webMethods Business Rules Reference

April 2015

This document applies to webMethods Business Rules 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: RULES-REF-98-20150415

Table of Contents

About this Guide

webMethods Rules Reference Help describes the structure of rule functions, rules-related event types and wMBusinessRules built-in services that are part of the Rules Development feature of Software AG Designer.

webMethods Rules Reference Help contains supporting documentation on the following main topics:

- "Functions Overview" on page 9.
- "Rules-Related Event Types Overview" on page 69.
- "WmBusinessRules Built-in Services Overview" on page 73.

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 (+).
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.

Convention	Description
[ ]	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 (...).

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at http://techcommunity.softwareag.com. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

1 Functions Overview

webMethods Rules Development provides a set of predefined functions. For detailed information about how to work with functions, see *webMethods BPM Rules Development Help*.

Five categories of functions exist:

- "Summary of Conversion Functions" on page 13.
- "Summary of Date Functions" on page 19.
- "Summary of List and Range Functions" on page 30.
- "Summary of Math Functions" on page 45.
- "Summary of String Functions" on page 59.

Summary of Conversion Functions

webMethods Rules Development provides the following predefined conversion functions:

Function	Returns	Description
toBoolean()	Boolean	Returns a boolean with a value represented by this string.
toDate()	Date	Returns a date that this string represents.
toDate()	Date Object	Allocates a date object for this long value.
toDate(String dateFormat)	Date	Returns a date that represents this string based on the specified format.
toDouble()	Double	Returns a double value holding the value of this string.
toDouble()	Double	Returns a double value represented by this long value.
toLong()	Long	Returns a long value holding the value of this string.

Function	Returns	Description
toLong()	Long	Returns a long value represented by this double value.
toString()	String	Returns a string representing the value of this long value.
toString()	String	Returns a string representing the value of this double value.
toString()	String Object	Returns a string object representing the value of this boolean.
toString()	String	Returns a string representing the value of this date.
toString(String dateFormat)	String	Returns a string representing the value of this date in the specified format.

toBoolean()

Returns a boolean with a value represented by this string. The boolean returned represents a true value if the string argument is not null and is equal, ignoring case, to the string `true`.

Input Parameters

None.

Return Value

Boolean The boolean value represented by this string.

toDate()

Returns a date that this string represents.

Input Parameters

None.

Return Value

Date A date if it can be determined, `null` otherwise.

toDate()

Allocates a date object and initializes it to represent the number of milliseconds since the standard base time known as "the epoch" (namely January 1, 1970, 00:00:00 GMT) for this long value.

Input Parameters

None.

Return Value

Date Object A date object.

toDate(String dateFormat)

Returns a date that represents this string based on the specified format.

Input Parameters

dateFormat	**String** A pattern that defines the format of this date. The pattern is based on Java date and time patterns.

Return Value

Date A date that represents this string based on the specified format.

toDouble()

Returns a double value holding the value of this string.

Input Parameters

None.

Return Value

Double A double value holding the value of this string.

toDouble()

Returns a double value represented by this long value.

Input Parameters

None.

Return Value

Double A double value represented by this long value.

toLong()

Returns a long value holding the value of this string.

Input Parameters

None.

Return Value

Long A long value holding the value of this string.

toLong()

Returns a long value represented by this double value.

Input Parameters

None.

Return Value

Long A long value holding the value (truncated toward zero) of this double value.

toString()

Returns a string representing the value of this long value.

Input Parameters

None.

Return Value

String A string representation of the value of this object in base 10.

toString()

Returns a string representing the value of this double value.

Input Parameters

None.

Return Value

String A string representation of the value of this object in base 10.

toString()

Returns a string object representing the value of this boolean. If this object represents the value `true`, a string equal to `true` is returned. Otherwise, a string equal to `false` is returned.

Input Parameters

None.

Return Value

String A string representation of the value of this object.

toString()

Returns a string representing the value of this date based on the locale that the application is running under.

Input Parameters

None.

Return Value

String A string representing the value of this date based on the locale that the application is running under.

toString(String dateFormat)

Returns a string representing the value of this date in the specified format.

Input Parameters

DateFormat	**String** A pattern that defines the format of this date. The pattern is based on Java date and time patterns.

Return Value

String A string representing the value of this date in the specified format.

Summary of Date Functions

webMethods Rules Development provides the following predefined date functions:

Note: Dates are based on the ISO 8601 standard which is based on the proleptic Gregorian calendar.

