Progress®
Corticon®

Corticon.js
Rule Language

Copyright

Last updated with new content: Corticon.js 2.4

Updated: 2025/11/19

Table of Contents

1

Introduction to Corticon.js Rule Language

Graphical modeling languages and tools (UML, ER, ORM, for example) are not sufficiently precise for specifications. Additional constraints on the objects in the model must also be defined. While natural languages are easily used by individuals without a programming background, they are often ambiguous. On the other hand, formal programming languages are precise, but not easily used by business analysts and other non-programmers.

The Corticon.js Rule Language has been developed to resolve this dilemma. Based on the Object Constraint Language (OCL, an extension of the Universal Modeling Language specification 1.1), the *Corticon.js Rule Language* (CRL) is designed to enable non-programmers to express rules clearly and precisely without the use of procedural programming languages. More information on OCL may be found at www.uml.org.

Note: A preferred user language might use different separator symbols than those documented for decimal values, list ranges, and dates.

For details, see the following topics:

- Rule structure
- Basic data types
- Truth values
- Collection operators
- Language operators
- Vocabulary used in this Language Guide

Rule structure

In traditional programming languages (or logic systems), most rules are expressed via IF/THEN structures. The IF clause contains a conditional expression and the THEN clause contains actions the rule should perform if all conditions have been met. This IF/THEN structure is expressed as Conditions and Actions in the Rulesheet user interface of Corticon.js Studio. For more information on building and organizing rules in Corticon.js Studio, see the *Corticon.js Studio Tutorial: Basic Rule Modeling*.

Basic data types

The proper expression and execution of rules in Corticon rules is dependent on the type of data involved. Each attribute in the Corticon Vocabulary has a data type, meaning that it has restrictions on the type of data it can contain. Corticon standard data types are as follows:

Data Type	Description
String	Any combination of alphanumeric characters, of any length.
Integer	A whole number, including zero and negative numbers, to the maximum values for a 64-bit long signed integer with 53 significant digits (-9007199254740991 to 9007199254740991).
Decimal	A number containing a decimal point, including zero and negative numbers to the limits of precision in our library. With an arbitrary-precision, the Decimal type for JavaScript calculations are slower than with a plain Number, but they can be executed with an arbitrary higher precision that reduces occurrence of round-off errors. Decimal values in JSON input payloads can be represented either as a number or as string. See below for details and examples.
Boolean	Values are `true` and `false`. `T` and `F` can also be used.
DateTime	DateTime values in a Corticon.js payload must conform to the ISO 8601 standard, or as the number of milliseconds since the epoch. For example, for `9/13/2025 2:00:00 PM EST` use either `1757782800000` or `"2025-09-13T13:00:00-04:00"`. See Formats for date and dateTime in JSON payloads on page 217 and Formats for date and dateTime in Corticon.js Studio tester on page 215 for details on date and dateTime values.
Date	A value with only date information. No time information is allowed.

In this guide, the data types Integer and Decimal are often referred to by the generic term `<Number>`. Wherever `<Number>` is used, either Integer or Decimal data types may be used.

Decimal data in payloads

Decimal data might not fit in a JavaScript number type because JavaScript numbers are a subset of all numbers that can be expressed with the Decimal data type. Therefore, decimals as JavaScript numbers in JSON payload are represented as Strings. For example:

Decimal format in JSON Input Payload

Decimals in a JSON input payload can be represented either as a number or as string. For example:

```
{
 "decimal1" : 123.45,
 "decimal2" : "98765432100000000000000000000000000000.987"}
{
```

Decimal format in JSON Output Payload

Decimals as JavaScript numbers in a JSON output payload are converted to Strings (stringified). For example:

```
{
 "decimal1" : "123.45",
 "decimal2" : "10000000000000000000000000000000.987"}
{
```

Truth values

This guide uses the notation <Expression> to refer to some combination of terms from the Vocabulary that resolves or evaluates to a single “truth value”. A truth value is the Boolean value (`true` or `false`) assigned to an expression upon evaluation by the rule engine. For example, the expression `Patient.name='John'` has a truth value of `true` whenever the patient's name is John. If it is not `John`, then the truth value of this expression is `false`.

Collection operators

Many of the operators provided in the Corticon.js Rule Language deal exclusively with collections of entities. When using collection operators, the expression **must** use aliases to represent the collection(s) operated on by the collection operator(s). A complete discussion of aliases is included in the *Rule Modeling Guide*. Reminders are included throughout this manual wherever collection operators are referenced.

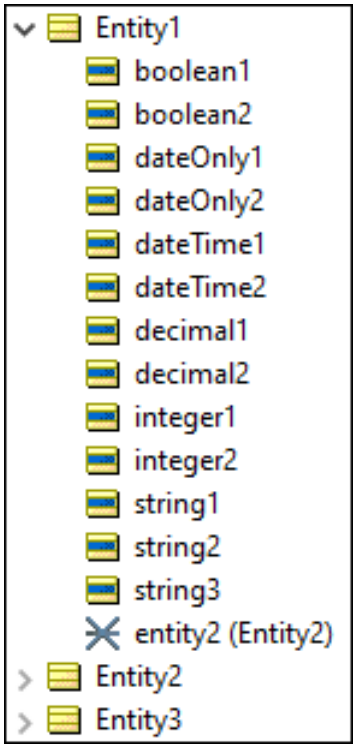
Language operators

The Corticon.js Rule Language operators can be grouped into various classifications as shown in Categories of rule operators. Each operator is subsequently described in detail in the Rule operator details and examples section of this document. That section includes a detailed description of the operator, its syntax, usage restrictions, and an example in a Corticon.js Rulesheet and Ruletest.

Vocabulary used in this Language Guide

This guide uses a generic Vocabulary in all its examples. The Vocabulary contains four entities, each of which contains the same attribute names and types. Attribute names reflect their data types. For example, `integer1` has a data type of Integer. This generic Vocabulary provides sufficient flexibility to create examples using all operators and functions in the Corticon.js Rule Language. `Entity1` is shown expanded in the following figure:

Figure 1: Vocabulary used in Corticon.js Language Guide examples



2

How to access rule operators

The Studio tools for accessing operators provide icons with decorations, and tooltips.

Icons

Rule Operators are assigned icons which provide the user with information about their usage. The following table describes these icons:

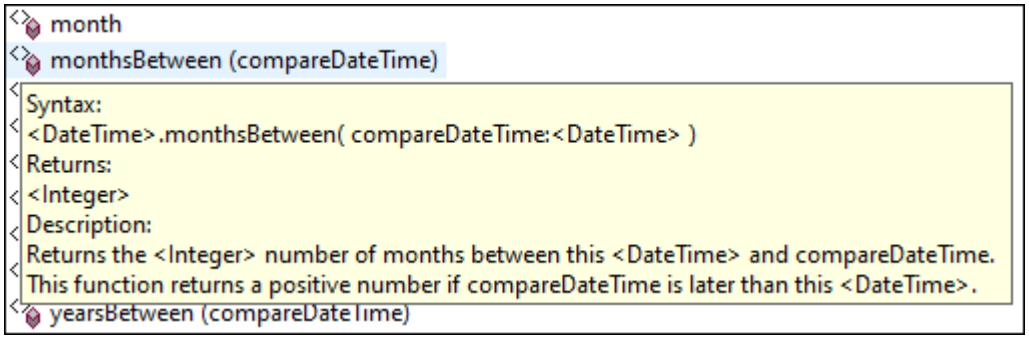
Icon	Where Found	Purpose	Examples
	General, Literals category	indicates special values or constants	null, true, other
	General, Functions category	indicates system values that are automatically retrieved upon rule execution.	now, today
—	Operators, Boolean category	this special “unary” operator icon is used only with not	not

Icon	Where Found	Purpose	Examples
	Operators, all categories	indicates the operator uses a period “.” to attach to its operand. Most operators with this icon typically fell into the previous “function” category.	day, round, contains
	Operators, all categories	indicates the operator is used between two operands. Most operators with this icon typically fell into the previous “comparison” category.	equals, multiply

Tool tips

In Corticon.js Studio, moving the mouse over a Vocabulary operator and pausing, or hovering for a moment, causes a dynamic tool tip text box to display. This tool tip contains information about operator syntax, return data type, and description, all of which are supplied in more detail in this set of topics. For questions not answered by the tool tip, refer to the detailed operator descriptions in this publication. The following figure shows a typical tool tip for the dateTime operator `.monthsBetween`:

Figure 2: Typical Rule Operator Tool Tip



3

Usage restrictions

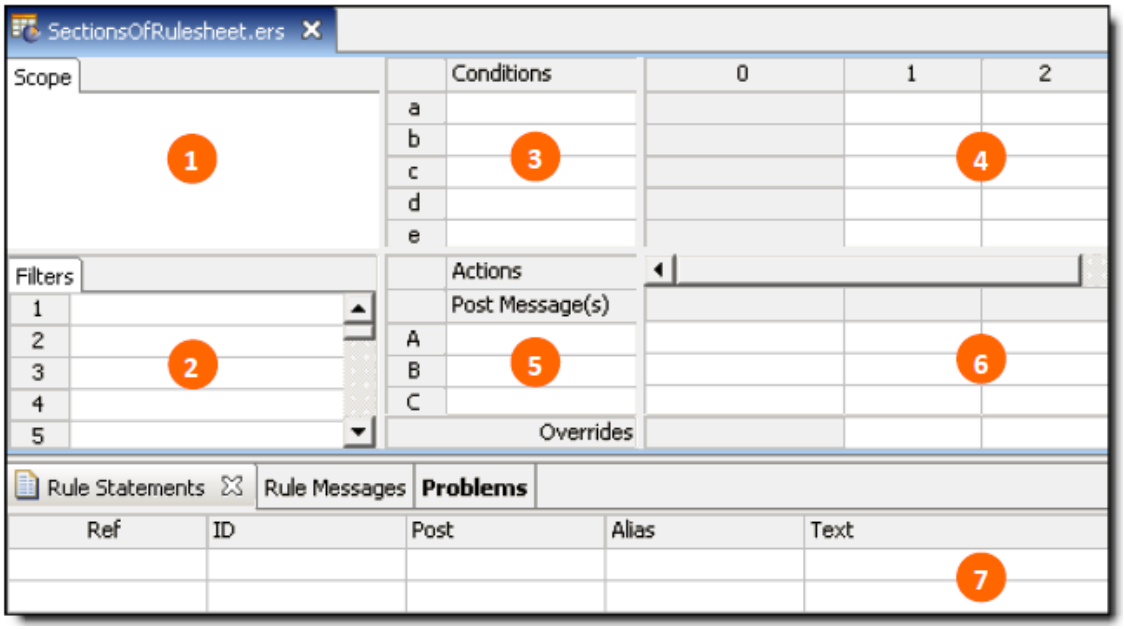
The following illustrations show the general usage restrictions for the various types of Vocabulary terms depending on where they are used in a Rulesheet. This table indicates, for example, that entities (terms from the Vocabulary) may be used in any section of the Rulesheet. Rule Operators, however, are restricted to only three sections.

