a rule that defines or constrains an aspect of your business. It is intended to create a business structure or to influence the behavior of the business.
Condition	A condition is the left hand side part of a rule: `IF Condition THEN Result`.
Condition Value	A condition value determines a condition. It can consist of: ■ An operator and a literal value. ■ An operator and a parameter element. ■ An operator and an action that delivers an output value. ■ An operator and a constant.
Data Model	Business rules must be able to interact with application data from other systems. This external application data is mapped to a data model, which is then stored in your workspace as part of the rule project.
Data Model Element	A data model element is an entity of a data model. For example, a `customer` data model can contain the data model elements `name` and `age`.
Decision Entity	A decision entity is a way to display one or more rules. Decision tables, event rules, etc. are different decision entities, even though they can contain the very same rule. Some decision entities are more suited for displaying certain kinds of rules than others.
Decision Table	A decision table is a decision entity. In the decision table, the conditions and corresponding results are

Term	Explanation
	sorted into rows and columns. A column can either represent a condition (the IF part) or a result (the THEN part) of a rule. Each row in a decision table represents one individual rule.
Event Model	Event rules can operate on the basis of pre-defined event types. This event type is mapped to an event model, which is then stored in your workspace as part of the rule project.
Event Rule	An event rule is a decision entity that specifies the reaction to an event. There are two types of events: ■ Internal Events. ■ External Events. Internal events are triggered by other event rules and decision tables during rule execution. External events are pre-defined event types that were created with the webMethods Event Type Editor, see *webMethods Event-Driven Architecture Help*.
New Data Action	A new data action is an action that was mapped from a data model. It creates a new instance of this data model in the rules engine. In this way, a new output parameter that was mapped from this data model is introduced to the rules engine. It can then trigger other decision entities within one rule set that use this output parameter as an input.
Parameter	A parameter is an instance of a data model or an event model.
Parameter Element	A parameter element is an entity of a parameter.
Process Action	A process action is an action that was mapped from an existing process and can be used in a decision entity to: ■ Start a new process instance. ■ Join a running process instance. ■ Suspend one or more running process instance(s). ■ Cancel one or more running process instance(s).

Term	Explanation
	■ Fail one or more running process instance(s). ■ Resume one or more suspended process instance(s). ■ Invoke a user task.
Result	A result is the right hand side part of a rule: `IF Condition THEN Result`. There are two types of results: ■ Assignment Result. This result type is applied, whenever you want to assign a value to a result. ■ Action Result. This result type is applied, whenever you want to execute an action from a decision entity.
Result Value	A result value determines a result. There are two types: ■ Assignment result values. ■ Action result values. An assignment result value can consist of: ■ An operator and a literal value. ■ An operator and a parameter element. ■ An operator and an action that delivers an output value. ■ An operator and a constant. An action result value determines the action status: ■ Active. ■ Inactive.
Rule	A rule is a single element that specifies a decision in a `IF Condition THEN Result` syntax.
Rule Set	A rule set is a grouping of logically related decision entities. Every rule set belongs to a rule project.
Rule Project	A rule project is used as a container for different rule sets and other elements, such as data models, event models, decision entities, actions, etc. In a rule project,

Term	Explanation
	these different elements can be defined and used by all parts of the rule project.
Service Action	A service action is an action that was mapped from an existing Integration Server service (IS service). Then you can execute this service from a decision entity, or use an output value from the service in a decision entity.

3 Understanding the User Interface

To access the webMethods Business Rules functionality in My webMethods, log in to an instance of My webMethods where the webMethods Business Rules functionality has been installed.

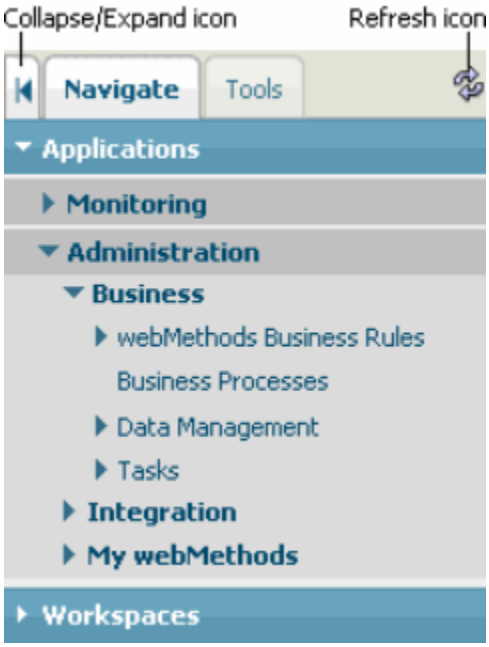
The overall layout of My webMethods is described in detail in *Working with My webMethods*. This section contains the following main topics:

- "The Navigation Pane" on page 18.
- "The Workspace Area" on page 20.

The Navigation Pane

The overall layout of the navigation pane is described in detail in *Working with My webMethods*.

The rule projects that were exported from the Rules Development feature in Software AG Designer can be accessed on the **Navigate** tab under **Applications > Administration** > **Business** > **webMethods Business Rules**.

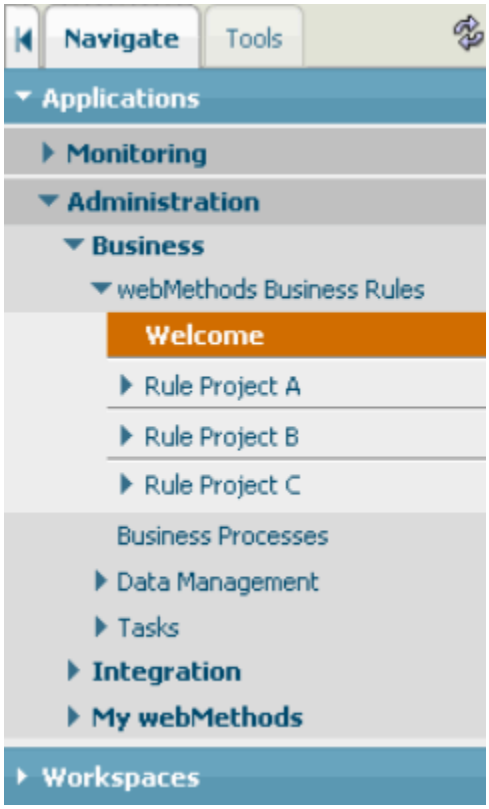


Business Rules Structure

Click **webMethods Business Rules** to open the entries for the business rules application pages.

Note: You can only see the application pages you have permission for, see "Getting Started" on page 11.

The first entry is the entry for the **Welcome** page. Below, there are the entries for the rule project application pages. The entry for the application page that is currently displayed in the workspace area is highlighted.



Rule Project Subentries

Click a rule project entry to open the subentries for this rule project. These subentries are categorized into **Decision Tables**, **Event Rules** and **Rule Sets**. The **Rule Sets** entry is subcategorized into the entries for the individual rule sets used in the rule project, and the **Master Rule Set**.

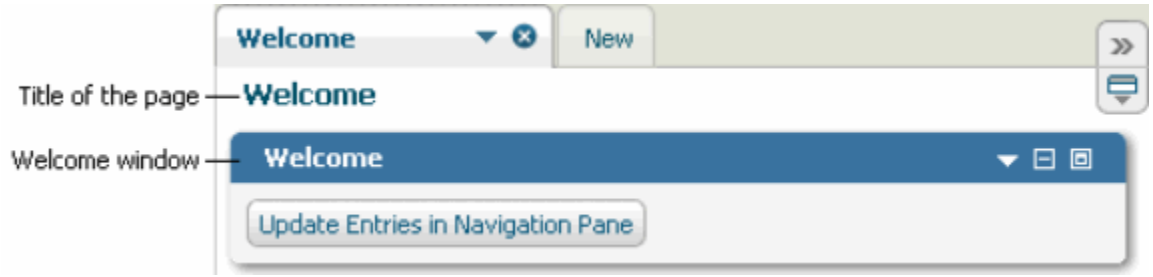
The Workspace Area

The workspace area is located on the right side of the user interface. This area contains the content of the application pages you selected from the navigation pane. There are two different types of business rules application pages:

- Welcome page, see "The Format of the Welcome Page" on page 20.
- Business rules page, see "The Format of the Business Rules Page" on page 21.

The Format of the Welcome Page

The following graphic shows the format of the Welcome page.

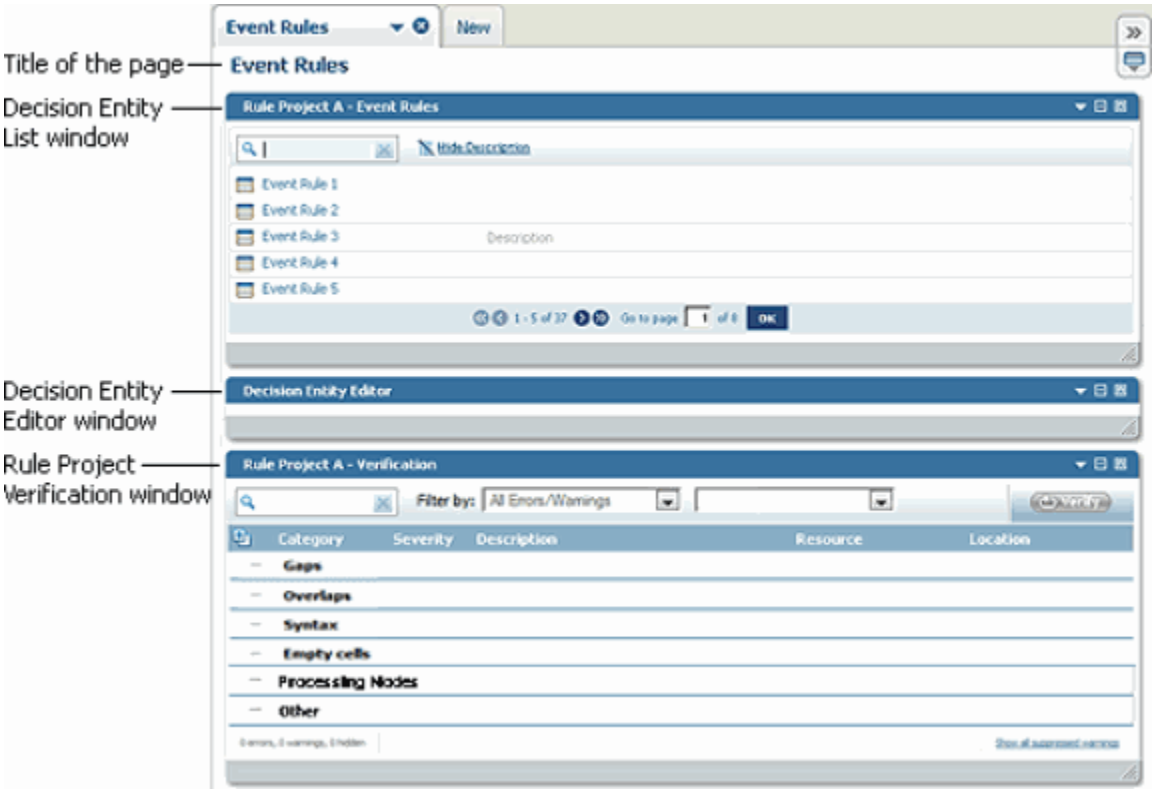


The Welcome window provides a button to update the entries for the business rules application pages on the **Navigate** tab.