Function	Returns	Description
int century()	Integer	Returns the century for this date.
int compareDates(Date date1, Date date2)	Integer	Compares one date against another date.
Date date()	Date	Returns the current date from the server where the decision entity is invoked.
Date dateMinusDays(int days)	Date	Returns a copy of this date minus the specified number of days.
Date datePlusDays(int days)	Date	Returns a copy of this date plus the specified number of days.
long dateToInt()	Integer	Returns the number of milliseconds since the standard base time.
int dayOfMonth()	Integer	Returns the day of the month for this date.
int dayOfWeek()	Integer	Returns the day of the week for this date.

Function	Returns	Description
int dayOfYear()	Integer	Returns the day of the year for this date.
int daysInMonth(int month, int year)	Integer	Returns the number of days in the specified month in the given year.
long diffInDays(Date anotherDate)	Integer	Returns the mathematical difference in days between this date and the specified date ignoring the times of both dates.
long diffInMonths(Date anotherDate)	Integer	Returns the mathematical difference in months between this date and the specified date ignoring the times of both dates.
long diffInYears(Date anotherDate)	Integer	Returns the mathematical difference in years between this date and the specified date ignoring the times of both dates.
Date intToDate(long millis)	Date	Returns a date that represents the specified number of milliseconds since the standard base time.
isLeapYear(int year)	Boolean	Indicates whether or not the specified year is a leap year.
int month()	Integer	Returns the numeric month for this date.
String time()	String	Returns the string representation of the time this date as Hours:Minutes:Seconds.Milliseconds.

Function	Returns	Description
boolean verifyDate(int year, int month, int day)	Boolean	Verifies that the date represented by the specified year, month and day is valid.
boolean verifyIntDate(long millis)	Boolean	Verifies that the specified number of milliseconds is a valid date representation.
boolean verifyMonth(int month)	Boolean	Verifies that the specified numeric month is valid.
boolean verifyYear(int year)	Boolean	Verifies that the specified numeric year is valid.
int year()	Integer	Returns the numeric 4-digit year for this date.
Date ymdToDate(int year, int month, int day)	Date	Creates and returns a date from three integers (year, month, day).

int century()

Returns the century for this date. The century is based on the era, where era is expressed as a constant, zero for BC/BCE, one for AD/CE.

Input Parameters

None.

Return Value

Integer The century for this date.

int compareDates(Date date1, Date date2)

Compares one date against another date and returns `+1` if the first date is greater than the second date, `-1` if the first date is less than the second date, and `0` if the dates are equal.

Input Parameters

date1	**Date** The first date to compare.
date2	**Date** The second date to compare against the first date.

Return Value

Integer The value `0` if dates are equal. The value `-1` if the first date is before the second date. The value `+1` if the first date is after the second date.

Date date()

Returns the current date from the server where the decision entity is invoked, measured to the nearest millisecond.

Input Parameters

None.

Return Value

Date The current date from the server where the decision entity is invoked, measured to the nearest millisecond.

Date dateMinusDays(int days)

Returns a copy of this date minus the specified number of days.

Input Parameters

days	**Integer** The amount of days to subtract, may be negative.

Return Value

Date The new date minus the specified days.

Date datePlusDays(int days)

Returns a copy of this date plus the specified number of days.

Input Parameters

days	**Integer** The amount of days to add, may be negative.

Return Value

Date The new date plus the specified days.

long dateToInt()

Returns the number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT represented by this date.

Input Parameters

None.

Return Value

Integer The number of milliseconds since January 1, 1970, 00:00:00 GMT represented by this date.

int dayOfMonth()

Returns the day of the month for this date (1 - 31 depending on the month).

Input Parameters

None.

Return Value

Integer The day of month for this date.

int dayOfWeek()

Returns the day of the week represented by this date. The returned value (1 = Sunday, 2 = Monday, 3 = Tuesday, 4 = Wednesday, 5 = Thursday, 6 = Friday, 7 = Saturday) represents the day of the week that contains or begins with the instant in time represented by this date, as interpreted in the local time zone.

Input Parameters

None.

Return Value

Integer The day of week for this date.

int dayOfYear()

Returns the day of the year for this date (1 - 366 depending on the year).

Input Parameters

None.

Return Value

Integer The day of the year for this date.

int daysInMonth(int month, int year)

Returns the number of days in the specified month in the given year.

Input Parameters

month	**Integer** The numeric month (1 = January, 2 = February, ..., 12 = December).
year	**Integer** The numeric 4-digit year.

Return Value

Integer The number of days in the specified month in the given year.

long diffInDays(Date anotherDate)

Returns the mathematical difference in days between this date and the specified date ignoring the times of both dates.

Input Parameters

anotherDate	**Date** The date to be used to compute the mathematical difference in days from this date.