Note: Some operators have specific restrictions that vary from this general table – see each operator's usage restrictions for details of these exceptions.

Figure 3: Vocabulary usage restrictions in Rulesheet sections

Rulesheet Section Name		Scope	Filter Rows	Condition Rows	Condition Cells	Actions Rows	Action Cells	Rule Statements
Rulesheet Section #		1	2	3	4	5	6	7
Literals			☑	☑	☑	☑	☑	
Functions			☑	☑	☑	☑	☑	☑
Operators			☑	☑		☑		
Data	Values		☑	☑	☑	☑	☑	☑
	Terms	☑	☑	☑	☑	☑	☑	☑

Figure 4: Sections of Rulesheet that correlate with usage restrictions



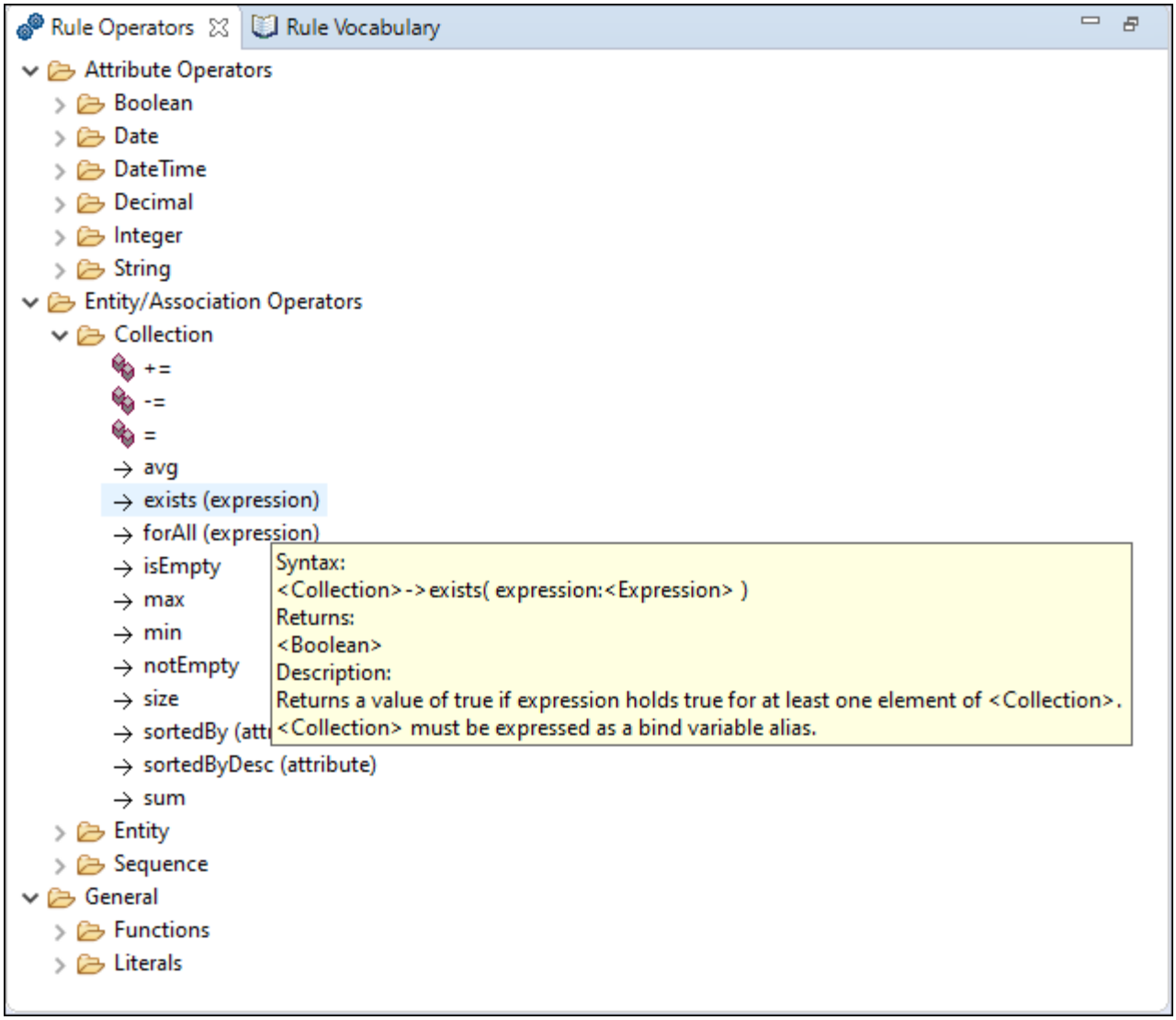
4

Rule operators

Corticon.js Studio presents its rule operators in logical groups.

Rule Operators are classified based on the data type(s) of the terms *to which the operator may be applied* (known as the “operand”). When you open an operator group and hover over an operator, a help window shows its syntax, and details about that operator, as shown:

Figure 5: JS Rule Operator categories



For details, see the following topics:

- Attribute operators
- Entity and Association operators
- General terms

Attribute operators

The Corticon.js Rule Language supports attribute operators categorized as Boolean, DateTime, Decimal, Integer, and String.

Boolean

Corticon.js's **Boolean** attribute operators are as follows:

Name and Syntax	Returns	Description
Equals (used as a comparison)		
<Expression1> = <Expression2>	Boolean	Returns a value of true if <Expression1> has the same value as <Expression2>.
Equals (used as an assignment)		
<Boolean1> = <Expression1>	Boolean	Assigns the truth value of <Expression1> to <Boolean1>
Not Equal To		
<Expression1> <> <Expression2>	Boolean	Returns a value of true if <Expression1> does not have the same truth value as <Expression2>
Or		
<Expression1> or <Expression2> or …	Boolean	Returns a value of true if either <Expression1> or <Expression2> evaluates to true. This operator can be used only in Actions and Preconditions/Filters.
And		
<<Boolean1> and <Boolean2>	Boolean	Returns a value of true if both <<Boolean1> and <Boolean2 are true. This operator can be used only in Actions and Preconditions/Filters.
Not		
not <Expression>	Boolean	Returns the negation of the truth value of <Expression>

Note: See also related information in the topics Precedence of rule operators on page 211 and Standard Boolean constructions on page 203..

Date

Corticon.js's **Date** attribute operators are as follows:

Name and Syntax	Returns	Description
Equals (used as a comparison)		

Name and Syntax	Returns	Description
`<Date1> = <Date2>`	Boolean	Returns a value of true if `<Date1>` is the same as `<Date2>`.
Equals (used as an assignment)		
`<Date1> = <Date2>`	DateTime	Assigns the value of `<Date2>` to `<Date1>`
Not Equal To		
`<Date1> <> <Date2>`	Boolean	Returns a value of true if `<Date1>` does not equal `<Date2>`
Less than		
`<Date1> < <Date2>`	Boolean	Returns a value of true if `<Date1>` is less than `<Date2>`
Greater than		
`<Date1> > <Date2>`	Boolean	Returns a value of true if `<Date1>` is greater than `<Date2>`
Less than or Equal to		
`<Date1> <= <Date2>`	Boolean	Returns a value of true if `<Date1>` is less than or equal to `<Date2>`
Greater than or Equal to		
`<Date1> >= <Date2>`	Boolean	Returns a value of true if `<Date1>` is greater than or equal to `<Date2>`
In (Range)		
`attributeReference in [ |(rangeExpression)|]`	Boolean	Returns a value of true if `attributeReference` is in the range of Date values *`from..to`*, and where opening and closing parentheses `(` `)`indicate exclusion of that limit and square brackets `[ ]` indicate inclusion of that limit.
In (List)		
`attributeReference in {listExpression}`	Boolean	Returns a value of true if `attributeReference` is in the comma-delimited list of literal values, defined enumeration values, or - if in use - enumeration labels.
Year		
`<Date>`.year	Integer	Returns the century/year portion of `<Date>` as a four digit Integer

Name and Syntax	Returns	Description
Month		
<Date>.month	Integer	Returns the month in <Date> as an Integer between 1 and 12
Day		
<Date>.day	Integer	Returns the day portion of <Date> as an Integer between 1 and 31
Add years		
<Date>.addYears(<Integer>)	Date	Adds the number of years in <Integer> to the number of years in <Date>
Add months		
<Date>.addMonth(<Integer>)	Date	Adds the number of months in <Integer> to the number of months in <DateTime>
Add days		
<Date>.addDays(<Integer>)	Date	Adds the number of days in <Integer> to the number of days in <Date>
Years between		
<Date1>.yearsBetween(<Date2>)	Integer	Returns the Integer number of years between <Date1> and <Date2>. This function returns a positive number if <Date2> is later than <Date1>.
Months between		
<Date1>.monthsBetween(<Date2>)	Integer	Returns the Integer number of months between <Date1> and <Date2>. If the month and year portions of <Date1> and <Date2> are the same, the result is zero. This function returns a positive number if <Date2> is later than <Date1>.
Days between		
<Date1>.daysBetween(<Date2>)	Integer	Returns the Integer number of days between <Date1> and <Date2>. If the two dates differ by less than a full 24-hour period, the value is zero. This function returns a positive number if <Date2> is later than <Date1>.
Day of Week		
<Date>.dayOfWeek	Integer	Returns an Integer corresponding to day of the week, with Sunday equal to 1, in <Date>.

Name and Syntax	Returns	Description
Week of Year		
<Date>.weekOfYear	Integer	Returns an Integer from 1 to 52, equal to the week number within the year in <Date>.
Day of Year		
<Date>.dayOfYear	Integer	Returns an Integer from 1 to 366, equal to the day number within the year in <Date>
Week of Month		
<Date>.weekOfMonth	Integer	Returns an Integer from 1 to 6, equal to the week number within the month in <DateTime> or <Date>. A week begins on Sunday and ends on Saturday.
To DateTime		
<Date>.toDateTime	DateTime	Returns a DateTime where the date portion is equal to the value of <Date> and the time portion is equal to 00:00:00 in the system's local timezone.
To DateTime with Timezone Offset		
<Date>.toDateTime (<string>)	DateTime	Returns a DateTime where the date portion is equal to the value of <Date> and the time portion is equal to 00:00:00 in the timezone specified by the value of <string>.