Note: You can only see the application pages you have permission for, see "Getting Started" on page 11.

The Format of the Business Rules Page

The following graphic shows the format of a business rules page.



A business rules page contains the following three different windows:

Name of the Window	Description
Decision Entity List window	See "The Decision Entity List Window" on page 21.
Decision Entity Editor window	See "The Decision Entity Editor Window" on page 25.
Rule Project Verification window	See "The Rule Project Verification Window" on page 26.

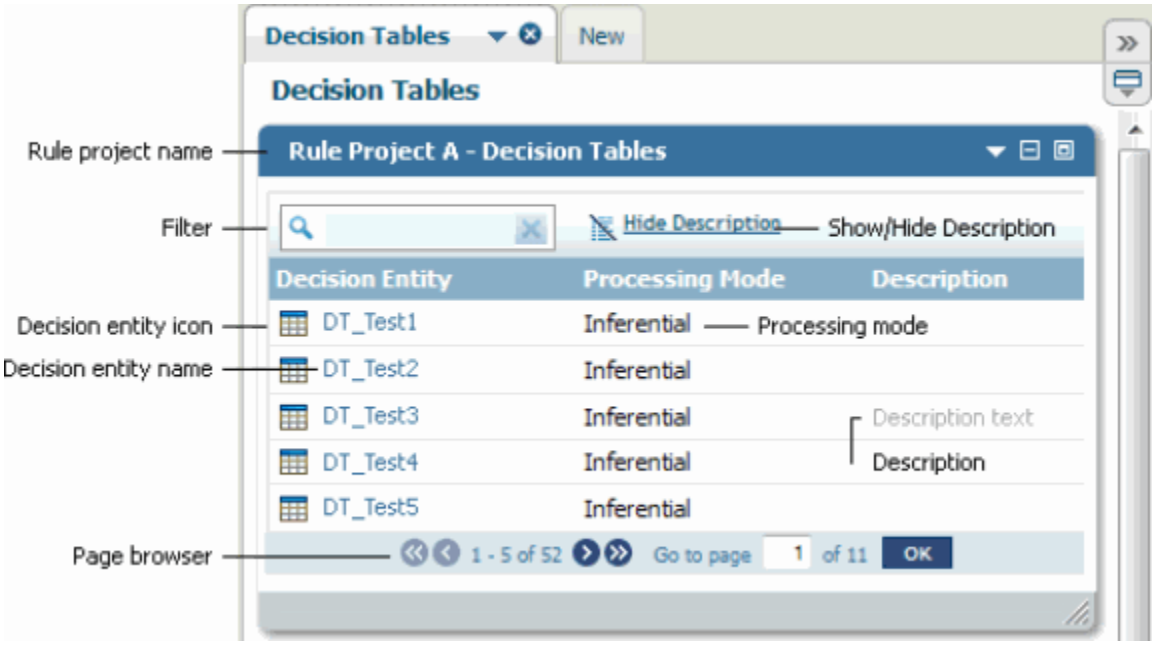
The Decision Entity List Window

The format of the Decision Entity List window depends on the business rules page that you selected from the navigation pane.

Selected Application Page	Decision Entities in the Decision Entity List Window
Decision tables	List of all decision tables used in a rule project.
Event rules	List of all event rules used in a rule project.
Rule set	List of all decision entities used in the rule set.
Master rule set	List of all decision entities used in a rule project (except external event rules).

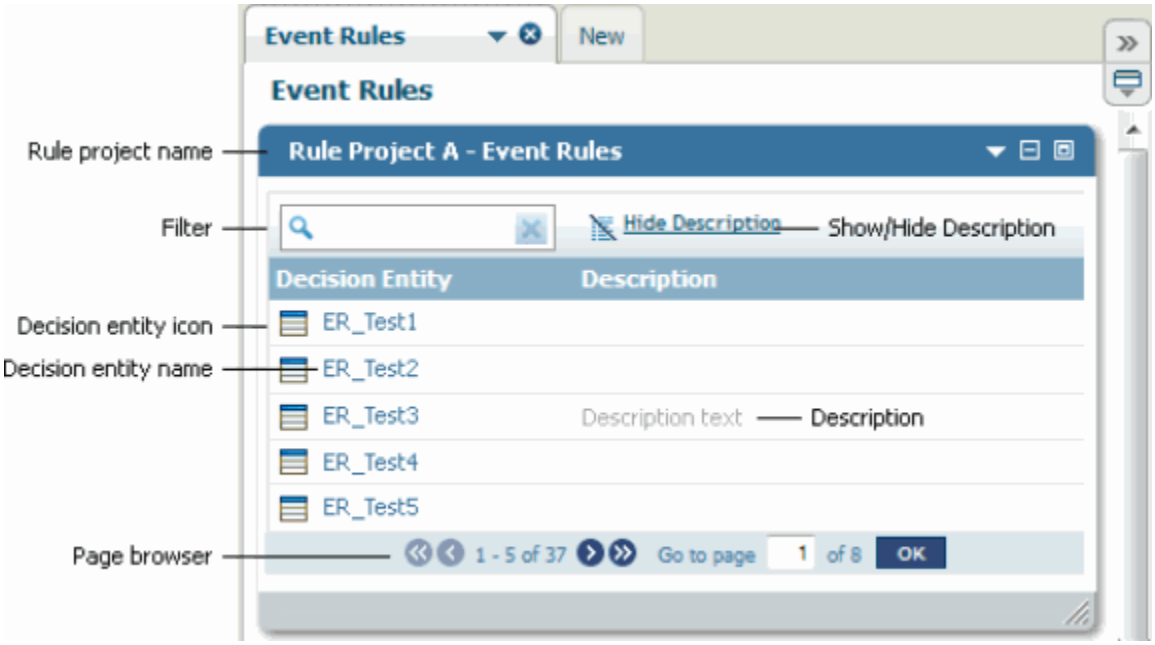
Interface for Decision Tables

The following graphic shows the format of the Decision Entity List window, if you select **Decision Tables** from the navigation pane.



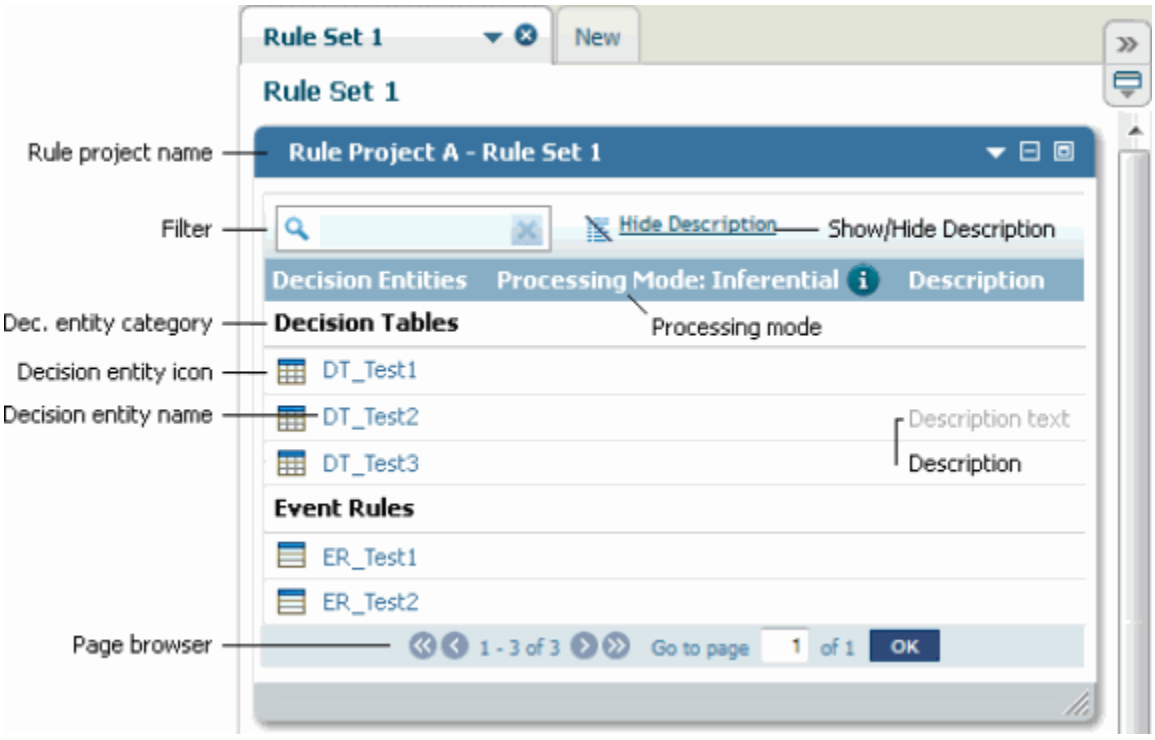
Interface for Event Rules

The following graphic shows the format of theDecision Entity List window, if you select **Event Rules** from the navigation pane.



Interface for Rule Sets

The following graphic shows the format of theDecision Entity List window, if you select **Rule Sets >*[YourRuleSetName]*** from the navigation pane.