Return Value

Integer The mathematical difference in days between this date and the specified date ignoring the times of both dates.

long diffInMonths(Date anotherDate)

Returns the mathematical difference in months between this date and the specified date ignoring the times of both dates.

Input Parameters

anotherDate	**Date** The date to be used to compute the mathematical difference in months from this date.

Return Value

Integer The mathematical difference in months between this date and the specified date ignoring the times of both dates.

long diffInYears(Date anotherDate)

Returns the mathematical difference in years between this date and the specified date ignoring the times of both dates.

Input Parameters

anotherDate	**Date** The date to be used to compute the mathematical difference in years from this date.

Return Value

Integer The mathematical difference in years between this date and the specified date ignoring the times of both dates.

Date intToDate(long millis)

Returns a date that represents the specified number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT.

Input Parameters

millis	**Integer** The number of milliseconds since January 1, 1970, 00:00:00 GMT.

Return Value

Date A date that represents the specified number of milliseconds since January 1, 1970, 00:00:00 GMT.

isLeapYear(int year)

Indicates whether or not the specified year is a leap year.

Input Parameters

year	**Integer** The year to test the leap for.

Return Value

Boolean Returns `true` if the specified year is a leap year, `false` otherwise.

int month()

Returns the numeric month for this date (1 = January, 2 = February, ..., 12 = December).

Input Parameters

None.

Return Value

Integer The numeric month for this date.

String time()

Returns the string representation of the time of this date using the default formatting style of the default locale.

Input Parameters

None.

Return Value

String The time for this date.

boolean verifyDate(int year, int month, int day)

Verifies that the date represented by the specified year, month and day is valid.

Input Parameters

year	**Integer** The 4-digit year of the date.
month	**Integer** The numeric month within the specified year (1 = January, 2 = February, ..., 12 = December).
day	**Integer** The numeric day within the specified month.

Return Value

Boolean Returns `true` if the date represented by the specified year, month and day is valid, `false` otherwise.

boolean verifyIntDate(long millis)

Verifies that the specified number of milliseconds is a valid date representation. (Should be the number of milliseconds since the standard base time known as "the epoch", namely January 1, 1970, 00:00:00 GMT.)

Input Parameters

millis	**Integer** The number of milliseconds since January 1, 1970, 00:00:00 GMT.

Return Value

Boolean Returns `true` if the specified number of milliseconds is valid, `false` otherwise.

boolean verifyMonth(int month)

Verifies that the specified numeric month is valid.

Input Parameters

month	**Integer** The numeric month within the specified year (1 = January, 2 = February, ..., 12 = December).

Return Value

Boolean Returns `true` if the specified month is valid, `false` otherwise.

boolean verifyYear(int year)

Verifies that the specified numeric year is valid.

Input Parameters

year	**Integer** The 4-digit year of the date.

Return Value

Boolean Returns `true` if the specified year is valid, `false` otherwise.

int year()

Returns the numeric 4-digit year for this date.

Input Parameters

None.

Return Value

Integer The numeric 4-digit year for this date.

Date ymdToDate(int year, int month, int day)

Creates and returns a date from three integers (year, month, day).

Input Parameters

year	**Integer** The 4-digit year of the date.
month	**Integer** The numeric month within the specified year (1 = January, 2 = February, ..., 12 = December).
day	**Integer** The numeric day within the specified month.

Return Value

Date A date for the specified year, month and day.

Summary of List and Range Functions

webMethods Rules Development provides the following predefined list and range functions:

Function	Returns	Description
boolean inList(String[] list)	Boolean	Indicates whether or not this string is in the specified list.
boolean inList(String[] list, boolean ignoreCase)	Boolean	Indicates whether or not this string is in the specified list. Case sensitivity can be ignored while checking.
boolean inRange(Date lowerBound, Date upperBound)	Boolean	Indicates whether or not this date is within the specified range.
boolean inRange(Date lowerBound, Date upperBound, Date[] exclusions)	Boolean	Indicates whether or not this date is within the specified range. Exclusions can be specified.
boolean inRange(Date[][] listOfRanges, Date[] exclusions)	Boolean	Indicates whether or not this date is within the specified list of ranges. Exclusions can be specified.
boolean inRange(Date[][] listOfRanges, boolean inclusiveLower, boolean inclusiveUpper, Date[] exclusions)	Boolean	Indicates whether or not this date is within the specified list of ranges. Exclusions and inclusion of lower and upper end of range can be specified.
boolean inRange(Double lowerBound, Double upperBound)	Boolean	Indicates whether or not this floating point (double, float) value is within the specified range.
boolean inRange(Double lowerBound, Double upperBound, Double[] exclusions)	Boolean	Indicates whether or not this floating point (double, float) value is within the specified range. A separate list of exclusions can be provided.
boolean inRange(Double[][] listOfRanges, Double[] exclusions)	Boolean	Indicates whether or not this floating point (double, float) value is within the specified