DateTime

Note: A DateTime data type **must contain both** date information **and** time information. Applying a DateTime operator to a DateTime attribute should always produce a result. Be sure to use the data type that suits your needs.

Corticon.js's **DateTime** attribute operators that enable date-only or time-only comparison are as follows:

Name and Syntax	Returns	Description
Is same date		
<DateTime1>.isSameDate (<DateTime2>)	Boolean	Returns a value of true if DateTime1 is the same as DateTime2, ignoring the time part.
After date		
<DateTime1>.afterDate (<DateTime2>)	Boolean	Returns a value of true if DateTime1 is after DateTime2, ignoring the time part.

Name and Syntax	Returns	Description
Before date		
<DateTime1>.beforeDate (<DateTime2>)	Boolean	Returns a value of true if DateTime1 is before DateTime2, ignoring the time part.
Is same time		
<DateTime1>.isSameTime (<DateTime2>)	Boolean	Returns a value of true if DateTime1 is the same as DateTime2, ignoring the date part.
After time		
<DateTime1>.afterTime (<DateTime2>)	Boolean	Returns a value of true if DateTime1 is after DateTime2, ignoring the date part.
Before time		
<DateTime1>.beforeTime(<DateTime2>)	Boolean	Returns a value of true if DateTime1 is before DateTime2, ignoring the date part.

Corticon.js's **DateTime** attribute operators are as follows:

Name and Syntax	Returns	Description
Equals (used as a comparison)		
<DateTime1> = <DateTime2>	Boolean	Returns a value of true if <DateTime1> is the same as <DateTime2>, including both the Date and the Time portions
Equals (used as an assignment)		
<DateTime1> = <DateTime2>	DateTime	Assigns the value of <DateTime2> to <DateTime1>
Not Equal To		
<DateTime1> <> <DateTime2>	Boolean	Returns a value of true if <DateTime1> does not equal <DateTime2>
Less than		
<DateTime1> < <DateTime2>	Boolean	Returns a value of true if <DateTime1> is less than <DateTime2>
Greater than		
<DateTime1> > <DateTime2>	Boolean	Returns a value of true if <DateTime1> is greater than or equal to <DateTime2>
Less than or Equal to		

Name and Syntax	Returns	Description
`<DateTime1> <= <DateTime2>`	Boolean	Returns a value of true if `<DateTime1>` is less than or equal to `<DateTime2>`
Greater than or Equal to		
`<DateTime1> >= <DateTime2>`	Boolean	Returns a value of true if `<DateTime1>` is greater than or equal to `<DateTime2>`
In (Range)		
`attributeReference in [ |(rangeExpression)|]`	Boolean	Returns a value of true if `attributeReference` is in the range of DateTime values *`from..to`*, and where opening and closing parentheses `(` `)`indicate exclusion of that limit and brackets `[ ]` indicate inclusion of that limit.
In (List)		
`attributeReference in {listExpression}`	Boolean	Returns a value of true if `attributeReference` is in the comma-delimited list of literal values, defined enumeration values, or - if in use - enumeration labels.
Year		
`<DateTime>`.year	Integer	Returns the century/year portion of `<DateTime>` as a four digit Integer
Month		
`<DateTime>`.month	Integer	Returns the month in `<DateTime>` as an Integer between 1 and 12
Day		
`<DateTime>`.day	Integer	Returns the day portion of `<DateTime>` as an Integer between 1 and 31
Hour		
`<DateTime>`.hour	Integer	Returns the hour portion of `<DateTime>`. The returned value is based on a 24-hour clock.
Minute		
`<DateTime>`.min	Integer	Returns the minute portion of `<DateTime>` as an Integer between 0 and 59
Second		

Name and Syntax	Returns	Description
<DateTime	Integer	Returns the seconds portion of <>.sec DateTime> as an Integer between 0 and 59
Add years		
<DateTime>.addYears (<Integer>)	DateTime	Adds the number of years in <Integer> to the number of years in <DateTime>
Add months		
<DateTime>>.addMonths (<Integer>)	DateTime	Adds the number of months in <Integer> to the number of months in <DateTime>
Add days		
<DateTime>.addDays (<Integer>)	DateTime	Adds the number of days in <Integer> to the number of days in <DateTime>
Add hours		
<DateTime>.addHours (<Integer>)	DateTime	Adds the number of hours in <Integer> to the number of hours in the Time portion of <DateTime>
Add minutes		
<DateTime>.addMinutes (<Integer>)	DateTime	Adds the number of minutes in <Integer> to the number of minutes in the Time portion of <DateTime>
Add seconds		
<DateTime>.addSeconds (<Integer>)	DateTime	Adds the number of seconds in <Integer> to the number of seconds in the Time portion of <DateTime>
Years between		
<DateTime1>.yearsBetween (<DateTime2>)	Integer	Returns the Integer number of years between <DateTime1> and <Date2>. This function returns a positive number if <DateTime2> is later than <DateTime1>.
Months between		
<DateTime1>.monthsBetween (<DateTime2>)	Integer	Returns the Integer number of months between <DateTime1> and <DateTime2>. If the month and year portions of <DateTime1> and <DateTime2> are the same, the result is zero. This function returns a positive number if <DateTime2> is later than <DateTime1>.

Name and Syntax	Returns	Description
Days between		
<DateTime1>.daysBetween (<DateTime2>)	Integer	Returns the Integer number of days between <DateTime1> and <DateTime2>. If the two dates differ by less than a full 24-hour period, the value is zero. This function returns a positive number if <DateTime2> is later than <DateTime1>.
Hours between		
<DateTime1>.hoursBetween (<DateTime2>)	Integer	Returns the Integer number of hours between <DateTime1> and <DateTime2>. If the two dates differ by less than a full hour, the value is zero. This function returns a positive number if <DateTime2> is later than <DateTime1>.
Minutes between		
<DateTime1>.minsBetween (<DateTime2>)	Integer	Returns the Integer number of minutes between <DateTime1> and <DateTime2>. This function returns a positive number if <DateTime2> is later than <DateTime1>.
Weeks between		
<DateTime1>.weeksBetween (<DateTime2>)	Integer	Returns the Integer number of weeks between <startDate> and <endDate>.
Seconds between		
<DateTime1>.secsBetween (<DateTime2>)	Integer	Returns the Integer number of seconds between <DateTime1> and <DateTime2>. This function returns a positive number if <DateTime2> is later than <DateTime1>.
Day of Week		
<DateTime>.dayOfWeek	Integer	Returns an Integer corresponding to day of the week, with Sunday equal to 1, in <DateTime>.
Week of Year		
<DateTime>.weekOfYear	Integer	Returns an Integer from 1 to 52, equal to the week number within the year in <DateTime>
Day of Year		
<DateTime>.dayOfYear	Integer	Returns an Integer from 1 to 366, equal to the day number within the year in <DateTime>
Week of Month		

Name and Syntax	Returns	Description
<DateTime>.weekOfMonth	Integer	Returns an Integer from 1 to 6, equal to the week number within the month in <DateTime>. A week begins on Sunday and ends on Saturday.
To Date		
<DateTime>.toDate	Date	Returns the date portion only of DateTime
getMilliseconds		
<DateTime>.getMilliseconds	Integer	Returns the internal date/time, namely the number of milliseconds that have transpired since the epoch, 1/1/1970 00:00:00 GMT.

Decimal

In this section, wherever the syntax includes <Number>, either Integer or Decimal data types may be used.

Corticon.js's **Decimal** attribute operators are as follows:

Name and Syntax	Returns	Description
Equals (used as a comparison)		
<Number1> = <Number2>	Boolean	Returns a value of true if <Number1> is the same as <Number2>.
Equals (used as an assignment)		
<Number1> = <Number2>	Number	Assigns the value of <Number2> to the value of <Number1>.
Not Equal To		
<Number1> <> <Number2>	Boolean	Returns a value of true if <Number1> is not equal to <Number2>.
Less than		
<Number1> < <Number2>	Boolean	Returns a value of true if <Number1> is less than <Number2>.
Greater than		
<Number1> > <Number2>	Boolean	Returns a value of true if <Number1> is greater than <Number2>.
Less than or Equal to		

Name and Syntax	Returns	Description
`<Number1> <= <Number2>`	Boolean	Returns a value of true if `<Number1>` is less than or equal to `<Number2>`.
Greater than or Equal to		
`<Number1> >= <Number2>`	Boolean	Returns a value of true if `<Number1>` is greater than or equal to `<Number2>`.
In (Range)		
`attributeReference in [ |(rangeExpression)|]`	Boolean	Returns a value of true if `attributeReference` is in the range of Decimal values *`from..to`*, and where opening and closing parentheses `( )` indicate exclusion of that limit and square brackets `[ ]` indicate inclusion of that limit.
In (List)		
`attributeReference in {listExpression}`	Boolean	Returns a value of true if `attributeReference` is in the comma-delimited list of literal values, defined enumeration values, or - if in use - enumeration labels.
Add		
`<Number1> + <Number2>`	Number	Returns the sum of `<Number1>` and `<Number2>`. The resulting data type is the more expansive of either `<Number1>` or `<Number2>`. For example, if an Integer value is added to a Decimal value, the resulting value will be a Decimal. See Precedence of rule operators on page 211.
Subtract		
`<Number1> - <Number2>`	Number	Subtracts `<Number2>` from `<Number1>`. The resulting data type is the more expansive of either `<Number1>` or `<Number2>`. See Precedence of rule operators on page 211.
Multiply		
`<Number1> * <Number2>`	Number	Returns the product of `<Number1>` and `<Number2>`. The resulting data type is the more expansive of either `<Number1>` or `<Number2>`. See Precedence of rule operators on page 211.
Divide		

Name and Syntax	Returns	Description
<Number1> / <Number2>	Number	Divides <Number1> by <Number2>. The resulting data type is the more expansive of either <Number1> or <Number2>. See Precedence of rule operators on page 211.
Exponent		
<Number1> ** <Number2>	Number	Raises <Number1> to the power of <Number2>. The resulting data type is the more expansive of either <Number1> or <Number2>. See Precedence of rule operators on page 211.
Absolute Value		
<Decimal>.absVal	Decimal	Returns the absolute value of <Number>. If the <Number> is positive, <Number> itself is returned; if <Number> is negative, the negation of <Number> is returned.
Floor		
<Decimal>.floor	Integer	Returns a new Decimal whose value is the value of this Decimal rounded to a whole number in the direction of negative infinity.
Ceiling		
<Decimal>.ceiling	Integer	Returns a new Decimal whose value is the value of this Decimal rounded to a whole number in the direction of positive infinity.
Round		
<Decimal>.round	Decimal	Rounds <Decimal> to the nearest Integer.
Round(n)		
<Decimal>.round(<decimalDigits<Integer>>)	Decimal	Returns <Decimal> rounded to the number of decimal places specified by decimalDigits. Rounds toward the nearest neighbor; where equivalent, rounds toward infinity.
Round(n)		
<Decimal>.round(<decimalDigits<Integer>>,mode:<integer>)	Decimal	Returns <Decimal> rounded to the number of decimal places specified by decimalDigits. See details page for information about mode.
Maximum Value		
<Decimal>.max(<Number>)	Number	Returns the greater of <Decimal> and <Number>.