Note: The decision entities of a rule set are sorted by the categories **Decision Tables** and **Event Rules**. The order of the decision entities within a category corresponds to the order

that was determined by the rule developer when creating the rule set in Software AG Designer.

Overview of Functions

The following table explains the different functions of the Decision Entity Editor window:

Function	Explanation
Rule project name	The name of the rule project that the selected decision entites or rule set belong to followed by the type of decision entity or the rule set name.
Filter	The input field of the text filter. To automatically filter the list of decision entities, type your filter text in the input field. To delete the filter text, click  .
Show/Hide Description	The button to suppress or restore the descriptions entered by the rule developer. To hide the descriptions, click **Hide Description**. To restore them, click **Show Description**. To modify a description, see "Modifying the Description of a Decision Entity" on page 51.
Decision entity category (only applicable for rule sets)	The category that the decision entities are sorted by in a rule set: **Decision Tables** or **Event Rules**.
Decision entity icon	The icon that specifies the type of decision entity:  (decision table) or  (event rule).
Decision entity name	The name of the decision entity. Clicking a decision entity name opens the decision entity in the Decision Entity Editor window where it can be modified as described in "Modifying a Decision Table" on page 33 and "Modifying an Event Result" on page 44.
Processing mode (only applicable for decision tables and rule sets)	The processing mode specified by the rule developer: **Inferential** (order of decision entities does not correspond to order of execution), **Sequential All** (order of decision entities corresponds to order of execution; rules are executed from top to bottom) or **Sequential First** (order of decision entities corresponds to order

Function	Explanation
	of execution; rules are executed from top to bottom; execution stops when first rule fires). For rule sets, the processing mode is shown in the table header, above the list of decision entities. For decision tables, the processing mode of each decision table is shown in the same row along with the name and description. For more information about processing modes, see*webMethods Rules Development Help*.
Description	The description of the decision entity entered by the rule developer.
Page browser	The buttons and input field of the page browser. Use the backward and forward arrows to browse through the list of decision entities. To jump to a page, type the page number in the **Go to page** input field and click **OK**.

The Decision Entity Editor Window

The following graphic shows the format of the Decision Entity Editor window.

Overview

The Decision Entity Editor window shows the decision entity that you selected from the Decision Entity List window. You can modify this decision entity as described in "Modifying a Decision Table" on page 33 and "Modifying an Event Result" on page 44.

Decision Entity Editor Toolbar

The following buttons appear in the toolbar:

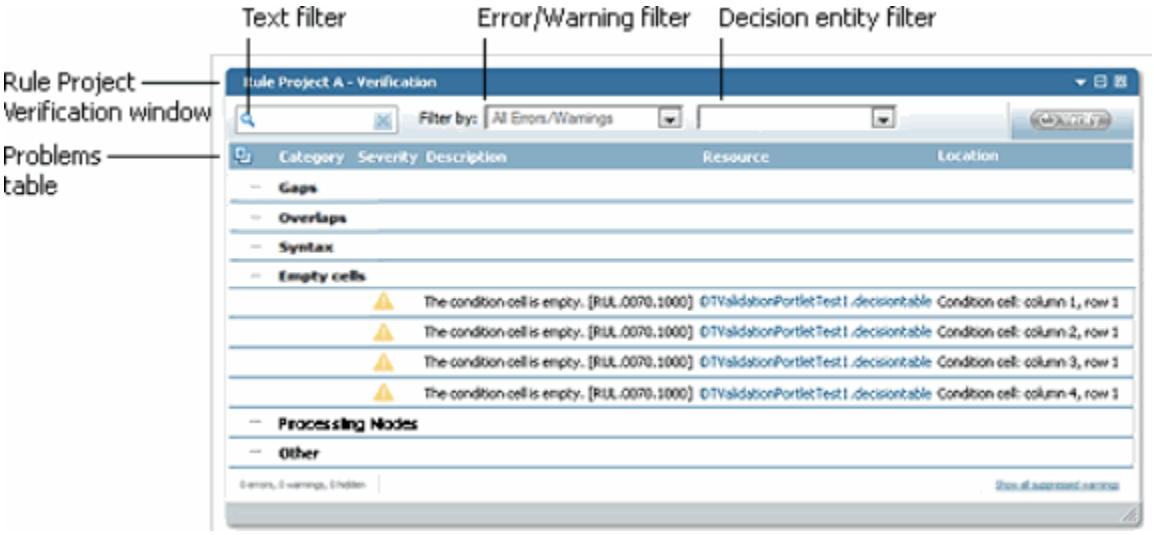
Button	Description
Filter (decision tables only)	Filters the displayed rules of a decision table, see "Filtering Rules" on page 38.
 **(decision tables only)**	Inserts a new rule after the last rule in a decision table, see "Adding a Rule" on page 37.
 **(decision tables only)**	Deletes the selected row(s) of a decision table, see "Deleting a Rule" on page 37.
	Locks the decision entity, see "Locking a Decision Entity" on page 50.
	Unlocks a locked decision entity, see "Locking a Decision Entity" on page 50.
	Hot deploys the rule project the displayed decision entity is part of, see "Hot Deploying a Rule Project" on page 61.
	Saves the changes to the decision entity, see "Saving Changes to a Decision Entity" on page 51.
	Discards the changes to the decision entity.

Decision Entity Description

Clicking the  button opens the description field that shows the description entered by the rule developer. To modify a description, see "Modifying the Description of a Decision Entity" on page 51.

The Rule Project Verification Window

Errors and warnings that are detected when verifying rules are logged in the Rule Project Verification window. The following graphic shows the format of this window.



The upper part of the Rule Project Verification window contains the filters and the **Verify** button. The following filters exist:

For this filter ...	You can do this ...
Text filter	Type a filter text in the input field to filter the entries in the problems table. To delete the filter text, click .
Error/Warning filter	Select **All Errors/Warnings** to see all errors and warnings in the problems table, or select **All Errors** to see only errors in the problems table, or select **All Warnings** to see only warnings in the problems table.
Decision entity filter	After you verified a rule set, select a decision entity to see only the errors and warnings that are associated with this decision entity in the problems table.

The middle part of the Rule Project Verification window contains the problems table. The table lists all errors and warnings sorted by verification categories. For more information about verification categories, see "About Verification Categories" on page 55.

If you click a link in the **Resource** column of the problems table, the respective decision entity opens in the Decision Entity Editor window, and it is highlighted in the Decision Entity List window.

On the left side of the lower part is the number of errors, warnings and hidden warnings. Hidden warnings are warnings that were suppressed by the filters. On the right side of the lower part is a link to show or hide warnings that were suppressed when creating the decision entities in Software AG Designer. For more information

about warning suppression, see "Showing or Hiding Suppressed Warnings" on page 55.

4 Modifying Rule Projects Overview

Modifying rule projects with My webMethods involves the following stages:

Stage 1	Log in to My webMethods. Ask your My webMethods administrator to assign the needed access privileges. For more information, see "Getting Started" on page 11.
Stage 2	Modify the rule project. For more information, see "Working with Decision Tables" on page 31, "Working with Event Rules" on page 41, and "Global Functions Overview" on page 49.
Stage 3 (optional)	Hot deploy the rule project. For more information, see "Hot Deploying Rule Projects to the Integration Server" on page 59.

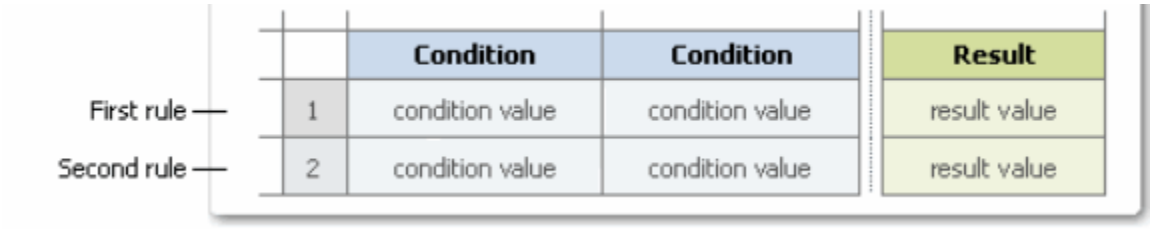
5 Working with Decision Tables

A decision table is a decision entity. It is a compact way to depict a complex set of rules in a `IF Condition THEN Result` syntax.

Decision Table Structure

In a decision table, the conditions and corresponding results are sorted into rows and columns. A column can either represent a condition (blue color) or a result (green color) of a rule. There can be more than one condition and more than one result. Each row in a decision table represents one individual rule.

Decision Table in the Decision Entity Editor



Condition

A condition is specified by a parameter element.

Condition Value

A condition value can consist of:

- An operator and a literal value.
- An operator and a parameter element (marked by a dotted line).
- An operator and an action that delivers an output value (marked by a dotted line and () behind the name).
- An operator and a constant (marked by a dotted line).
- An operator and a function (marked by a dotted line).

Result

There are two types of results:

Result	Description
Assignment Result	An assignment result is specified by a parameter element. This result type is applied, whenever you want to assign a value to a result.
Action Result	An action result is specified by an action. This result type is applied, whenever you want to execute an action from a decision table.

Assignment Result Value

An assignment result value can consist of:

- An operator and a literal value.
- An operator and a parameter element (marked by a dotted line).
- An operator and an action that delivers an output value (marked by a dotted line and () behind the name).
- An operator and a constant (marked by a dotted line).
- An operator and a function (marked by a dotted line).

Action Result Value

The action result value expresses the action status. There are two types:

- ✓ (action is enabled).
- ✗ (action is disabled).

The following rules can be modeled in a decision table:

Rule 1: IF a customer has a good credit history and the annual order value is equal to or larger than $ 5,000, THEN this customer is a VIP customer.

Rule 2: IF a customer is a VIP customer, THEN he/she will receive a bonus at the end of a year and will be notified of this by email.