Function	Returns	Description
		list of ranges. A separate list of exclusions can be provided.
boolean 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. Upper and lower end of list can optionally be included. A separate list of exclusions can be provided.
boolean inRange(Long lowerBound, Long upperBound)	Boolean	Indicates whether or not this integer (long, integer, short) value is within the specified range.
boolean inRange(Long lowerBound, Long upperBound, Long[] exclusions)	Boolean	Indicates whether or not this integer (long, integer, short) value is within the specified range. A separate list of exclusions can be provided.
boolean inRange(Long[][] listOfRanges, 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.
boolean 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. Upper and lower end of list can optionally be included. A separate list of exclusions can be provided.
boolean inRange(String lowerBound, String upperBound)	Boolean	Indicates whether or not this string is within the specified range.
boolean inRange(String lowerBound, String upperBound, String[] exclusions)	Boolean	Indicates whether or not this string is within the specified range. A separate list of exclusions can be provided.

Function	Returns	Description
boolean inRange(String[][] listOfRanges, String[] exclusions)	Boolean	Indicates whether or not this string is within the specified list of ranges. A separate list of exclusions can be provided.
boolean 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. 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.

boolean inList(String[] list)

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 Returns `true` if this string exists in the specified list, `false` otherwise.

boolean inList(String[] list, boolean ignoreCase)

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 Returns `true` if this string exists in the specified list, `false` otherwise.

boolean inRange(Date lowerBound, Date upperBound)

Indicates whether or not this date is within the specified range. The checking is inclusive, meaning that the lower and upper bound of the range will be tested for the date.

Input Parameters

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

Return Value

Boolean Returns `true` if this date exists within the specified range, `false` otherwise.

boolean inRange(Date lowerBound, Date upperBound, Date[] exclusions)

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

Input Parameters

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

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

Return Value

Boolean Returns `true` if this date exists within the specified range, `false` otherwise.

boolean inRange(Date[][] listOfRanges, Date[] exclusions)

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

Input Parameters

listOfRanges	**Date List** The list of ranges to check against. This is a two-dimensional date 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	**Date** An array of items to be excluded from the range check.

Return Value

Boolean Returns `true` if this date exists within the specified list of ranges, `false` otherwise.

boolean inRange(Date[][] listOfRanges, boolean inclusiveLower, boolean inclusiveUpper, Date[] exclusions)

Indicates whether or not this date is within the specified list of ranges. A separate list of exclusions can be provided to indicate that even though the date is within the list of ranges, it should not be accepted if found within the exclusions list.

Input Parameters

listOfRanges	**Date List** The list of ranges to check against. This is a two-dimensional date 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	**Date** An array of items to be excluded from the range check.

Return Value

Boolean Returns `true` if this date exists within the specified list of ranges, `false` otherwise.

boolean inRange(Double lowerBound, Double upperBound)

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 Returns `true` if this integer exists within the specified range, `false` otherwise.

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

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 Returns `true` if this integer exists within the specified range, `false` otherwise.

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

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 Returns `true` if this integer exists within the specified list of ranges, `false` otherwise.

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

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 Returns `true` if this integer exists within the specified list of ranges, `false` otherwise.

boolean inRange(Long lowerBound, Long upperBound)

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 Returns `true` if this integer exists within the specified range, `false` otherwise.

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

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 Returns `true` if this integer exists within the specified range, `false` otherwise.

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

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 Returns `true` if this integer exists within the specified list of ranges, `false` otherwise.

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

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 Returns `true` if this integer exists within the specified list of ranges, `false` otherwise.

boolean inRange(String lowerBound, String upperBound)

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 Returns `true` if this string exists within the specified range, `false` otherwise.

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

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 Returns `true` if this string exists within the specified range, `false` otherwise.

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

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 Returns `true` if this string exists within the specified list of ranges, `false` otherwise.

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

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 Returns `true` if this string exists within the specified list of ranges, `false` otherwise.