Name and Syntax	Returns	Description
Minimum Value		
<Decimal>.min(<Number>)	Number	Returns the lesser of <Decimal> and <Number>.
Logarithm (base 10)		
<Decimal>.log	Decimal	Returns the logarithm (base 10) of <Decimal>. <Decimal> may not be zero.
Logarithm (base x)		
<Decimal1>.log(<Decimal2>)	Decimal	Returns the logarithm (base <Decimal2>) of <Decimal1>. <Decimal1> may not be zero.
Natural Logarithm		
<Decimal>.ln	Decimal	Returns the logarithm (base e) of <Decimal>.<Decimal> may not be zero.
Random		
<Decimal>.random(minRange, maxRange)	Decimal	Returns a random decimal between minRange and maxRange.
Truncate		
<Decimal>.truncate	Integer	Truncates "this" Decimal value to an integer by removing the fractional portion.

Integer

In this section, wherever the syntax includes <Number>, either Integer or Decimal data types may be used.

Corticon.js's **Integer** attribute operators are as follows:

Name and Syntax	Returns	Description
Equals (used as a comparison)		
<Number1> = <Number2>	Boolean	Returns a value of true if <Number1> is the same as <Number2>.
Equals (used as an assignment)		
<Number1> = <Number2>	Number	Assigns the value of <Number2> to the value of <Number1>. The data type of <Number1> must be expansive enough to accommodate <Number2>.
Not Equal To		

Name and Syntax	Returns	Description
`<Number1> <> <Number2>`	Boolean	Returns a value of true if `<Number1>` is not equal to `<Number2>`.
Less than		
`<Number1> < <Number2>`	Boolean	Returns a value of true if `<Number1>` is less than `<Number2>`.
Greater than		
`<Number1> > <Number2>`	Boolean	Returns a value of true if `<Number1>` is greater than `<Number2>`.
Less than or Equal to		
`<Number1> <= <Number2>`	Boolean	Returns a value of true if `<Number1>` is less than or equal to `<Number2>`.
Greater than or Equal to		
`<Number1> >= <Number2>`	Boolean	Returns a value of true if `<Number1>` is greater than or equal to `<Number2>`.
In (Range)		
`attributeReference in [ |(rangeExpression)|]`	Boolean	Returns a value of true if `attributeReference` is in the range of Integer values *`from..to`*, and where opening and closing parentheses `(` `)`indicate exclusion of that limit and square brackets `[ ]` indicate inclusion of that limit.
In (List)		
`attributeReference in {listExpression}`	Boolean	Returns a value of true if `attributeReference` is in the comma-delimited list of literal values, defined enumeration values, or - if in use - enumeration labels.
Add		
`<Number1> + <Number2>`	Number	Returns the sum of `<Number1>` and `<Number2>`. The resulting data type is the more expansive of either `<Number1>` or `<Number2>`. For example, if an Integer value is added to a Decimal value, the resulting value will be a Decimal. See Precedence of rule operators on page 211.
Subtract		

Name and Syntax	Returns	Description
<Number1> - <Number2>	Number	Subtracts <Number2> from <Number1>. The resulting data type is the more expansive of either <Number1> or <Number2>. See Precedence of rule operators on page 211.
Multiply		
<Number1> * <Number2>	Number	Returns the product of <Number1> and <Number2>. The resulting data type is the more expansive of either <Number1> or <Number2>. See Precedence of rule operators on page 211.
Increment		
<Number1> += <Number2>	Number	Increments <Number1> by <Number2>. The data type of <Number1> must accommodate the addition of <Number2>. See Precedence of rule operators on page 211.
Decrement		
<Number1> -= <Number2>	Number	Decrements <Number1> by the value of <Number2>. The data type of <Number1> must accommodate the addition of <Number2>. See Precedence of rule operators on page 211.
Absolute value on page 47.		
<Integer>.absVal	Number	Returns the absolute value of <Integer>. If the <Integer> is positive, <Integer> itself is returned; if <Integer> is negative, the negation of <Integer> is returned.
To Decimal		
<Integer>.toDecimal	Decimal	Converts an attribute of type Integer to type Decimal.
To String		
<Integer>.toString	String	Converts an attribute of type Integer to type String.
Maximum Value		
<Integer1>.max(<Integer2>)	Integer	Returns the greater of <Integer1> and <Integer2>.
Minimum Value		

Name and Syntax	Returns	Description
<Integer1>.min(<Integer2>)	Integer	Returns the lesser of <Integer1> and <Integer2>.
Random		
<Decimal>.random(minRange, maxRange)	Decimal	Returns a random integer between minRange and maxRange.
Mod		
<Integer1>.mod(<Integer2>)	Integer	Returns the whole number remainder that results from dividing <Integer1> by <Integer2>. If the remainder is a fraction, then zero is returned.
Logarithm (base 10)		
<Integer>.log	Decimal	Returns the logarithm (base 10) of <Integer>. <Integer> may not be zero.
Logarithm (base x)		
<Integer>.log(<Decimal>)	Decimal	Returns the logarithm (base <Decimal>) of <Integer>. <Integer> may not be zero.
Natural Logarithm		
<Integer>.ln	Decimal	Returns the natural logarithm (base e) of <Number>. <Integer> may not be zero.

String

Corticon.js's **String** attribute operators are as follows:

Name and Syntax	Returns	Description
Equals (used as a comparison)		
<String1> = <String2>	Boolean	Returns a value of true if <String1> exactly matches <String2>. Both case and length are examined to determine equality.
Equals (used as an assignment)		
<String1> = <String2>	String	Assigns the value of <String2> to the value of <String1>.
Not Equal to		

Name and Syntax	Returns	Description
<String1> <> <String2>	Boolean	Returns a value of true if <String1> is not equal to <String2>.
Less than		
<String1> < <String2>	Boolean	Returns a value of true if <String1> is less than <String2>.
Greater than on page 100		
<String1> > <String2>	Boolean	Returns a value of true if <String1> is greater than <String2>.
Less than or Equal to		
<String1> <= <String2>	Boolean	Returns a value of true if <String1> is less than or equal to <String2>.
Greater than or Equal to		
<String1> >= <String2>	Boolean	Returns a value of true if <String1> is greater than or equal to <String2>.
In (Range)		
attributeReference in [|(rangeExpression)|]	Boolean	Returns a value of true if attributeReference is in the range of String values *from..to*, and where opening and closing parentheses ( )indicate exclusion of that limit and square brackets [ ] indicate inclusion of that limit.
In (List)		
attributeReference in {listExpression}	Boolean	Returns a value of true if attributeReference is in the comma-delimited list of literal values, defined enumeration values, or - if in use - enumeration labels.
Adding Strings		
<String1> + <String2>	String	Concatenates <String1> to <String2>. Alternative syntax.
Size		
<String>.size	String	Returns the number of characters in <String>.
Concatenate		
<String1>.concat(<String2>)	String	Concatenates <String1> to <String2>.

Name and Syntax	Returns	Description
Uppercase		
<String>.toUpper	String	Converts all characters <String> to uppercase.
Lowercase		
<String>.toLower	String	Converts all characters in <String> to lowercase.
To DateTime		
<String>.toDateTime	DateTime	Converts the value in <String> to data type DateTime ONLY if date as ISO8601 string or millisecs since epoch as a string.
To Decimal		
<String>.toDecimal	Decimal	Converts an attribute of type String to data type Decimal ONLY if all characters in <String> are numeric and contain not more than one decimal point. If any non-numeric characters are present (other than a single decimal point or leading minus sign), no value is returned.
To Integer		
<String>.toInteger	Integer	Converts an attribute of type String to type Integer ONLY if all characters in <String> are numeric. If any non-numeric characters are present, no value is returned.
Substring		
<String>.substring (<Integer1>,<Integer2>)	String	Returns that portion of <String> between character positions <Integer1> and Integer2>.
Equals Ignoring Case		
<String1>.equalsIgnoreCase (<String2>)	Boolean	Returns a value of true if <String1> is the same as <String2>, irrespective of case.
Starts with		
<String1>.startsWith (<String2>)	Boolean	Returns a value of true if the <String1> begins with the characters specified in <String2>.
Ends with		
<String1>.endsWith (<String2>)	Boolean	Evaluates the contents of <String1> and returns a value of true if the String ends with the characters specified in <String2>.

Name and Syntax	Returns	Description
Contains		
<String1>.contains (<String2>)	Boolean	Evaluates the contents of <String1> and returns a value of true if it contains the exact characters defined by <String2>
Equals		
<String1>.equals (<String2>)	Boolean	Returns a value of true if <String1> is the same as <String2>.
Index Of		
<String1>.indexOf (<String2)	Integer	Returns the beginning character position number of <String2> within <String1>, if <String1> contains <String2>. If it does not, the function returns a value of zero.
Replace String		
<String>.replaceString(stringToBeReplaced,replacementString)	String	Returns a new String where the instances of the String to be replaced are replaced by the value of the replacement String.
Regular expression replace String		
<String>.regexReplaceString(regularExpression,replacementString)	String	Returns a new String where the Strings matching the regular expression are replaced by the replacement String.
Matches		
<String>.matches(regularExpression:String)	Boolean	Returns true if the regular expression matches the String.
characterAt(index)		
<String>.characterAt(index:Integer)	String	Returns the character at the specified position in the String.
isInteger		

Name and Syntax	Returns	Description
`<String>.isInteger`	Boolean	Determines whether "this" String contains only integer digits. **Note:** This operator examines each character in a string to determine whether it is in the range 0 to 9. Therefore, the operator returns `true` when the entire string evaluates as a positive integer, and `false` when a minus sign is the first character of a string that would evaluate as a negative integer. A new extended operator could be created if the string as a whole is to be evaluated as `true` whether positive or negative (for example, by allowing the first character to be a minus sign.)
trimSpaces		
`<String>.trimSpaces`	String	Trims leading and trailing spaces from "this" String.