The corresponding decision table uses two conditions, two assignment results and one action result:

Decision Table Example

	Order Value	Credit History	VIP-Status	Bonus	sendEmail()
1	< = 5,000	= poor	= no	= no	✗
2	< = 5,000	= good	= no	= no	✗
3	> = 5,000	= poor	= no	= no	✗
4	> = 5,000	= good	= yes	= yes	✓

Modifying a Decision Table

The Decision Entity Editor supports the following actions for decision tables:

- Adding and modifying condition or result values.
- Clearing condition or result values.
- Adding and deleting rules.
- Assigning a principal to condition or result values.
- Filtering rules.

Important: You must lock the decision table, before you modify it. Otherwise you cannot save the changes you made. For more information, see "Locking a Decision Entity" on page 50.

Adding an Operator and a Literal Value

To add an operator and a literal value:

1. Open the decision table as described in "Opening a Decision Entity" on page 50.
2. Lock the decision table as described in "Locking a Decision Entity" on page 50.
3. Click the cell you want to modify.
4. Select an operator as specified in "About Condition Operators" on page 38 or "About Result Operators" on page 39.

   Important: Adding only an operator without entering a literal value results in a semantically invalid cell.

5. Enter a literal value in the input field as required:

   Note: The literal value must match the data type as specified in "About Data Type Assignment" on page 40.

For this data type ...	You can do this ...
Boolean	Select **true** or **false** from the drop down list.
Date	a. Click the  icon. b. Select the date. c. Enter a time of day (optional). **Note:** The format and time zone of displayed date and time values can be configured in the **My Profile** settings. The webMethods Business Rules functionality only supports hours, minutes and seconds.

For this data type ...	You can do this ...
Byte Character Double Float Integer Long Short	Type the literal value.
String	Type the literal value.

6. Press ENTER, or click anywhere in the Decision Entity Editor window to remove the focus from the cell.

Modifying an Operator

To modify an operator:

1. Open the decision table as described in "Opening a Decision Entity" on page 50.
2. Lock the decision table as described in "Locking a Decision Entity" on page 50.
3. Click the operator you want to modify.
4. Select a new operator as specified in "About Condition Operators" on page 38 or "About Result Operators" on page 39.

Modifying a Literal Value

To modify a literal value:

1. Open the decision table as described in "Opening a Decision Entity" on page 50.
2. Lock the decision table as described in "Locking a Decision Entity" on page 50.
3. Click the literal value you want to modify.
4. Do one of the following:
   a. Type a new literal value as described in "Adding an Operator and a Literal Value" on page 34, Step 4.
   b. Press DEL to delete the literal value.
5. Press ENTER.

Adding a Condition or Result Value with the Cell Editor

To add a condition or result value with the cell editor:

1. Open the decision table as described in "Opening a Decision Entity" on page 50.
2. Lock the decision table as described in "Locking a Decision Entity" on page 50.
3. Click the cell you want to modify.
4. Click the  (in conditions) or  (in results) icon.
5. In the *[Value Type]* Modification dialog box, modify the value info as required:

For this field or tab ...	You can do this ...
Select field	Select an operator as specified in "About Condition Operators" on page 38 or respectively "About Result Operators" on page 39. **Note:** If you select a range operator, the dialog box splits so that you can specify a literal value or parameter element for each side of the range.
Literal value tab	Enter a literal value in the **Enter value** field as described in "Adding an Operator and a Literal Value" on page 34, Step 4. The literal value is then displayed above the tab row. **Note:** The literal value must match the data type as specified in "About Data Type Assignment" on page 40.
Parameter element tab	Expand the parameter and select a parameter element from the list. The parameter element is then displayed above the tab row. To filter the list of parameter elements, enter a filter text in the search field above the parameter element list. **Note:** The data type of the parameter element must match the data type as specified in "About Data Type Assignment" on page 40.
Functions tab	Add a function as described in "Adding a Function" on page 85.

6. Click Finish.

Modifying a Condition or Result Value with the Cell Editor

To modify a condition or result value with the cell editor:

1. Open the decision table as described in "Opening a Decision Entity" on page 50.
2. Lock the decision table as described in "Locking a Decision Entity" on page 50.
3. Click the cell you want to modify.
4. Click the  (in conditions) or  (in results) icon.
5. Modify the condition or result value as described in "Adding a Condition or Result Value with the Cell Editor" on page 36.

Clearing a Condition or Result Value

To clear a condition or result value:

1. Open the decision table as described in "Opening a Decision Entity" on page 50.
2. Lock the decision table as described in "Locking a Decision Entity" on page 50.
3. Click the operator of the cell you want to clear.
4. Select **Clear** from the context menu.

Adding a Rule

To add a rule:

1. Open the decision table as described in "Opening a Decision Entity" on page 50.
2. Lock the decision table as described in "Locking a Decision Entity" on page 50.
3. Click Add Rule in the upper left corner of the Decision Entity Editor window.

The new rule is inserted after the last rule.

Deleting a Rule

To delete a rule:

1. Open the decision table as described in "Opening a Decision Entity" on page 50.
2. Lock the decision table as described in "Locking a Decision Entity" on page 50.
3. Select the rules you want to delete by clicking the row number. To deselect a rule, click the row number again.

4. Click Delete in the upper left corner of the Decision Entity Editor window.

Note: If you delete the only rule of a decision table, an empty rule is automatically inserted.

Assigning a Principal to a Condition or Result Value

A principal is a user or group on My webMethods Server. You can assign a principal to a condition value or result value of a string type condition or result. This is only possible, if the rules developer who created the decision table with the Rules Development feature in Software AG Designer annotated the condition or result column as principal. For more information, see *webMethods Rules Development Help*.

To assign a principal to a condition or result value:

1. Open the decision table as described in "Opening a Decision Entity" on page 50.
2. Lock the decision table as described in "Locking a Decision Entity" on page 50.
3. Click the condition value cell or result value cell you want to modify.
4. Click (in conditions) or (in results).
5. In the Select Principals window, select one or more principals, and click Apply. For more information on the Select Principals window, see *Working with My webMethods Help*.

Filtering Rules

You can specify which rules of a decision table should be displayed.

Note: Any selected rules are deselected when you modify the filter.

To filter the rules to be displayed:

1. Open the decision table as described in "Opening a Decision Entity" on page 50.
2. Lock the decision table as described in "Locking a Decision Entity" on page 50.
3. Do one of the following:
   a. In the toolbar, enter a text in the filter and press ENTER.
   b. In the toolbar, click in the filter and select a user, group or role from the drop down list.
4. To restore all rules, click in the filter.

About Condition Operators

The following operators can be assigned to the different data types of decision table conditions:

Data Type(s)	Operator	Definition
Boolean	= !=	(Equals; default operator) (Does not equal)
Date	= != < <= > >= < ... <= <= ... <= < ... < <= ... <	(Equals; default operator) (Does not equal) (Less than) (Less than or equal) (Greater than) (Greater than or equal) (Less than ... less than or equal) (Less than or equal ... less than or equal) (Less than ... less than) (Less than or equal ... less than)
Byte Character Double Float Integer Long Short	= != < <= > >= < ... <= <= ... <= < ... < <= ... <	(Equals; default operator) (Does not equal) (Less than) (Less than or equal) (Greater than) (Greater than or equal) (Less than ... less than or equal) (Less than or equal ... less than or equal) (Less than ... less than) (Less than or equal ... less than)
String	= !=	(Equals; default operator) (Does not equal)

About Result Operators

The following operators can be assigned to the different data types of decision table assignment results:

Data Type(s)	Operator	Definition
Boolean	=	(Equals; default operator)
Date	=	(Equals; default operator)
Byte Character Double	=	(Equals; default operator)

Data Type(s)	Operator	Definition
Float Integer Long Short		
String	=	(Equals; default operator)

About Data Type Assignment

The following data types can be assigned to a parameter element that was specified for a condition or result:

Data type of the parameter element for the condition or result is ...	Literal value must be ...	Data type of assigned parameter element must be ...
Boolean	Boolean	Boolean
Date	Date	Date
String	String	String
Numeric (Byte, Character, Double, Float, Integer, Long, Short)	Same data type or numeric data type with a smaller value.	Any numeric data type. Numeric data types with a greater value are truncated.

6 Working with Event Rules

An event rule is a decision entity that specifies the results triggered by an event. There are two types of events:

- Internal Events.
- External Events.

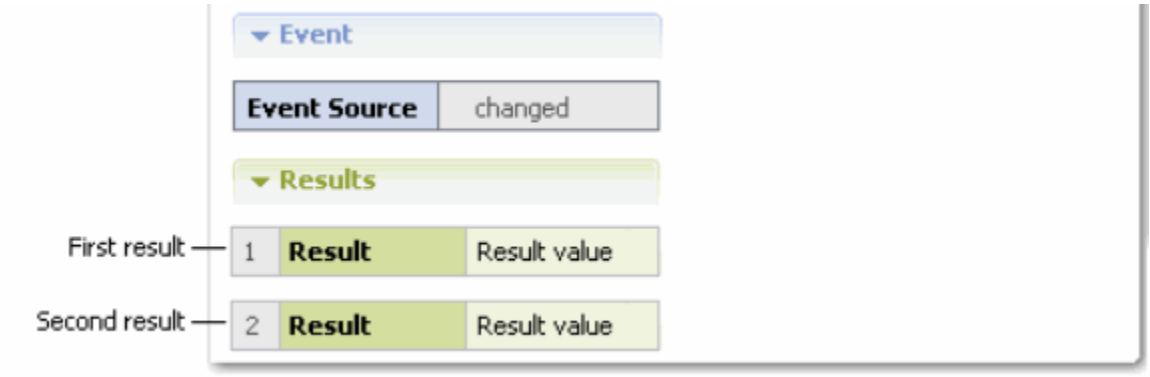
Internal events are triggered by other event rules and decision tables during rule execution. External events are pre-defined event types that were created with the webMethods Event Type Editor, see *webMethods Event-Driven Architecture Help*.

Important: To work properly, internal and external event rules must be part of a rule set.

Event Rule Structure

An event rule consists of an event (blue color) and one or more results (green color).