Summary of Math Functions

webMethods Rules Development provides the following predefined math functions:

Function	Returns	Description
long abs(long value)	Integer	Returns the absolute value of the specified long value.
double abs(long double)	Integer	Returns the absolute value of the specified double value.
double acos(double val)	Integer	Returns the arc cosine of the specified double value.
double asin(double val)	Integer	Returns the arc sine of the specified double value.
double atan(double val)	Integer	Returns the arc tangent of the specified double value.
double ceil(double val)	Integer	Returns the smallest integer that is greater than or equal to the specified double value.
double cos(double val)	Integer	Returns the trigonometric cosine of the specified angle.
double cosh(double val)	Integer	Returns the hyperbolic cosine of the specified double value.
double degreesToRadians(double angdeg)	Integer	Returns an approximately equivalent angle measured in radians for the specified angle measured in degrees.
double exp(double val)	Integer	Returns Euler's number `e` raised to the power of the specified double value.
double floor(double val)	Integer	Returns the largest integer that is less than or equal to the specified double value.
double log(double val)	Integer	Returns the natural logarithm (base e) of the specified double value.

Function	Returns	Description
long max(long val1, long val2)	Integer	Returns the larger of the two specified long values. If the specified values are equal, then the result is that same value.
double max(double val1, double val2)	Integer	Returns the larger of the two specified double values. If the specified values are equal, then the result is that same value.
long min(long val1, long val2)	Integer	Returns the lesser of the two specified long values. If the specified values are equal, then the result is that same value.
double min(double val1, double val2)	Integer	Returns the lesser of the two specified double values. If the specified values are equal, then the result is that same value.
long mod(long val1, long val2)	Integer	Returns the remainder of two long values. The remainder is obtained when dividing `val1` by `val2`.
double mod(double val1, double val2)	Integer	Returns the remainder of two double values. The remainder is obtained when dividing `val1` by `val2`.
double pi()	Integer	Returns the value of pi to 15 decimal places.
double pow(double base, double exponent)	Integer	Returns $base^{exponent}$ or the value of the base argument raised to the power of the exponent argument.
double radiansToDegrees(double angrad)	Integer	Returns an approximately equivalent angle measured in degrees for the specified angle measured in radians.

Function	Returns	Description
long round(double val)	Integer	Returns the closest long integer to the specified double argument, with with ties rounding up.
double round(double val, int scale)	Integer	Rounds the given value to the specified number of decimal places.
double round(double val, int scale, int roundingMethod)	Integer	Rounds the given value to the specified number of decimal places. The value is rounded using the given method which is any method defined in `java.math.BigDecimal`.
double sin(double val)	Integer	Returns the trigonometric sine of the specified angle.
double sinh(double val)	Integer	Returns the hyperbolic sine of the specified double value.
double tan(double val)	Integer	Returns the trigonometric tangent of the specified angle.
double tanh(double val)	Integer	Returns the hyperbolic tangent of the specified double value.

long abs(long value)

Returns the absolute value of the specified long value.

Input Parameters

val **Integer** The argument whose absolute value is to be determined.

Return Value

Integer The absolute value of the argument.

double abs(long double)

Returns the absolute value of the specified double value.

Input Parameters

val	**Integer** The argument whose absolute value is to be determined.

Return Value

Integer The absolute value of the argument.

double acos(double val)

Returns the arc cosine of the specified double value.

Input Parameters

val	**Integer** The value whose arc cosine is to be returned.

Return Value

Integer The arc cosine of the argument.

double asin(double val)

Returns the arc sine of the specified double value.

Input Parameters

val	**Integer** The value whose arc sine is to be returned.

Return Value

Integer The arc sine of the argument.

double atan(double val)

Returns the arc tangent of the specified double value.

Input Parameters

val **Integer** The value whose arc tangent is to be returned.

Return Value

Integer The arc tangent of the argument.

double ceil(double val)

Returns the smallest integer that is greater than or equal to the specified double value.

Input Parameters

val **Integer** The value whose ceiling is to be returned.

Return Value

Integer The smallest (closest to negative infinity) floating-point value that is greater than or equal to the argument and is equal to a mathematical integer.

double cos(double val)

Returns the trigonometric cosine of the specified angle.

Input Parameters

val **Integer** An angle in radians.

Return Value

Integer The cosine of the argument.

double cosh(double val)

Returns the hyperbolic cosine of the specified double value.

Input Parameters

val	**Integer** The number whose hyperbolic cosine is to be returned.

Return Value

Integer The hyperbolic cosine of the argument.

double degreesToRadians(double angdeg)

Returns an approximately equivalent angle measured in radians for the specified angle measured in degrees.

Input Parameters

angdeg	**Integer** An angle in degrees.

Return Value

Integer The measurement of the angle angdeg in radians.

double exp(double val)

Returns Euler's number `e` raised to the power of the specified double value.

Input Parameters

val	**Integer** The exponent to raise `e` to.