Entity and Association operators

The Corticon.js rule language supports Entity, Association, and Sequence operators.

Collection

Corticon.js's **Collection** operators are as follows:

Name and Syntax	Returns	Description
Replace element(s)		
`<Collection1> = <Collection2>` `<Collection1> = <Entity>`	*modifies a collection*	replaces all elements in `<Collection1>` with elements of `<Collection2>` or with `<Entity>`, provided the new associations are allowed by the Business Vocabulary.
Associate element(s)		
`<Collection1> += <Collection2>` `<Collection1> += <Entity>`	*modifies a collection*	Associates all elements of `<Collection2>` or `<Entity>` with `<Collection1>`. Every `<Collection>` must be expressed as a unique alias.
Disassociate element(s)		

Name and Syntax	Returns	Description
<Collection1> -= <Collection2>	*modifies a collection*	Disassociates all elements of <Collection2> from <Collection1>. Does not delete the disassociated elements. Every <Collection> must be expressed as a unique alias.
Is empty		
<Collection> ->isEmpty	Boolean	Returns a value of true if <Collection> contains *no* elements
Not empty		
<Collection> ->notEmpty	Boolean	Returns a value of true if <Collection> contains *at least one* element.
Exists		
<Collection> ->exists (<Expression>)	Boolean	Returns a value of true if <Expression> holds true for *at least one* element of <Collection>
For all		
<Collection> ->forAll (<Expression>)	Boolean	Returns a value of true if *every* <Expression> holds true for *every* element of <Collection>
Sorted by		
<Collection> ->sortedBy (<Attribute>)	*converts a collection into a sequence*	Sequences the elements of <Collection> in ascending order, using the value of <Attribute> as the index. <Collection> must be expressed as a unique alias.
Sorted by descending		
<Collection> ->sortedByDesc (<Attribute>)	*converts a collection into a sequence*	Sequences the elements of <Collection> in descending order, using the value of <Attribute> as the index. <Collection> must be expressed as a unique alias.
Size of collection		
<Collection> ->size	Integer	Returns the number of elements in <Collection>. <Collection> must be expressed as a unique alias.
Sum		
<Collection.attribute> ->sum	Number	Sums the values of the specified <attribute> for all elements in <Collection>. <attribute> must be a numeric data type.

Name and Syntax	Returns	Description
Average		
<Collection.attribute> ->avg	Number	Averages all of the specified attributes in <Collection>. <Collection> must be expressed as a unique alias. <attribute> must be a numeric data type
Minimum		
<Collection.attribute> ->min	Number	Returns the lowest value of <attribute> for all elements in <Collection>. <attribute> must be a numeric data type
Maximum		
<Collection.attribute> ->max	Number	Returns the highest value of <attribute> for all elements in <Collection>. <attribute> must be a numeric data type

Entity

Corticon.js's **Entity** operators are as follows:

Name and Syntax	Returns	Description
New		
<Entity> .new [<Expression1>,...]	Entity	Creates a new instance of <Entity>. Expressions (optional to assign attribute values) in square brackets [..] must be written in the form: *attribute = value*.
New Unique		
<Entity> .newUnique [<Expression1>,...]	Entity	Creates a new instance of <Entity> only if the instance created is unique as defined by optional <Expression1>,...
Clone		
<Entity>.clone [<Expression1>,...]	Entity	Creates a new instance of <Entity> with the same attributes and their respective values. Expressions (optional to override attribute values) in square brackets [..] must be written in the form: *attribute = value*.

Name and Syntax	Returns	Description
Remove		
< Entity>.remove [(true)|(false)]	Entity	Deletes the entity from memory and from the resultant JSON document. Children can be removed as well when set to (true, or retained after moving to root (false). Blank or no value defaults to true.

Sequence

Sequence operators act on collections that have *already* been ordered by a sorting operator (see sortedBy and sortedByDesc). In other words, sequence operators operate on collections that have been turned into sequences. The notation <Sequence> used below, is shorthand for a completed sorting operation. For example:

<Collection> -> sortedBy(<Attribute>)

produces a <Sequence>, in this case the elements of <Collection> arranged in ascending order using <Attribute> as the index. This <Sequence> can then be used with one of the sequence operators described below. The design of the Object Constraint Language (upon which the Corticon.js Rule Language is based), allows for the “chaining” of operators, so a collection operator and a sequence operator can be used in the same expression to produce a sequence and identify a particular element of that sequence in the same step. For example:

<Entity.attribute1> = <Collection> ->sortedBy(<Attribute3>) ->first.<Attribute2>

performs the following:

1. Sorts <Collection> in ascending order according to <Attribute3>, turning it into a <Sequence>
2. Locates the first element of <Sequence>
3. Reads the value of <Attribute2> of the first element
4. Assigns the value of <Attribute2> of the first element to <Entity.attribute1>

Corticon.js's **Sequence** operators are as follows:

	Name and Syntax	Returns	Description
At			
	<Sequence> ->at(<Integer>)	Entity	Returns the element at position <Integer>. <Sequence> must be expressed as a unique alias.
First			
	<Sequence> ->first	Entity	Returns the first element of <Sequence>. <Sequence> must be expressed as a unique alias.

	Name and Syntax	Returns	Description
Last			
	<Sequence> ->last	Entity	Returns the last element of <Sequence>. <Sequence> must be expressed as a unique alias.

General terms

Corticon.js's **General** operators are **Functions** and **Literals**.

Functions

Corticon.js's **Functions** operators are as follows:

Name and Syntax	Returns	Description
Custom operators with return datatype		
`getBoolean getDate getDateTime getDecimal getInteger getString`	Corresponding datatype	Calls the specified custom function with the custom string parameter. Returns the data specified in the custom function as the requested datatype. See *"Customized data access operators" in the Corticon.js Integration Guide* for more information.
Custom operators with no return		
`setBoolean setDate setDateTime setDecimal setInteger setString`	nothing	Call the specified function with t he custom parameters. See *"Customized data access operators" in the Corticon.js Integration Guide* for more information.
Now		
`now`	DateTime	Returns the current system date and time when the rule is executed.
Today		
`today`	Date	Returns the current system date when the rule is executed.

Literals

Literal Terms can be used in any section of the Rulesheet, except **Scope** and **Rule Statements**. Exceptions to this general statement exist. See individual literals for detailed usage restrictions.

Corticon.js's **Literals** operators are as follows:

Name and Syntax	Returns	Description
Null		
`null`	*none*	The null value corresponds to one of three different scenarios: • the absence of an attribute in a Ruletest scenario • the absence of data for an attribute in a Ruletest scenario • an object that has a value of null
True		
`true` or `T`	Boolean	Represents Boolean value true
False		
`false` or `F`	Boolean	Represents the Boolean value false
Other		
`other`	*any*	When included in a condition's Values set, other represents any value not explicitly included in the set, including null.
CellValue		
`cellValue`	*any*	cellValue is a variable whose value is determined by the rule Column that executes

5

Rule operator details and examples

The following pages describe each operator in greater detail. Each Rule Operator has the following sections

1. **Syntax** – Describes the standard syntax used with this operator. In this section, as in the previous summary tables, the angle bracket convention <. .> is used to indicate what types of terms and their data types can be used with the operator. When using the operator with real terms from the Vocabulary, do not include the angle brackets.
2. **Description** – Provides a plain-language description of the operator's purpose and details of its use. Important reminders, tips, or cautions are included in this section.
3. **Usage Restrictions** – Describes what limitations exist for this operator, and where an operator may not be used in a Rulesheet. Such limitations are rare, but important to a good understanding of Corticon.js Studio.
4. **Example** – Shows an example of each operator in a Rulesheet. A screenshot of the example Rulesheet is provided, with portions of the Rulesheet not used by the example collapsed or truncated for clarity. The example also includes sample input and output data for Ruletest scenarios run against the Rulesheet.

The entire list of operators is presented in alphabetic order.

For details, see the following topics:

- Add months
- Add seconds
- Add years
- After date
- After time
- Associate elements
- At
- Average
- Before date
- Before time
- Ceiling
- CellValue
- Character at
- Clone
- Concatenate
- Contains
- Day
- Day of week
- Day of year
- Days between
- Decrement
- Disassociate elements
- Divide
- Ends with
- Equals ignoring case
- Equals when used as an assignment
- Equals when used as a comparison
- Equals when using Strings
- Exists
- Exponent
- False
- First
- Floor

- For all
- Get Milliseconds
- Greater than
- Greater than or equal to
- Hour
- Hours between
- In LIST
- In RANGE
- Increment
- Index of
- Is empty
- Is integer
- Last
- Is same date
- Is same time
- Less than
- Less than or equal to
- Logarithm BASE 10
- Logarithm BASE X
- Lowercase
- Matches
- Maximum value
- Maximum value COLLECTION
- Minimum value
- Minimum value COLLECTION
- Minute
- Minutes between
- Mod
- Month
- Months between
- Multiply
- Natural logarithm
- New

- New unique
- Not
- Not empty
- Not equal to
- Now
- Null
- Other
- Or
- Random
- Regular expression to replace String
- Remove element
- Replace elements
- Replace String
- Round
- Second
- Seconds between
- Size of collection
- Size of string
- Sorted by
- Sorted by descending
- Starts with
- Substring
- Subtract
- Sum
- Today
- To dateTime
- To date Casting a dateTime to a date
- To dateTime Casting a string to a dateTime
- To dateTime Casting a date to a dateTime
- To dateTime Timezone offset
- To decimal
- To integer
- To string

- Trim spaces
- True
- Truncate
- Uppercase
- Week of month
- Week of year
- Weeks between
- Year
- Years between

Absolute value

SYNTAX

`<Number>.absVal`

DESCRIPTION

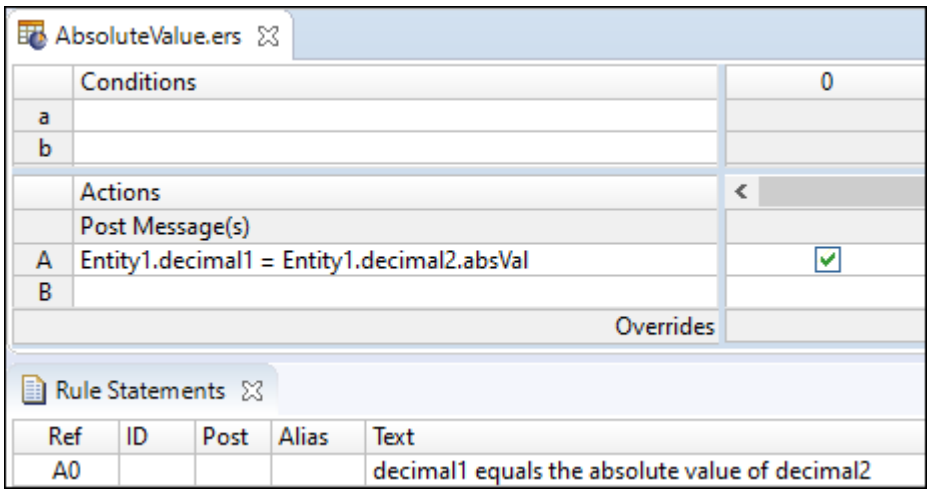
Returns the absolute value of `<Number>`. If the `<Number>` is positive, `<Number>` itself is returned; if `<Number>` is negative, the negation of `<Number>` is returned.