Event Rule in the Decision Entity Editor



Event

The event consists of an event source that is specified by a parameter element and a type. The following types are supported:

Event	Type	Description
Internal Event	`changed`	This type triggers one or more results whenever the event source changes. The change must be triggered by other event rules or decision tables. `Changed` type event rules have the following syntax: `WHENEVER an Event Source CHANGED THEN Result.`
External Event	`occurred`	This type triggers one or more results whenever the external event has

Event	Type	Description
		occurred. `Occurred` type event rules have the following syntax: `WHENEVER an Event Source OCCURRED THEN Result.`

Results

There are two types of results:

Result	Description
Assignment Result	An assignment result is specified by a parameter element. This result type is applied, whenever you want to assign a value to a result.
Action Result	An action result is specified by an action. This result type is applied, whenever you want to execute an action from an event rule.

Assignment Result Value

An assignment result value can consist of:

- An operator and a literal value.
- An operator and a parameter element (marked by a dotted line).
- An operator and an action that delivers an output value (marked by a dotted line and () behind the action name).
- An operator and a constant (marked by a dotted line).
- An operator and a function (marked by a dotted line).

Action Result Value

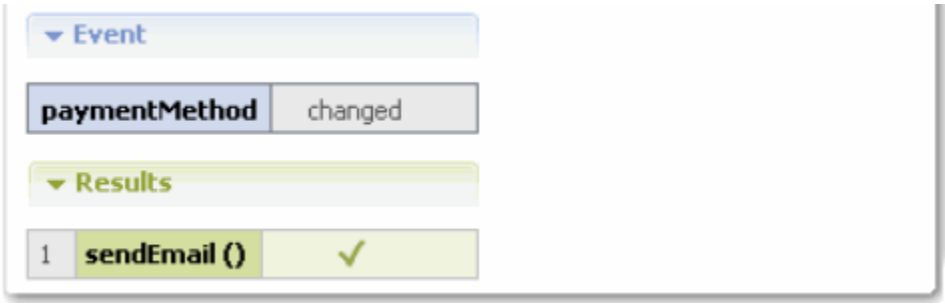
The action result value expresses the action status. There are two types:

- ✓ (action is enabled).
- ✗ (action is disabled).

The following rule can be modeled in an event rule using an action result:

Rule WHENEVER a permitted payment method changes for a customer, THEN this customer is notified of this by email.

Event Rule Example



Modifying an Event Result

The Decision Entity Editor supports the following actions for event rule results:

- Adding and modifying result values.
- Clearing result values.

Important: You must lock the event rule, before you modify it. Otherwise you cannot save the changes you made. For more information, see "Locking a Decision Entity" on page 50.

Adding an Operator and a Literal Value

To add an operator and a literal value:

1. Open the event rule as described in "Opening a Decision Entity" on page 50.
2. Lock the event rule as described in "Locking a Decision Entity" on page 50.
3. Click the cell you want to modify.
4. Select an operator as specified in "About Result Operators" on page 47.

   Important: Adding only an operator without entering a literal value results in a semantically invalid cell.

5. Enter a literal value in the input field as required:
6. Press ENTER, or click anywhere in the Decision Entity Editor window to remove the focus from the cell.

Modifying an Operator

To modify an operator:

1. Open the event rule as described in "Opening a Decision Entity" on page 50.
2. Lock the event rule as described in "Locking a Decision Entity" on page 50.
3. Click the operator you want to modify.
4. Select a new operator as specified in "About Result Operators" on page 47.

Modifying a Literal Value

To modify a literal value:

1. Open the event rule as described in "Opening a Decision Entity" on page 50.
2. Lock the event rule as described in "Locking a Decision Entity" on page 50.
3. Click the literal value you want to modify.
4. Do one of the following:
   a. Type a new literal value as described in "Adding an Operator and a Literal Value" on page 44, Step 4.
   b. Press DEL to delete the literal value.
5. Press ENTER.

Adding a Result Value with the Cell Editor

To add a result value with the cell editor:

1. Open the event rule as described in "Opening a Decision Entity" on page 50.
2. Lock the event rule as described in "Locking a Decision Entity" on page 50.
3. Click the cell you want to modify.
4. Click the  icon.
5. In the Result Value Modification dialog box, modify the value info as required:

For this field or tab ...	You can do this ...
Select field	Select an operator as specified in "About Result Operators" on page 47.

For this field or tab ...	You can do this ...
	Note: If you select a range operator, the dialog box splits so that you can specify a literal value or parameter element for each side of the range.
Literal value tab	Enter a literal value in the **Enter value** field as described in "Adding an Operator and a Literal Value" on page 44, Step 4. The literal value is then displayed above the tab row. **Note:** The literal value must match the data type as specified in "About Data Type Assignment" on page 47.
Parameter element tab	Expand the parameter and select a parameter element from the list. The parameter element is then displayed above the tab row. To filter the list of parameter elements, enter a filter text in the search field above the parameter element list. **Note:** The data type of the parameter element must match the data type as specified in "About Data Type Assignment" on page 47.
Functions tab	Add a function as described in "Adding a Function" on page 85.

6. Click Finish.

Modifying a Result Value with the Cell Editor

To modify a result value with the cell editor:

1. Open the event rule as described in "Opening a Decision Entity" on page 50.
2. Lock the event rule as described in "Locking a Decision Entity" on page 50.
3. Click the cell you want to modify.
4. Click the icon.
5. Modify the result value as described in "Adding a Result Value with the Cell Editor" on page 45, Step 4.

Clearing a Result Value

To clear a result value:

1. Open the event rule as described in "Opening a Decision Entity" on page 50.
2. Lock the event rule as described in "Locking a Decision Entity" on page 50.
3. Click the operator of the cell you want to clear.
4. Select **Clear** from the context menu.

About Result Operators

The following operators can be assigned to the different data types of an event rule assignment result:

Data Type(s)	Operator	Definition
Boolean	=	(Equals; default operator)
Date	=	(Equals; default operator)
Byte Character Double Float Integer Long Short	=	(Equals; default operator)
String	=	(Equals; default operator)

About Data Type Assignment

The following data types can be assigned to a parameter element that was specified for an event result:

Data type of the parameter element for the result is ...	Literal value must be ...	Data type of assigned parameter element must be ...
Boolean	Boolean	Boolean

Data type of the parameter element for the result is ...	Literal value must be ...	Data type of assigned parameter element must be ...
Date	Date	Date
String	String	String
Numeric (Byte, Character, Double, Float, Integer, Long, Short)	Same data type or numeric data type with a smaller value.	Any numeric data type. Numeric data types with a greater value are truncated.

7 Global Functions Overview

My webMethods supports the following functions that apply to all decision entities:

- Opening a decision entity.
- Saving changes to a decision entity.
- Modifying the description of a decision entity.
- Locking a decision entity.

Important: You must lock the decision entity, before you modify it. Otherwise you cannot save the changes you made. For more information, see "Locking a Decision Entity" on page 50.

Opening a Decision Entity

To open a decision entity:

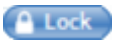
1. On the **Navigate** tab, do one of the following:
   a. Click ***[RuleProjectName]*** **> Decision Tables** (for decision tables).
   b. Click ***[RuleProjectName]*** **> Event Rules** (for event rules).
   c. Click ***[RuleProjectName]*** **> Rule Sets >*[RuleSetName]*** (for all decision entities of a rule set).
2. In the Decision Entity List window, click the decision entity name.

The decision entity opens in the Decision Entity Editor window. The order of rules corresponds to the order specified by the rule developer in Software AG Designer.

Locking a Decision Entity

You must lock a decision entity, before you modify it. Otherwise you cannot save the changes you made.

To lock a decision entity:

1. Open the decision entity as described in "Opening a Decision Entity" on page 50.
2. Click 

in the upper right corner of the Decision Entity Editor window.
3. Modify and save the decision entity.
4. To make the decision entity available to others for modification, click 

in the upper right corner of the Decision Entity Editor window.

Saving Changes to a Decision Entity

To save changes to a decision entity:

1. Open the decision entity as described in "Opening a Decision Entity" on page 50.
2. Lock the decision entity as described in "Locking a Decision Entity" on page 50.
3. Modify the decision entity.
4. Click Save in the upper right corner of the Decision Entity Editor window.

> **Note:** You cannot save a decision entity that contains errors. In this case, an error message is displayed, and the decision entity is not saved until you fix the error and click the Save button again.

Modifying the Description of a Decision Entity

To modify the description of a decision entity:

1. Open the decision entity as described in "Opening a Decision Entity" on page 50.
2. Lock the decision entity as described in "Locking a Decision Entity" on page 50.
3. In the Decision Entity Editor window, click .
4. Type a new description.
5. Click Save.

8 Rule Verification Overview

My webMethods supports two kinds of verification:

- **Automatic Verification** is performed on decision entities when they are opened or saved after modification and can reflect both errors and warnings. For more information, see "About Automatic Verification" on page 54.
- **Manual Verification** is performed on-demand on rule set or decision entity level. It is designed to detect potential logic problems in decision entities and only creates warnings. For more information, see "Verifying Rules Manually" on page 54.

For more information about verification categories, see "About Verification Categories" on page 55.

About Automatic Verification

Automatic verification is performed on decision entities when they are opened or saved after modification and can reflect both errors and warnings.

Representation of Warnings and Errors in the Decision Entity List Window

Decision tables with errors or warnings are marked by a  (error) or  icon (warning). Event rules with errors or warnings are marked by a  (error) or  icon (warning).

Representation of Warnings and Errors in the Decision Entity Editor Window

The respective cell of the decision entity in the Decision Entity Editor window is marked red (error) or yellow (warning).

Representation of Warnings in the Rule Project Verification Window

The warnings and errors appear in the Rule Project Verification window in the verification categories **Syntax**, **Empty cells**, **Processing Modes** or **Other**. For more information about verification categories, see "About Verification Categories" on page 55.

Errors are marked by a  icon, warnings are marked by a  icon.

If you click a link in the **Resource** column of the problems table, the respective decision entity opens in the Decision Entity Editor window, and it is highlighted in the Decision Entity List window.

Verifying Rules Manually

Manual Verification is performed on-demand on rule set or decision entity level. It is designed to detect potential logic problems in decision entities and only creates warnings.

Keep the following points in mind when verifying rules:

- Rules can be verified on rule set or decision entity level.
- If you verify a rule set, all of its decision entities are combined and tested as a single entity.
- Decision entities with errors cannot be verified.
- Only conditions with more than one value are considered, and each condition value is processed independently.
- Condition values containing a parameter element or an action are not considered.
- There can be multiple warnings for one condition value.
- Event rules are not considered, as they do not have condition values.
- Condition values of the data type date are not considered.