Return Value

Integer Value e^{val}, where `e` is the base of the natural logarithms.

double floor(double val)

Returns the largest integer that is less than or equal to the specified double value.

Input Parameters

val	**Integer** The value whose floor is to be returned.

Return Value

Integer The largest (closest to positive infinity) floating-point value that less than or equal to the argument and is equal to a mathematical integer.

double log(double val)

Returns the natural logarithm (base e) of the specified double value.

Input Parameters

val	**Integer** The number whose natural logarithm (base e) is to be returned.

Return Value

Integer The value `ln val`, the natural logarithm of `val`.

long max(long val1, long val2)

Returns the larger of the two specified long values. If the specified values are equal, then the result is that same value.

Input Parameters

val1	**Integer** The first argument.
val2	**Integer** The second argument.

Return Value

Integer The larger of `val1` and `val2`.

double max(double val1, double val2)

Returns the larger of the two specified double values. If the specified values are equal, then the result is that same value.

Input Parameters

val1	**Integer** The first argument.
val2	**Integer** The second argument.

Return Value

Integer The larger of `val1` and `val2`.

long min(long val1, long val2)

Returns the lesser of the two specified long values. If the specified values are equal, then the result is that same value.

Input Parameters

val1	**Integer** The first argument.
val2	**Integer** The second argument.

Return Value

Integer The lesser of `val1` and `val2`.

double min(double val1, double val2)

Returns the lesser of the two specified double values. If the specified values are equal, then the result is that same value.

Input Parameters

val1	**Integer** The first argument.
val2	**Integer** The second argument.

Return Value

Integer The lesser of `val1` and `val2`.

long mod(long val1, long val2)

Returns the remainder of two long values. The remainder is obtained when dividing `val1` by `val2`.

Input Parameters

val1	**Integer** The dividend.
val2	**Integer** The divisor.

Return Value

Integer The remainder after performing the division.

double mod(double val1, double val2)

Returns the remainder of two double values. The remainder is obtained when dividing `val1` by `val2`.

Input Parameters

val1	**Integer** The dividend.
val2	**Integer** The divisor.

Return Value

Integer The remainder after performing the division.

double pi()

Returns the value of pi to 15 decimal places (3.141592653589793).

Input Parameters

None.

Return Value

Integer The value of pi to 15 decimal places.

double pow(double base, double exponent)

Returns $base^{exponent}$ or the value of the base argument raised to the power of the exponent argument.

Input Parameters

base	**Integer** The base.

exponent **Integer** The exponent.

Return Value

Integer The value of base^{exponent}.

double radiansToDegrees(double angrad)

Returns an approximately equivalent angle measured in degrees for the specified angle measured in radians.

Input Parameters

angrad **Integer** An angle in radians.

Return Value

Integer The measurement of the angle `angrad` in degrees.

long round(double val)

Returns the closest long integer to the specified double argument, with with ties rounding up.

Input Parameters

val **Integer** A floating-point value to be rounded to a long.

Return Value

Integer The value of the argument rounded to the nearest long value.

double round(double val, int scale)

Rounds the given value to the specified number of decimal places. The value is rounded using the `BigDecimal.ROUND_HALF_UP` method (rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round up).

Input Parameters

val	**Integer** The value to round.
scale	**Integer** The number of digits to the right of the decimal point.

Return Value

Integer The rounded value.

double round(double val, int scale, int roundingMethod)

Rounds the given value to the specified number of decimal places. The value is rounded using the given method which is any method defined in `java.math.BigDecimal`. BigDecimal values are:

- ROUND_UP = 0, Rounding mode to round away from zero.
- ROUND_DOWN = 1, Rounding mode to round towards zero.
- ROUND_CEILING = 2, Rounding mode to round towards positive infinity.
- ROUND_FLOOR = 3, Rounding mode to round towards negative infinity.
- ROUND_HALF_UP = 4, Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round up.
- ROUND_HALF_DOWN = 5, Rounding mode to round towards "nearest neighbor" unless both neighbors are equidistant, in which case round down.
- ROUND_HALF_EVEN = 6, Rounding mode to round towards the "nearest neighbor" unless both neighbors are equidistant, in which case, round towards the even neighbor.
- ROUND_UNNECESSARY = 7, Rounding mode to assert that the requested operation has an exact result, hence no rounding is necessary.

Input Parameters

val	**Integer** The value to round.
scale	**Integer** The number of digits to the right of the decimal point.
roundingMethod	**Integer** Rounding method as defined in BigDecimal.

Return Value

Integer The rounded value.

double sin(double val)

Returns the trigonometric sine of the specified angle.