USAGE RESTRICTIONS

The Operators row in the table of Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

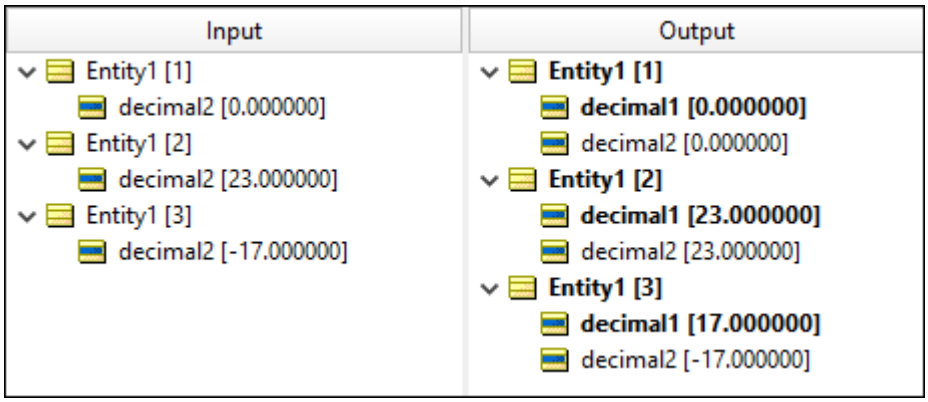
RULESHEET EXAMPLE

This sample Rulesheet uses **.absVal** to produce the absolute value of `decimal2` and assign it to `decimal1`



SAMPLE RULETEST

A sample Ruletest provides `decimal2` values for three different scenarios of `Entity1`. Input and Output panels are shown below.



Add numbers

SYNTAX

`<Number1>` + `<Number2>`

DESCRIPTION

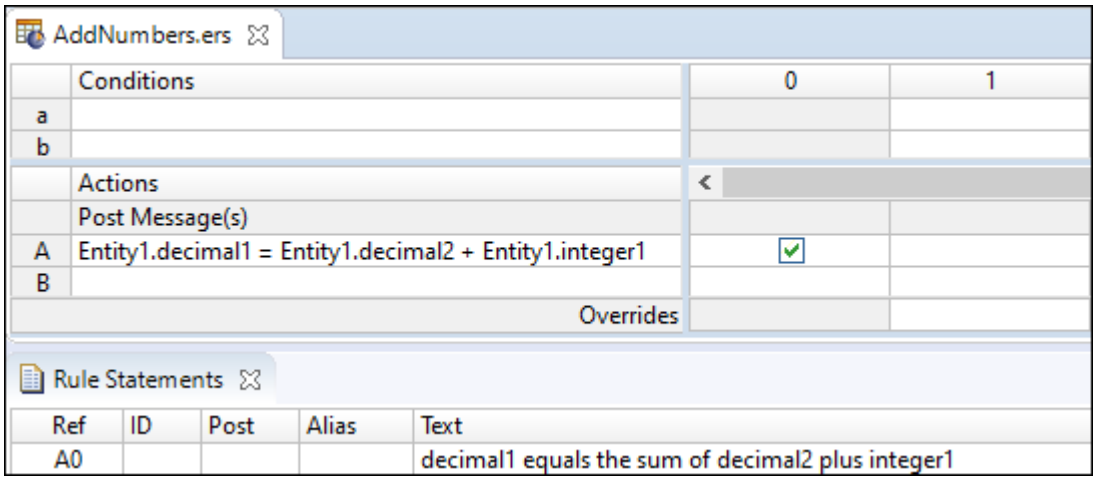
Adds `<Number1>` to `<Number2>`. The resulting data type is the more expansive of those of `<Number1>` and `<Number2>`. For example, if you are adding an Integer value and a Decimal value, the resulting value will be a Decimal. See Precedence of rule operators on page 211.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

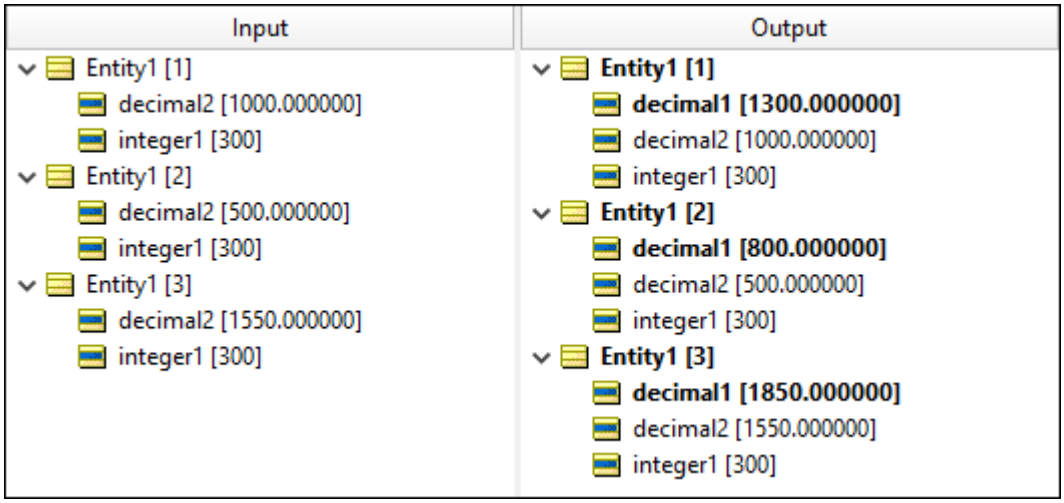
RULESHEET EXAMPLE

This sample Rulesheet uses the **add numbers** operation to add the value of `decimal2` to the value of `integer1` and assign the result to `decimal1`



SAMPLE RULETEST

A sample Ruletest provides an `integer1` value of `300` which is added to the value of `decimal2` and assigned to the value of `decimal1` for three instances of `Entity1`. Input and Output panels are shown below.



Add strings

SYNTAX

`<String1>` + `<String2>`

DESCRIPTION

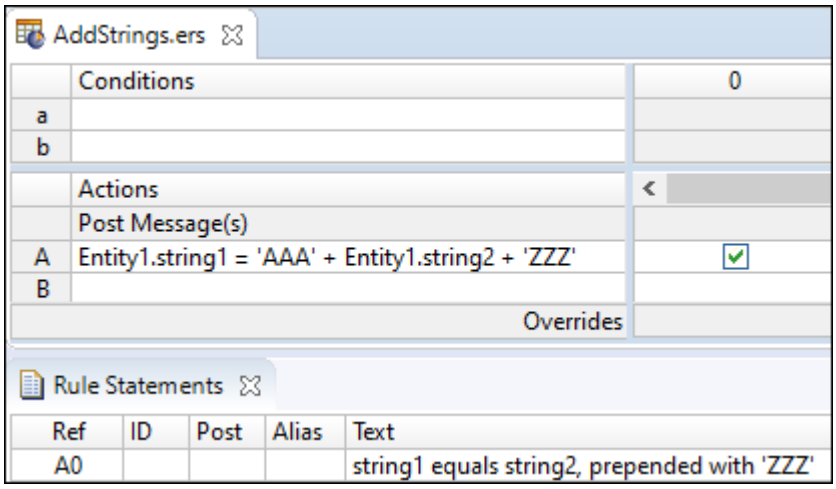
Adds `<String1>` to `<String2>`. This has the same effect as using the .concat operator. However, the “+” syntax permits concatenation of more than two String values without nesting, as shown in the example below.

USAGE RESTRICTIONS

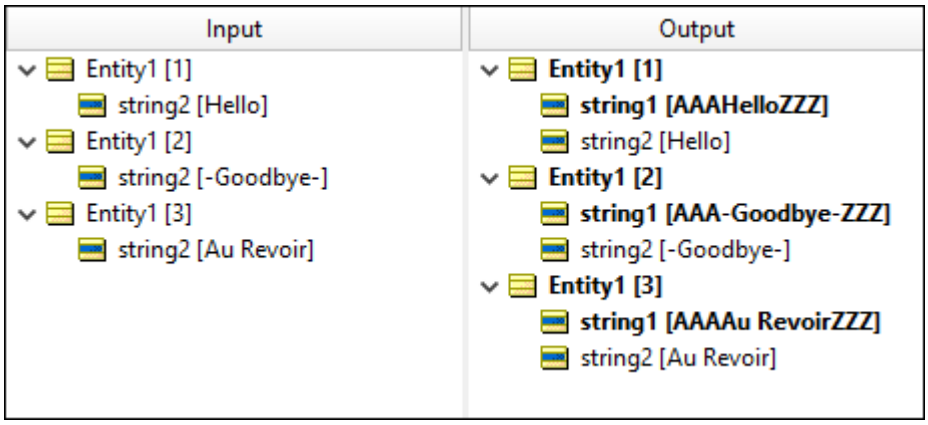
The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

This sample Rulesheet uses **add strings** operation to add the String AAA to `string2` to ZZZ and assign the result to `string1`



SAMPLE RULETEST



Add days

SYNTAX

`<DateTime>`.addDays(`<Integer>`)

`<Date>`.addDays(`<Integer>`)