To verify rules manually:

1. Click  in the upper right corner of the Rule Project Verification window.

The warnings appear in the problems table of the Rule Project Verification window and are sorted by verification categories. For more information about verification categories, see "About Verification Categories" on page 55. If the warning can be associated with a decision entity, the respective decision entity is marked by a  icon (decision table) or icon (event rule) in the Decision Entity List window.

Showing or Hiding Suppressed Warnings

Warnings can be suppressed when creating decision entities in Software AG Designer. You can make these suppressed warnings visible or hide them again in the Rule Project Verification window.

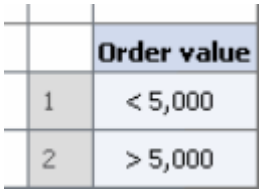
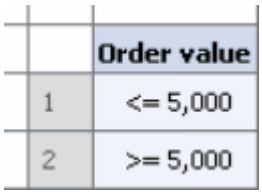
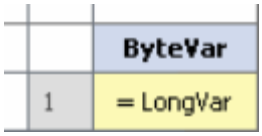
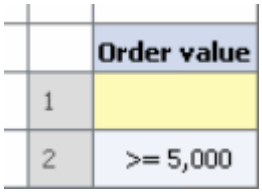
To show or hide suppressed warnings:

1. Click **Show all suppressed warnings** in the lower right corner of the Rule Project Verification window.
2. To hide the warnings again, click **Hide all suppressed warnings** in the lower right corner of the Rule Project Verification window after the page has been reloaded.

The warnings are shown or hidden in the Rule Verification window, in the Decision Entity List window and in the Decision Entitywindow.

About Verification Categories

The following verification categories exist:

Type	Explanation	Example	Action
Gap	A gap warning is reported, if a condition value or a range of condition values is not explicitly tested in a decision entity or a rule set.	 A gap will be reported, because the value = *5,000* is not tested.	If the gap is not intended, specify the missing condition value.
Overlap	An overlap warning is reported, if the same condition value or range of condition values is tested multiple times in a decision entity or a rule set.	 An overlap will be reported, because the value = *5,000* is tested multiple times.	If the overlap is not intended, modify the decision table in such a way that the respective condition value is only tested once.
Syntax	A syntax warning is for instance reported, if data will be lost due to truncation. A syntax error is for instance reported, if a parameter element that is used in the decision entity is missing in the parameter.	 A syntax warning will be reported, because a value of the data type `long` is assigned to a value of the data type `byte` and will therefore be truncated.	If the data loss is not tolerable, assign a value of the correct data type.
Empty cells	An empty cell warning is reported, if a condition value cell or a result value cell is empty.		If the empty cell is not intended, specify the missing value.

Type	Explanation	Example	Action
		An empty cell warning will be reported, because a condition value cell is empty.	
Processing Modes	A processing mode warning is reported, if the processing mode of a decision table within a rule set differs from the processing mode of this rule set, because the processing mode of the rule set overwrites that of the decision table.	Differences in processing modes can occur, if you add an inferential decision table to a sequential rule set or vice versa; or if you modify the processing mode of a rule set or of a decision table within this rule set.	If the different processing mode is not intended, the same processing mode for the decision table and rule set must be specified with the Rules Development feature in Software AG Designer, and the rule project must be reimported.
Other	All warnings and errors that do not fit into the other categories.		

9 Hot Deploying Rule Projects to the Integration Server

The Software AG rules engine executes the rules that were created with the Rules Development feature in Software AG Designer. The rules engine exists on the Integration Server as part of the WmBusinessRules package.

You can deploy your modified rules to the Integration Server, which is used as a target runtime environment. There these rules can be accessed and used by multiple business processes. For more information, see *webMethods Process Development Help*.

My webMethods supports the deployment of rule projects to a single or multiple Integration Server(s) using the hot deploy command as described in "Hot Deploying a Rule Project" on page 61.

Before you can hot deploy a rule project, you must be connected to the Integration Server(s). To configure an Integration Server connection, follow the instructions as described in "Configuring an Integration Server Connection" on page 60.

Configuring an Integration Server Connection

Before you can hot deploy a rule project, you must configure the Integration Server connection(s).

If you use a cluster or a non-clustered group of Integration Servers (Integration Servers that share the same database components), you only need to configure the connection to one of the Integration Servers of the group.

To configure an Integration Server connection:

1. On the **Navigate** tab, click **Administration > My webMethods > System Settings > webMethods Business Rules Settings**.
2. In the workspace area, click **Add Integration Server**.
3. Modify the Integration Server info as required:

For this field ...	You can do this ...
Logical Name	Type a name for the Integration Server.
IS Host	Type the Integration Server address.
IS Port	Type the Integration Server port.
IS Username	Type your Integration Server user name.
IS Password	Type your Integration Server password.

4. Click **Save**.

Hot Deploying a Rule Project

To hot deploy a rule project:

1. Open any decision entity that is part of the rule project as described in "Opening a Decision Entity" on page 50.
2. Click 

in the upper right corner of the Decision Entity Editorwindow.

   Note: You cannot hot deploy a decision entity that contains unsaved changes or errors. In this case, the **Hot Deploy** button is disabled.

3. In the Hot Deployment Confirmation dialog box, click 

to deploy the decision entity and all other components of the rule project to the Integration Server runtime.

   Note: All decision entities of the rule project are verified before the rule project is deployed. If the rule project contains any errors, an error dialog box informs you in which rule sets the errors occur. You must eliminate all errors before you can retry to hot deploy the rule project.

The Hot Deployment Results dialog box lists the results of the operation. To see a detailed list of successfully and unsuccessfully deployed rule projects, click **See Details**.

10 Working with Functions

My webMethods provides a set of predefined functions that allow you to perform simple or even complex functionality for a decision table condition, a decision table assignment result or an event rule assignment result with a minimal amount of effort. A function can require input parameters. These input parameters can be manually entered, or they can be mapped to existing parameter elements.

Example of a simple function: You can check, if the input values for a `Customer.City` parameter start with the string `New`.

Chaining Functions

You can call multiple functions by chaining them.

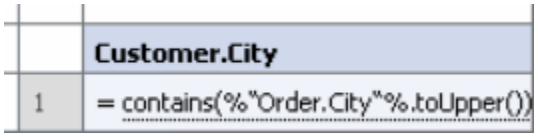
Example of a chained function: You can check, if the trimmed input values for a `Customer.City` parameter end with the string `York`.

Nesting Functions

Functions can be nested. In this case, the return value of the inner function serves as input parameter for the outer function.

Note: You cannot add or modify nested functions in My webMethods. Nested functions that were imported from the Rules Development feature of Software AG Designer can be used but not modified.

Example of a nested function: You can check, if the input values for a `Customer.City` parameter contains the upper case value of a `Order.City` parameter.



Use of Operators

With functions, you can use the operators `=` or `!=` for conditions, and `=` for results.

Note: If you use a boolean function in a condition that is not of data type boolean, the return value of the function is not compared to the condition, but it is compared against the value `True`. You cannot use a boolean function in a result that is not of data type boolean, as the return value of a function that is used in a result must match the data type that was specified for the result.

Escaping

Java escaped sequences are not supported. You can use escaped Unicode characters. For example, to insert a new line character use `\u000a` instead of `\n`.

Summary of Functions

My webMethods provides the following predefined functions:

Function	Returns	Description
concat(String str): String	String	Concatenates the specified string to the end of this string.
contains(String str): boolean	Boolean	Returns true if and only if this string contains the specified string.
endsWith(String suffix): boolean	Boolean	Tests if this string ends with the specified suffix.
equals(String anotherString): boolean	Boolean	Compares this string to the specified string.
equalsIgnoreCase(String anotherString): boolean	Boolean	Compares this string to another string, ignoring case considerations.
inList(String[] list): boolean	Boolean	Indicates whether or not this string is in the specified list.
inList(String[] list, boolean ignoreCase): boolean	Boolean	Indicates whether or not this string is in the specified list. Case sensitivity can be ignored while checking.
inRange(String lowerBound, String upperBound): boolean	Boolean	Indicates whether or not this string is within the specified range.
inRange(String lowerBound, String upperBound, String[] exclusions): boolean	Boolean	Indicates whether or not this string is within the specified

Function	Returns	Description
		range. A separate list of exclusions can be provided.
inRange(String[][] listOfRanges, String[] exclusions): boolean	Boolean	Indicates whether or not this string is within the specified list of ranges. A separate list of exclusions can be provided.
inRange(String[][] listOfRanges, boolean ignoreCase, boolean inclusiveLower, boolean inclusiveUpper, String[] exclusions): boolean	Boolean	Indicates whether or not this string is within the specified list of ranges. Upper and lower end of list can optionally be included. A separate list of exclusions can be provided. Case sensitivity can be ignored while checking.
inRange(Long lowerBound, Long upperBound): boolean	Boolean	Indicates whether or not this integer (long, integer, short) value is within the specified range.
inRange(Long lowerBound, Long upperBound, Long[] exclusions): boolean	Boolean	Indicates whether or not this integer (long, integer, short) value is within the specified range. A separate list of exclusions can be provided.
inRange(Long[][] listOfRanges, Long[] exclusions): boolean	Boolean	Indicates whether or not this integer (long, integer, short) value is within the specified list of ranges. A separate list of exclusions can be provided.
inRange(Long[][] listOfRanges, boolean inclusiveLower, boolean inclusiveUpper, Long[] exclusions): boolean	Boolean	Indicates whether or not this integer (long, integer, short) value is within the specified list of ranges. Upper and lower end of list can optionally be included. A separate list of exclusions can be provided.
inRange(Double lowerBound, Double upperBound): boolean	Boolean	Indicates whether or not this floating point (double, float)

Function	Returns	Description
		value is within the specified range.
inRange(Double lowerBound, Double upperBound, Double[] exclusions): boolean	Boolean	Indicates whether or not this floating point (double, float) value is within the specified range. A separate list of exclusions can be provided.
inRange(Double[][] listOfRanges, Double[] exclusions): boolean	Boolean	Indicates whether or not this floating point (double, float) value is within the specified list of ranges. A separate list of exclusions can be provided.
inRange(Double[][] listOfRanges, boolean inclusiveLower, boolean inclusiveUpper, Double[] exclusions): boolean	Boolean	Indicates whether or not this floating point (double, float) value is within the specified list of ranges. Upper and lower end of list can optionally be included. A separate list of exclusions can be provided.
matches(String regex): boolean	Boolean	Tells whether or not this string matches the given regular expression.
regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len): boolean	Boolean	Tests if two string regions are equal. Case sensitivity can be ignored while checking.
replaceAll(String regex, String replacement): String	String	Replaces each substring of this string that matches the given regular expression with the given replacement.
replaceFirst(String regex, String replacement): String	String	Replaces the first substring of this string that matches the given regular expression with the given replacement.
startsWith(String prefix): boolean	Boolean	Tests if this string starts with the specified prefix.