Input Parameters

val	**Integer** An angle in radians.

Return Value

Integer The sine of the argument.

double sinh(double val)

Returns the hyperbolic sine of the specified double value.

Input Parameters

val	**Integer** The number whose hyperbolic sine is to be returned.

Return Value

Integer The hyperbolic sine of the argument.

double tan(double val)

Returns the trigonometric tangent of the specified angle.

Input Parameters

val	**Integer** An angle in radians.

Return Value

Integer The tangent of the argument.

double tanh(double val)

Returns the hyperbolic tangent of the specified double value.

Input Parameters

val	**Integer** The number whose hyperbolic tangent is to be returned.

Return Value

Integer The hyperbolic tangent of the argument.

Summary of String Functions

webMethods Rules Development provides the following predefined string functions:

Function	Returns	Description
concat(String str): String	String	Appends the specified string to the end of this string.
contains(String str): boolean	Boolean	Indicates whether or not the specified string is contained within this string.

Function	Returns	Description
endsWith(String suffix): boolean	Boolean	Indicates whether or not this string ends with the specified suffix.
equals(String anotherString): boolean	Boolean	Indicates whether or not this string is equal to the specified string.
equalsIgnoreCase(String anotherString): boolean	Boolean	Indicates whether or not this string is equal to the specified string, ignoring case considerations.
matches(String regex): boolean	Boolean	Indicates whether or not this string matches the given regular expression.
regionMatches(boolean ignoreCase, int toffset, String other, int ooffset, int len): boolean	Boolean	Indicates whether or not two string regions (substrings within the specified strings) are equal.
replaceAll(String regex, String replacement): String	String	Returns a string where each substring of this string that matches the given regular expression is replaced with the specified replacement value.
replaceFirst(String regex, String replacement): String	String	Returns a string where the first substring of this string that matches the given regular expression is replaced with the specified replacement value.
startsWith(String prefix): boolean	Boolean	Indicates whether or not this string starts with the specified prefix.
substring(int beginIndex): String	String	Returns a string that resides within this string.
substring(int beginIndex, int endIndex): String	String	Returns a string that resides within this string.

Function	Returns	Description
toLowerCase(): String	String	Returns a string with all of the characters of this string converted to lower case.
toUpperCase(): String	String	Returns a string with all of the characters of this string converted to upper case.
trim(): String	String	Returns a copy of this string, with leading and trailing empty characters removed.

concat(String str): String

Appends the specified string to the end of this string.

Input Parameters

str **String** The string that is appended 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

Indicates whether or not the specified string is contained within this string.

Input Parameters

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

Return Value

Boolean Returns `true` if this string contains the specified string, `false` otherwise.

endsWith(String suffix): boolean

Indicates whether or not this string ends with the specified suffix.

Input Parameters

suffix **String** The suffix.

Return Value

BooleanReturns `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

Indicates whether or not this string is equal 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. Case (upper/lower) must match in order for the strings to be considered equal.

Input Parameters

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

Return Value

Boolean Returns `true` if the argument is not null and it represents an equivalent string, `false` otherwise.

equalsIgnoreCase(String anotherString): boolean

Indicates whether or not this string is equal to the specified 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 Returns `true` if the argument is not null and it represents an equivalent string ignoring case, `false` otherwise.

matches(String regex): boolean

Indicates 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 Returns `true` if this string matches the given regular expression, `false` otherwise.

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

Indicates whether or not two string regions (substrings within the specified strings) are equal.

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 Returns `true` if the two string regions are equal, `false` otherwise.

replaceAll(String regex, String replacement): String

Returns a string where each substring of this string that matches the given regular expression is replaced with the specified replacement value. If no match is found, then the original string is returned.

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

Returns a string where the first substring of this string that matches the given regular expression is replaced with the specified replacement value. If no match is found, then the original string is returned.

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

Indicates whether or not this string starts with the specified prefix.

Input Parameters

prefix	**String** The prefix.

Return Value

Boolean Returns `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 string that resides within this string. The returned 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 string that resides within this string. The returned 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

Returns a string with all of the characters of this string converted to lower case.

Input Parameters

None.

Return Value

String The string converted to lower case.

toUpperCase(): String

Returns a string with all of the characters of this string converted to upper case.

Input Parameters

None.

Return Value

String The string converted to upper case.

trim(): String

Returns a copy of this string, with leading and trailing empty characters removed.

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.

2 Rules-Related Event Types Overview

webMethods Rules Development provides a set of predefined event types that allow you to monitor rules-related events on the Integration Server or on the My webMethods Server. For detailed information about how to work with rules-related event types, see *webMethods BPM Rules Development Help*. For the existing event types, see "Summary of Rules-Related Event Types" on page 70.