DESCRIPTION

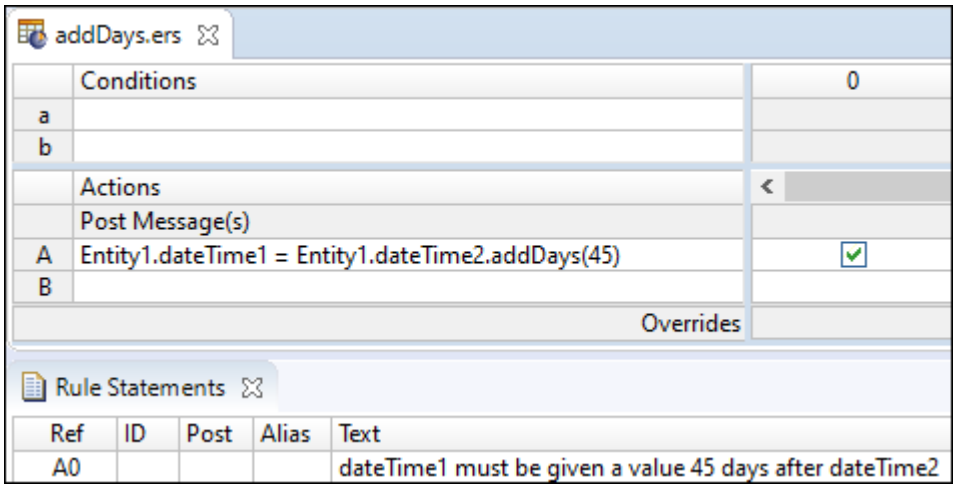
Adds the number of days in `<Integer>` to the number of days in `<DateTime>` or `<Date>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

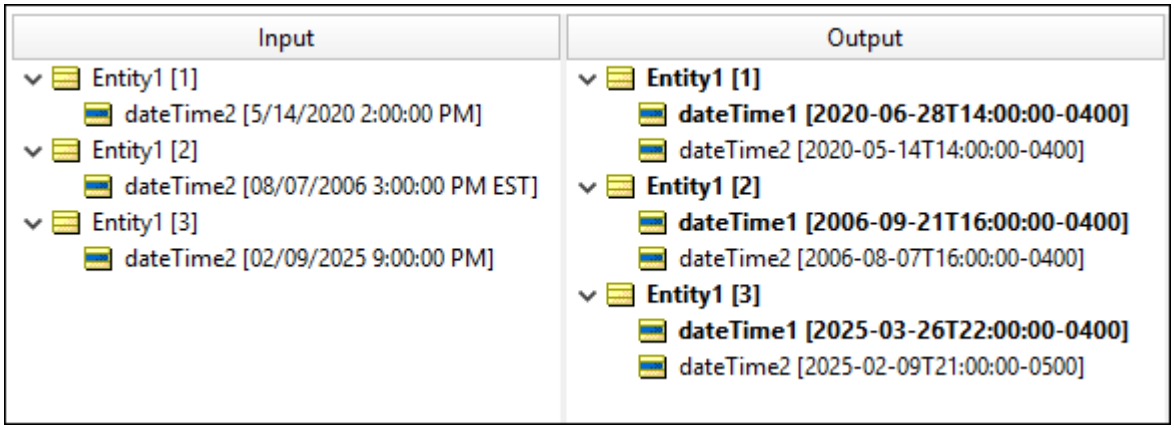
RULESHEET EXAMPLE

This sample Rulesheet uses **.addDays** to add 45 days to the value of `dateTime2` and assign the result to `dateTime1`.



SAMPLE RULETEST

A sample Ruletest provides values of `dateTime2` for three instances of `Entity1`. Input and Output panels are shown below. Notice the month portion of `dateTime1` also changes accordingly.



Add hours

SYNTAX

`<DateTime>.addHours(<Integer>)`

DESCRIPTION

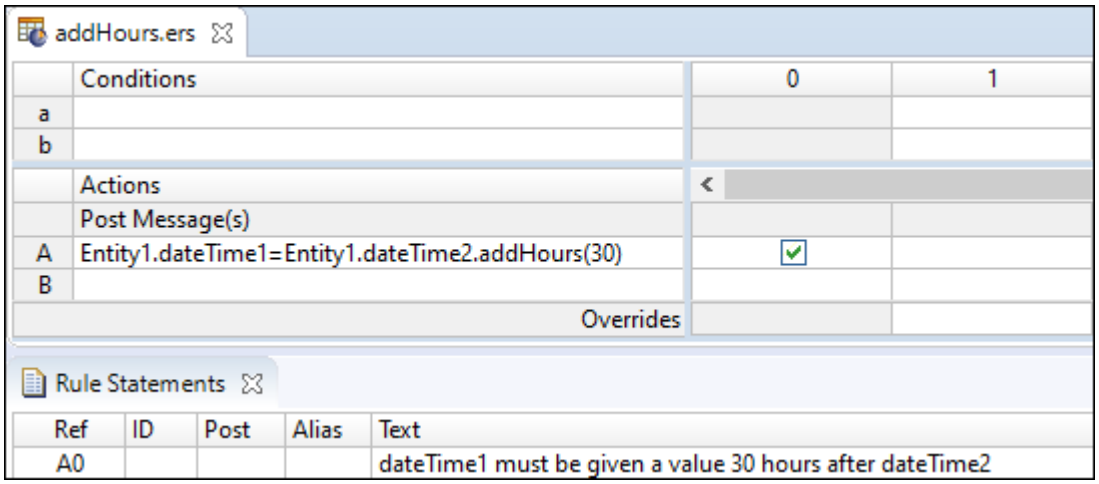
Adds the number of hours in `<Integer>` to the number of hours in the Time portion of `<DateTime`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

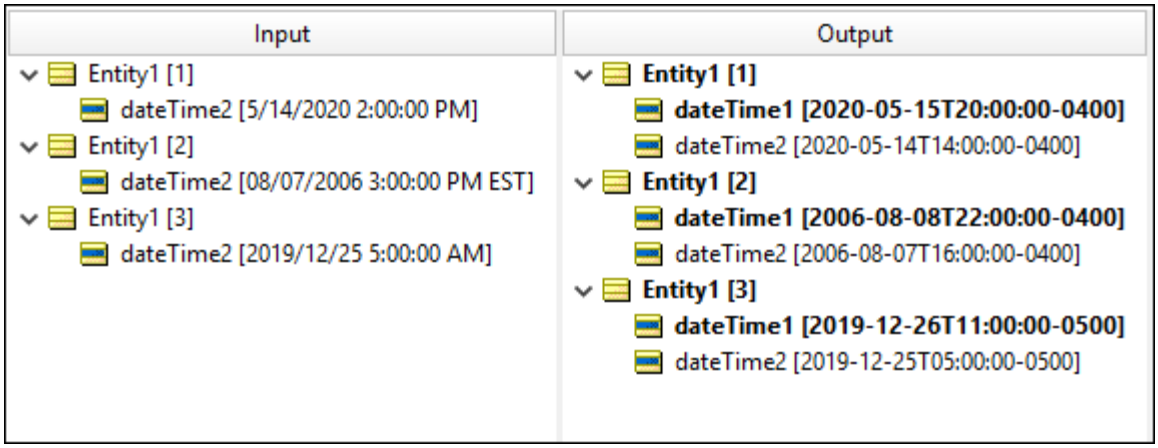
RULESHEET EXAMPLE

This sample Rulesheet uses the **. addHours** to add `30` hours to the value of `dateTime2` and assign the result to `dateTime1`.



SAMPLE RULETEST

A sample Ruletest provides values of `dateTime2` for three instances of `Entity1`. Input and Output panels are shown below.



Add minutes

SYNTAX

`<DateTime>.addMinutes(<Integer>)`

DESCRIPTION

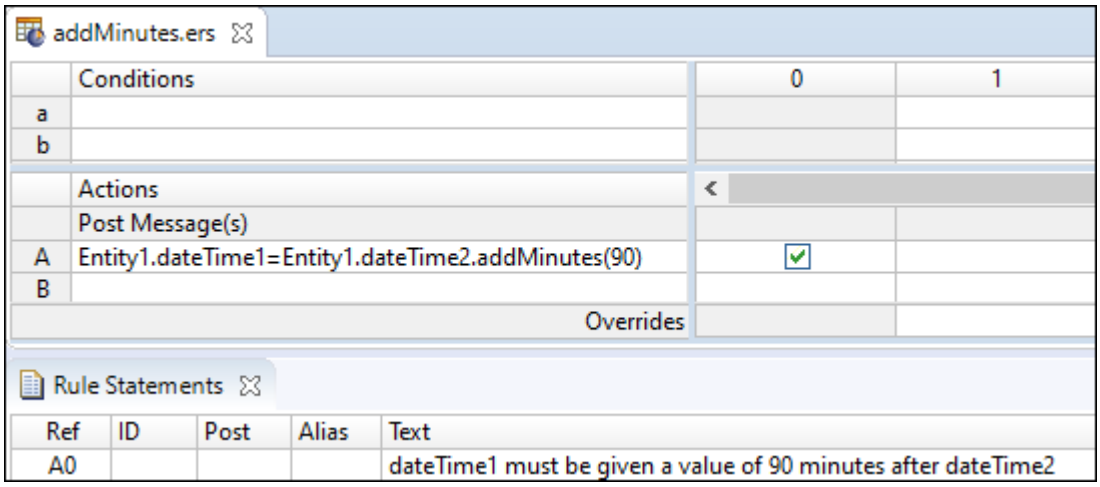
Adds the number of minutes in `<Integer>` to the number of minutes in the Time portion of `<DateTime>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

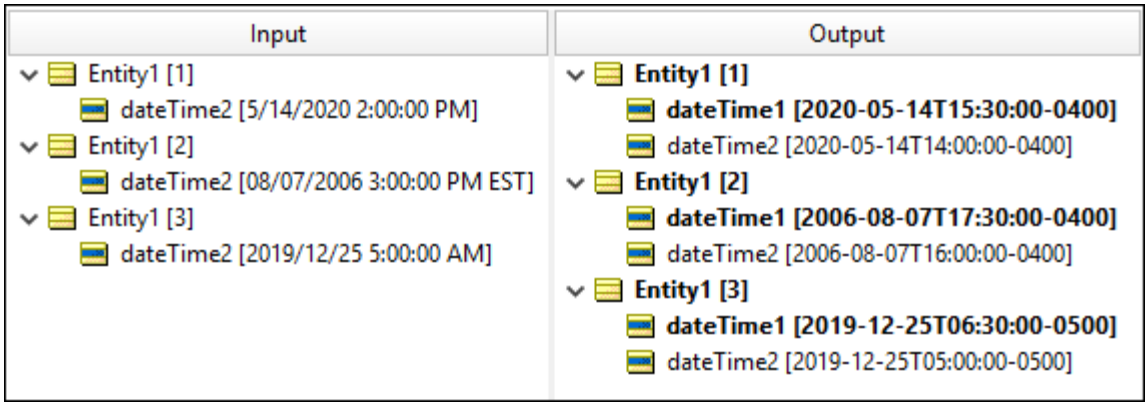
RULESHEET EXAMPLE

This sample Rulesheet uses the **. addMinutes** add `90` minutes to the value of `dateTime2` and assign the result to `dateTime1`.