Function	Returns	Description
substring(int beginIndex): String	String	Returns a new string that is a substring of this string.
substring(int beginIndex, int endIndex): String	String	Returns a new string that is a substring of this string.
toLowerCase(): String	String	Converts all of the characters in this string to lower case.
toUpperCase(): String	String	Converts all of the characters in this string to upper case.
trim(): String	String	Returns a copy of the string, with leading and trailing whitespace omitted.

concat(String str): String

Concatenates the specified string to the end of this string.

Input Parameters

str **String** The string that is concatenated to the end of this String.

Return Value

String Returns a string that represents the concatenation of this object's characters followed by the string argument's characters.

contains(String str): boolean

Returns true if and only if this string contains the specified string.

Input Parameters

str **String** The string to search for.

Return Value

Boolean `True` if this string contains the specified string. `False` otherwise.

endsWith(String suffix): boolean

Tests if this string ends with the specified suffix.

Input Parameters

suffix	**String** The specified suffix.

Return Value

Boolean `True` if the character sequence represented by the argument is a suffix of the character sequence represented by this object. `False` otherwise.

Note: The result will be true if the argument is the empty string or is equal to this String object as determined by the equals(String) method.

equals(String anotherString): boolean

Compares this string to the specified string. The result is true if and only if the argument is not null and is a string that represents the same sequence of characters as this string.

Input Parameters

anotherString	**String** The string to compare this string against.

Return Value

Boolean `True` if the argument is not null and it represents an equivalent string. `False` otherwise.

equalsIgnoreCase(String anotherString): boolean

Compares this string to another string, ignoring case considerations. Two strings are considered equal ignoring case if they are of the same length and corresponding characters in the two strings are equal ignoring case.

Input Parameters

anotherString **String** The string to compare this string against.

Return Value

Boolean `True` if the argument is not null and it represents an equivalent string ignoring case. `False` otherwise.

inList(String[] list): boolean

Indicates whether or not this string is in the specified list. The case (upper, lower) of the string is taken into consideration when matching matching against the list.

Input Parameters

list **String List** A list of strings to match this string against.

Return Value

Boolean `True` if this string exists in the specified list. `False` otherwise.

inList(String[] list, boolean ignoreCase): boolean

Indicates whether or not this string is in the specified list. Case sensitivity can be ignored while checking.

Input Parameters

list **String List** A list of strings to match this string against.

ignoreCase **Boolean** Indicates whether or not case sensitivity should be ignored.

Return Value

Boolean `True` if this string exists in the specified list. `False` otherwise.

inRange(String lowerBound, String upperBound): boolean

Indicates whether or not this string is within the specified range. The checking is inclusive, meaning that the lower and upper bounds of the range will be tested for the string. Case differences (upper/lower) are not ignored.

Input Parameters

lowerBound **String** The lower bound of the range to check against (inclusive).

upperBound **String** The upper bound of the range to check against (inclusive).

Return Value

Boolean `True` if this string exists within the specified range. `False` otherwise.

inRange(String lowerBound, String upperBound, String[] exclusions): boolean

Indicates whether or not this string is within the specified range. The checking is inclusive, meaning that the lower and upper bounds of the range will be tested for the string. A separate list of exclusions can be provided to indicate that even though the string is within the given range, it should not be accepted if found within the exclusions list. Case differences (upper/lower) are not ignored.

Input Parameters

lowerBound **String** The lower bound of the range to check against (inclusive).

upperBound	**String** The upper bound of the range to check against (inclusive).
exclusions	**String List** An array of items to be excluded from the range check.

Return Value

Boolean `True` if this string exists within the specified range. `False` otherwise.

inRange(String[][] listOfRanges, String[] exclusions): boolean

Indicates whether or not this string is within the specified list of ranges. The checking is inclusive, meaning that the lower and upper bounds of the ranges will be tested for the string. A separate list of exclusions can be provided to indicate that even though the string is within the given list of ranges, it should not be accepted if found within the exclusions list. Case differences (upper/lower) are not ignored.

Input Parameters

listOfRanges	**String List** The list of ranges to check against. This is a two-dimensional string array with the outer dimension containing the list of ranges and the inner dimension containing the upper and lower bounds of each range. The outer dimension can contain 1 - n elements, while the inner dimension must always contain exactly two elements.
exclusions	**String List** An array of items to be excluded from the range check.

Return Value

Boolean `True` if this string exists within the specified list of ranges. `False` otherwise.

inRange(String[][] listOfRanges, boolean ignoreCase, boolean inclusiveLower, boolean inclusiveUpper, String[] exclusions): boolean

Indicates whether or not this string is within the specified list of ranges. A separate list of exclusions can be provided to indicate that even though the string is within any of the lists of ranges, it should not be accepted if found within the exclusions list. Upper and lower end of list can optionally be included. Case sensitivity can be ignored while checking.

Input Parameters

listOfRanges	**String List** The list of ranges to check against. This is a two-dimensional string array with the outer dimension containing the list of ranges and the inner dimension containing the upper and lower bounds of each range. The outer dimension can contain 1 - n elements, while the inner dimension must always contain exactly two elements.
ignoreCase	**Boolean** Indicates whether or not case differences should be ignored.
inclusiveLower	**Boolean** Indicates whether or not to include the lower end of each range.
inclusiveUpper	**Boolean** Indicates whether or not to include the upper end of each range.
exclusions	**String List** An array of items to be excluded from the range check.

Return Value

Boolean `True` if this string exists within the specified list of ranges. `False` otherwise.

inRange(Long lowerBound, Long upperBound): boolean

Indicates whether or not this integer (long, integer, short) value is within the specified range. The checking is inclusive, meaning that the lower and upper bound of the range will be tested for the long value.

Note: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$. Long, integer and short data types can be passed as arguments to the function. The inRange function is overloaded and supports other numeric ranges such as double and float. To ensure that the correct signature is invoked, make sure that the proper numeric syntax is used (no decimal for integer, decimal for floating point).

Input Parameters

lowerBound	**Integer** The lower bound of the range to check against (inclusive).
upperBound	**Integer** The upper bound of the range to check against (inclusive).

Return Value

Boolean `True` if this integer exists within the specified range. `False` otherwise.

inRange(Long lowerBound, Long upperBound, Long[] exclusions): boolean

Indicates whether or not this integer (long, integer, short) value is within the specified range. The checking is inclusive, meaning that the lower and upper bounds of the range will be tested for the long value. A separate list of exclusions can be provided to indicate that even though the long value is within the range, it should not be accepted if found within the exclusions list.

Note: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$. Long, integer and short data types can be passed as arguments to the function. The inRange function is overloaded and supports other numeric ranges such as double and float. To ensure that the correct signature is invoked, make sure that the proper numeric syntax is used (no decimal for integer, decimal for floating point).

Input Parameters

lowerBound	**Integer** The lower bound of the range to check against (inclusive).
upperBound	**Integer** The upper bound of the range to check against (inclusive).
exclusions	**Integer List** An array of items to be excluded from the range check.

Return Value

Boolean `True` if this integer exists within the specified range. `False` otherwise.

inRange(Long[][] listOfRanges, Long[] exclusions): boolean

Indicates whether or not this integer (long, integer, short) value is within the specified list of ranges. The checking is inclusive, meaning that the lower and upper bounds of the ranges will be tested for the long value. A separate list of exclusions can be provided to indicate that even though the long value is within the list of ranges, it should not be accepted if found within the exclusions list.

Note: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$. Long, integer and short data types can be passed as arguments to the function. The inRange function is overloaded and supports other numeric ranges such as double and float. To ensure that the correct signature is invoked, make sure that the proper numeric syntax is used (no decimal for integer, decimal for floating point).

Input Parameters

listOfRanges	**Integer List** The list of ranges to check against. This is a two-dimensional long array with the outer dimension containing the list of ranges and the inner dimension containing the upper and lower bounds of each range. The outer dimension can contain 1-n elements, while the inner dimension must always contain exactly two elements.

exclusions	**Integer List** An array of items to be excluded from the range check.

Return Value

Boolean `True` if this integer exists within the specified list of ranges. `False` otherwise.

inRange(Long[][] listOfRanges, boolean inclusiveLower, boolean inclusiveUpper, Long[] exclusions): boolean

Indicates whether or not this integer (long, integer, short) value is within the specified list of ranges. A separate list of exclusions can be provided to indicate that even though the long value is within the list of ranges, it should not be accepted if found within the exclusions list. Upper and lower end of list can optionally be included.

Note: The long data type is a 64-bit two's complement integer. The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$. Long, integer and short data types can be passed as arguments to the function. The inRange function is overloaded and supports other numeric ranges such as double and float. To ensure that the correct signature is invoked, make sure that the proper numeric syntax is used (no decimal for integer, decimal for floating point).

Input Parameters

listOfRanges	**Integer List** The list of ranges to check against. This is a two-dimensional long array with the outer dimension containing the list of ranges and the inner dimension containing the upper and lower bounds of each range. The outer dimension can contain 1-n elements, while the inner dimension must always contain exactly two elements.
inclusiveLower	**Boolean** Indicates whether or not to include the lower end of each range.
inclusiveUpper	**Boolean** Indicates whether or not to include the upper end of each range.
exclusions	**Integer List** An array of items to be excluded from the range check.