Summary of Rules-Related Event Types

webMethods Rules Development provides the following predefined event types:

Event Type	Emitted On	Description
Decision Entity Changed	My webMethods Server	Is triggered when a difference is detected between two equally named decision entities.
Hot Deployment Started	My webMethods Server	Is triggered when you hot deploy a rule project on the My webMethods Server.
Project Deployed	Integration Server	Is triggered when a rule project is deployed on the Integration Server.
Project Undeployed	Integration Server	Is triggered when a rule project is undeployed from the Integration Server.

Decision Entity Changed

This event type is triggered on the My webMethods Server when a difference is detected between two equally named decision entities. The event instance on the Universal Messaging Server contains the following entries:

- dateTime
- projectName
- sessionId
- host
- port
- user
- decisionEntityType

- decisionEntityName
- correlationId
- changesType (1=rule added, 2=rule deleted, 3=rule changed)
- cellType (1=condition cell, 2=result cell)
- row
- columnName
- parameterElementName
- old Cell Value
- new Cell Value

Hot Deployment Started

This event type is triggered on the My webMethods Server when you hot deploy a rule project. The event instance on the Universal Messaging Server contains the following entries:

- dateTime
- projectName
- sessionId
- host
- port
- user
- deployedISHost (the Integration Server host you configured on the My webMethods Server for hot deployment)
- deployedISPort (the Integration Server port you configured on the My webMethods Server for hot deployment)
- correlationId
- projectVersion

Project Deployed

This event type is triggered when a rule project is deployed on the Integration Server. The event instance on the Universal Messaging Server contains the following entries:

- dateTime
- projectName

- sessionId
- host
- port
- user
- deployedWith
- projectCreatorApp
- correlationId
- projectVersion
- deployedBy

Project Undeployed

This event type is triggered when a rule project is undeployed from the Integration Server. The event instance on the Universal Messaging Server contains the following entries:

- dateTime
- projectName
- sessionId
- host
- port
- user
- correlationId
- projectVersion

3 WmBusinessRules Built-in Services Overview

webMethods Rules Development provides a set of built-in services. For the existing built-in services, see "Summary of WmBusinessRules Built-in Services" on page 74.

Summary of WmBusinessRules Built-in Services

webMethods Rules Development provides the following of built-in services in the WmBusinessRules package:

Element	Package and Description
pub.businessrules.client:invoke	WmBusinessRules. Executes a rule set or decision table and returns the results of execution.

pub.businessrules.client:invoke

WmBusinessRules. Executes a rule set or decision table and returns the results of execution.

Input Parameters

Project Name

String Name of the rule project that contains the rule set or decision table that you want to execute.

> **Note:** The rule project that contains the rule set or decision table that you want to execute must be deployed on the Integration Server.

Invocation Target

String Name of the rule set or decision table that you want to execute.

Specify one of the following:

Specify...	To...
`RS\`*[RuleSetName]*	Define a rule set.
`DT\`*[DecisionTableName]*	Define a decision table.

Create Missing Inputs

Boolean Optional. Creates an empty input parameter without input values if no input parameter is specified for the rule set or decision table. Set to:

- `true` if missing inputs should be created. This is the default.
- `false` if missing inputs should not be created.

Inputs

Document Defines the input parameters for the rule set or decision table.

Note: The generic invoke service pub.businessrules.client:genericInvoke that is included in the WmBusinessRules package demonstrates how to specify inputs.

Key	Description
[Input Parameter Name]	**Document** Defines the input parameter elements and their values.
Fact IData	Parameter instance at runtime.

Desired Outputs

Document List Optional. Defines a list of output parameters that you want the service execution to return.

Note: If no output parameters are specified, all output parameters of the *Invocation Target* are returned. If a specified output parameter does not exist, the specified output parameter is ignored.

Output Parameters

Outputs

Document Defines the output parameters returned by the rule set or decision table.

Key	Description
[Output Parameter Name]	**Document** Defines the output parameter elements and their values.
Fact IData	Parameter instance at runtime.

Usage Note

Dragging and dropping a rule set or decision entity from the Rules Explorer view to a flow service inserts an invoke step that calls the pub.businessrules.client:invoke into the flow service. Software AG Designer specifies the *Project Name* and *Invocation Target* values automatically based on the rule set or decision entity dragged into the flow service. Software AG Designer creates the input parameters under *Service In* > *Inputs* and the outputs parameters under *Service Out* > *Outputs* automatically based on the input and output parameters of the rule set or decision table dragged into the flow service.