Return Value

Boolean `True` if this integer exists within the specified list of ranges. `False` otherwise.

inRange(Double lowerBound, Double upperBound): boolean

Indicates whether or not this floating point (double, float) value is within the specified range. The checking is inclusive, meaning that the lower and upper bound of the range will be tested for the double value.

Note: The double data type is a double-precision 64-bit IEEE 754 floating point. A double literal is of type double if it contains a decimal (e.g., 7.9). Double and float data types can be passed as arguments to the function. The inRange function is overloaded and supports other numeric ranges such as integer, short, long, etc. To ensure that the correct signature is invoked, make sure that the proper numeric syntax is used (no decimal for integer, decimal for floating point).

Input Parameters

lowerBound	**Integer** The lower bound of the range to check against (inclusive).
upperBound	**Integer** The upper bound of the range to check against (inclusive).

Return Value

Boolean `True` if this integer exists within the specified range. `False` otherwise.

inRange(Double lowerBound, Double upperBound, Double[] exclusions): boolean

Indicates whether or not this floating point (double, float) value is within the specified range. The checking is inclusive, meaning that the lower and upper bounds of the range will be tested for the double value. A separate list of exclusions can be provided to indicate that even though the double value is within the range, it should not be accepted if found within the exclusions list.

Note: The double data type is a double-precision 64-bit IEEE 754 floating point. A double literal is of type double if it contains a decimal (e.g., 7.9). Double and float data types

can be passed as arguments to the function. The inRange function is overloaded and supports other numeric ranges such as integer, short, long, etc. To ensure that the correct signature is invoked, make sure that the proper numeric syntax is used (no decimal for integer, decimal for floating point).

Input Parameters

lowerBound	**Integer** The lower bound of the range to check against (inclusive).
upperBound	**Integer** The upper bound of the range to check against (inclusive).
exclusions	**Integer List** An array of items to be excluded from the range check.

Return Value

Boolean `True` if this integer exists within the specified range. `False` otherwise.

inRange(Double[][] listOfRanges, Double[] exclusions): boolean

Indicates whether or not this floating point (double, float) value is within the specified list of ranges. The checking is inclusive, meaning that the lower and upper bounds of the ranges will be tested for the double value. A separate list of exclusions can be provided to indicate that even though the double value is within the list of ranges, it should not be accepted if found within the exclusions list.

Note: The double data type is a double-precision 64-bit IEEE 754 floating point. A double literal is of type double if it contains a decimal (e.g., 7.9). Double and float data types can be passed as arguments to the function. The inRange function is overloaded and supports other numeric ranges such as integer, short, long, etc. To ensure that the correct signature is invoked, make sure that the proper numeric syntax is used (no decimal for integer, decimal for floating point).

Input Parameters

listOfRanges	**Integer List** The list of ranges to check against. This is a two-dimensional long array with the outer dimension containing the list of ranges and the inner dimension containing the upper and lower bounds of each range.

	The outer dimension can contain 1-n elements, while the inner dimension must always contain exactly two elements.
exclusions	**Integer List** An array of items to be excluded from the range check.

Return Value

Boolean `True` if this integer exists within the specified list of ranges. `False` otherwise.

inRange(Double[][] listOfRanges, boolean inclusiveLower, boolean inclusiveUpper, Double[] exclusions): boolean

Indicates whether or not this floating point (double, float) value is within the specified list of ranges. A separate list of exclusions can be provided to indicate that even though the double value is within the list of ranges, it should not be accepted if found within the exclusions list. Upper and lower end of list can optionally be included.

Note: The double data type is a double-precision 64-bit IEEE 754 floating point. A double literal is of type double if it contains a decimal (e.g., 7.9). Double and float data types can be passed as arguments to the function. The inRange function is overloaded and supports other numeric ranges such as integer, short, long, etc. To ensure that the correct signature is invoked, make sure that the proper numeric syntax is used (no decimal for integer, decimal for floating point).

Input Parameters

listOfRanges	**Integer List** The list of ranges to check against. This is a two-dimensional long array with the outer dimension containing the list of ranges and the inner dimension containing the upper and lower bounds of each range. The outer dimension can contain 1-n elements, while the inner dimension must always contain exactly two elements.
inclusiveLower	**Boolean** Indicates whether or not to include the lower end of each range.
inclusiveUpper	**Boolean** Indicates whether or not to include the upper end of each range.

exclusions **Integer List** An array of items to be excluded from the range check.

Return Value

Boolean `True` if this integer exists within the specified list of ranges. `False` otherwise.

matches(String regex): boolean

Tells whether or not this string matches the given regular expression.

Input Parameters

regex **String** The regular expression to which this string is to be matched.

Return Value

Boolean `True` if this string matches the given regular expression. `False` otherwise.

regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len): boolean

Tests if two string regions are equal. A substring of this string object is compared to a substring of the argument other. The result is true if these substrings represent identical character sequences. The substring of this string object to be compared begins at index `toffset` and has length `len`. The substring of other to be compared begins at index `ooffset` and has length `len`. The result is false if and only if at least one of the following is true:

- `toffset` is negative.
- `ooffset` is negative.
- `toffset+len` is greater than the length of this string object.
- `ooffset+len` is greater than the length of the other argument.
- There is some nonnegative integer `k` less than `len` such that: `this.charAt(toffset +k) != other.charAt(ooffset+k).`

Case sensitivity can be ignored while checking.

Input Parameters

ignoreCase	**Boolean** Indicates whether or not case should be ignored when comparing characters.
toffset	**Integer** The starting offset of the subregion in this string.
other	**String** The string argument.
ooffset	**Integer** The starting offset of the subregion in the string argument.
len	**Integer** The number of characters to compare.

Return Value

Boolean `True` true if these substrings represent identical character sequences. `False` otherwise.

replaceAll(String regex, String replacement): String

Replaces each substring of this string that matches the given regular expression with the given replacement.

Input Parameters

regex	**String** The regular expression to which this string is to be matched.
replacement	**String** The string to be substituted for each match.

Return Value

String The resulting string.

replaceFirst(String regex, String replacement): String

Replaces the first substring of this string that matches the given regular expression with the given replacement.

Input Parameters

regex	**String** The regular expression to which this string is to be matched.
replacement	**String** The string to be substituted for each match.

Return Value

String The resulting string.

startsWith(String prefix): boolean

Tests if this string starts with the specified prefix.

Input Parameters

prefix	**String** The prefix.

Return Value

Boolean `True` if the character sequence represented by the argument is a prefix of the character sequence represented by this string. `False` otherwise.

Note: The result will be true if the argument is the empty string or is equal to this string object as determined by the equals(String) method.

substring(int beginIndex): String

Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

Input Parameters

beginIndex	**Integer** The beginning index, inclusive.

Return Value

String The specified substring.

substring(int beginIndex, int endIndex): String

Returns a new string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex-beginIndex`.

Input Parameters

beginIndex	**Integer** The beginning index, inclusive.
endIndex	**Integer** The ending index, exclusive.

Return Value

String The specified substring.

toLowerCase(): String

Converts all of the characters in this string to lower case.

Input Parameters

None.

Return Value

String The string converted to lower case.

toUpperCase(): String

Converts all of the characters in this string to upper case.

Input Parameters

None.

Return Value

String The string converted to upper case.

trim(): String

Returns a copy of the string, with leading and trailing whitespace omitted.

Input Parameters

None.

Return Value

String A copy of this string with leading and trailing white space removed, or this string if it has no leading or trailing white space.

Adding a Function

My webMethods provides a set of predefined functions for decision table conditions, decision table assignment results or event rule assignment results.

To add a function:

1. For decision tables, execute steps 1 to 3 as described in "Adding a Condition or Result Value with the Cell Editor" on page 36. For event rules, execute steps 1 to 3 as described in "Adding a Result Value with the Cell Editor" on page 45.
2. In the *[Value Type]* Modification dialog box, select an operator as specified in "About Condition Operators" on page 38 (for decision table conditions), "About Result

Operators" on page 39 (for decision table assignment results) or "About Result Operators" on page 47 (for event rule assignment results).

3. Select the **Functions** tab.
4. The function always refers to the source element which is the parameter element that was specified for the condition or result (default). To specify a new source element, click ... next to the source element name. In the Parameter Elements dialog box, select a parameter element from the list of parameters that are used in the decision entity, and click Ok. To restore the default, click Reset next to the source element name.
5. To add a function, click + in the upper right corner of the **Functions** tab. In the Add Function dialog box, select a function from the list of predefined functions, and click Ok. To chain functions, repeat Step 5.
6. To remove a function you added, select the check box next to the function name, and click − in the upper right corner of the **Functions** tab.
7. To specify string input parameters of a function, do one of the following:
   a. Click ... next to the input field that is located below the function name. In the Parameter Elements dialog box, select a parameter element from the list of parameters that are used in the decision entity and that are of the data type specified for the condition or result. You can also select the EMPTY STRING constant from the dialog box. Click Ok.
   b. Enter a literal value manually in the input field below the function name.

   Note: You cannot enter a parameter element manually. Typed values such as `%"input.parameter"%` are interpreted as literal values.
8. To specify range input parameters of a function, do one of the following:
   a. Click ... next to the input fields that are located below the function name. In the Parameter Elements dialog box, select a parameter element from the list of parameters that are used in the decision entity and that are of the data type specified for the condition or result. For strings, you can also select the EMPTY STRING constant from the dialog box. Click Ok.
   b. Enter a literal value manually in the input fields below the function name.

   Note: You cannot enter a parameter element manually. Typed values such as `%"input.parameter"%` are interpreted as literal values.
9. To specify list input parameters of a function, do one of the following:
   a. Click + in the blue field that is located below the function name. Click ... next to the input field that appears after you clicked +. In the Parameter Elements dialog box, select a parameter element from the list of parameters that are used in the decision entity and that are of the data type specified for the condition or result. For strings, you can also select the EMPTY STRING constant from the dialog box. Click Ok. Repeat the sub-step to specify multiple list input parameters.

b. Enter a literal value manually in the input field that appears after you clicked  . Repeat the sub-step to specify multiple list input parameters.

> **Note:** You cannot enter a parameter element manually. Typed values such as `%"input.parameter"%` are interpreted as literal values.

10. To specify boolean input parameters of a function, select **true** or **false** from the drop down list that is located below the function name.
11. Click 

.