SAMPLE RULETEST

A sample Ruletest provides values of `dateTime2` for three instances of `Entity1`. Input and Output panels are shown below.



Add months

SYNTAX

`<DateTime>.addMonths(<Integer>)`

`<Date>.addMonths(<Integer>)`

DESCRIPTION

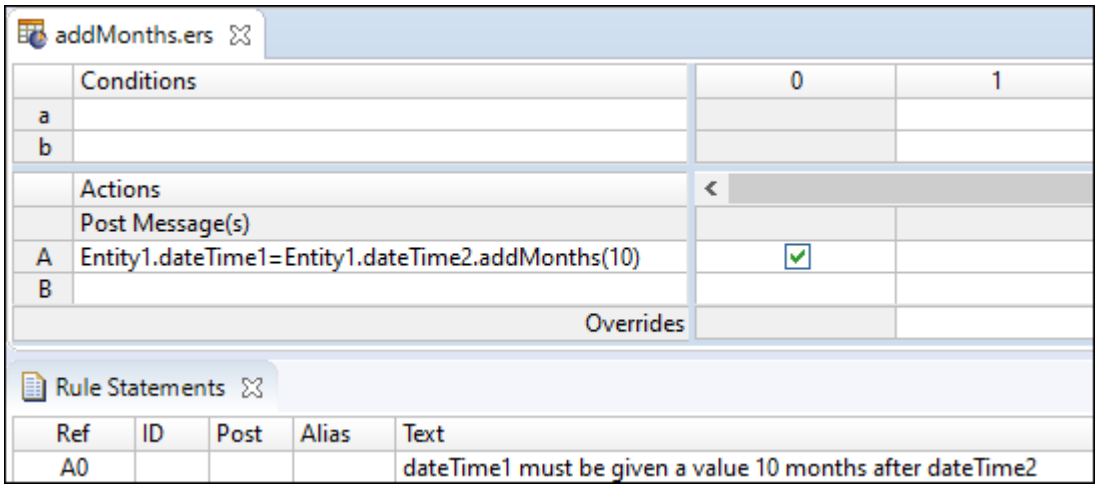
Adds the number of months in `<Integer>` to the number of months in `<DateTime>` or `<Date>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

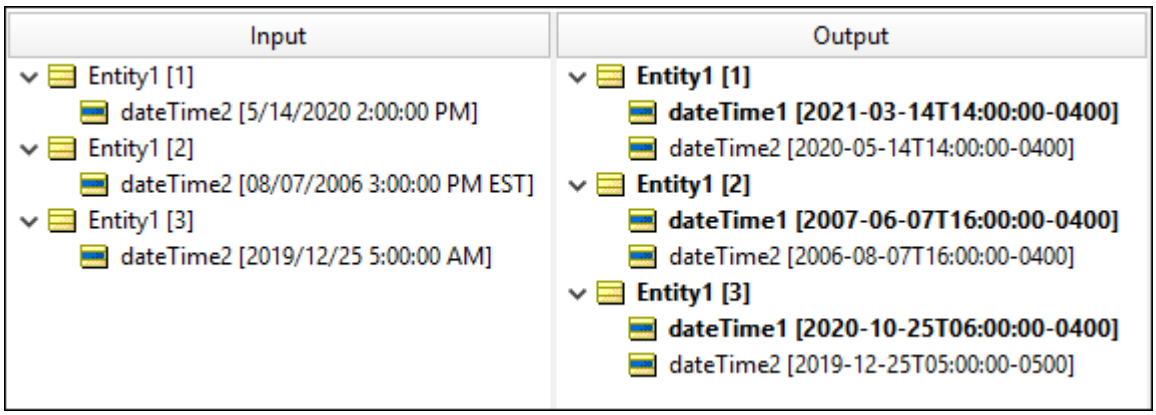
RULESHEET EXAMPLE

This sample Rulesheet uses **. addMonths** in a Nonconditional rule to add `10` months to the value of `dateTime2` and assign the result to `dateTime1`.



SAMPLE RULETEST

A sample Ruletest provides values of `dateTime2` for three instances of `Entity1`. Input and Output panels are shown below. Notice the year portion of `dateTime1` also changes accordingly.



Add seconds

SYNTAX

`<DateTime>.addSeconds(<Integer>)`

DESCRIPTION

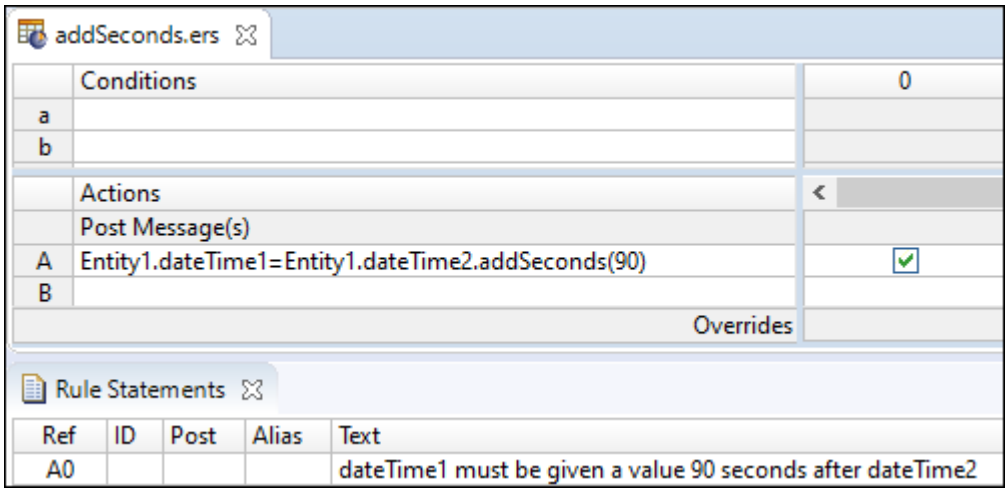
Adds the number of seconds in `<Integer>` to the number of seconds in the Time portion of `<DateTime>` .

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

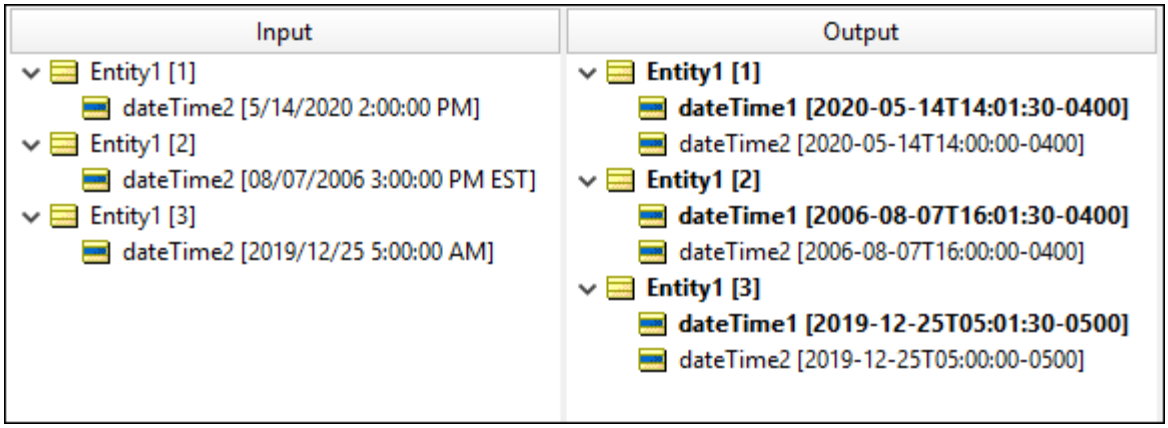
RULESHEET EXAMPLE

This sample Rulesheet uses **.addSeconds** in a Nonconditional rule to add `90` seconds to the value of `dateTime2` and assign the result to `dateTime1`.



SAMPLE RULETEST

A sample Ruletest provides values of `dateTime2` for three instances of `Entity1`. Input and Output panels are shown below.



Add years

SYNTAX

`<DateTime>.addYears(<Integer>)`

`<Date>.addYears(<Integer>)`

DESCRIPTION

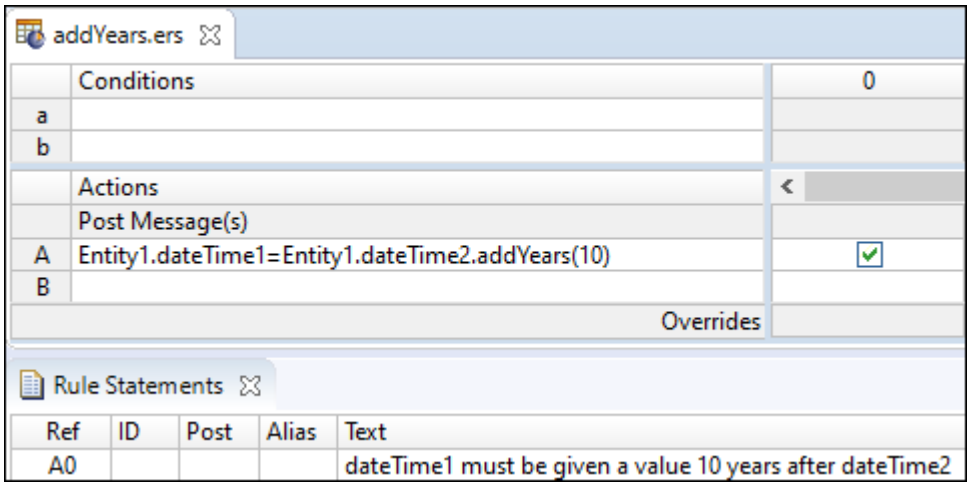
Adds the number of years in `<Integer>` to the number of years in the Date portion of `<DateTime>` or `<Date>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

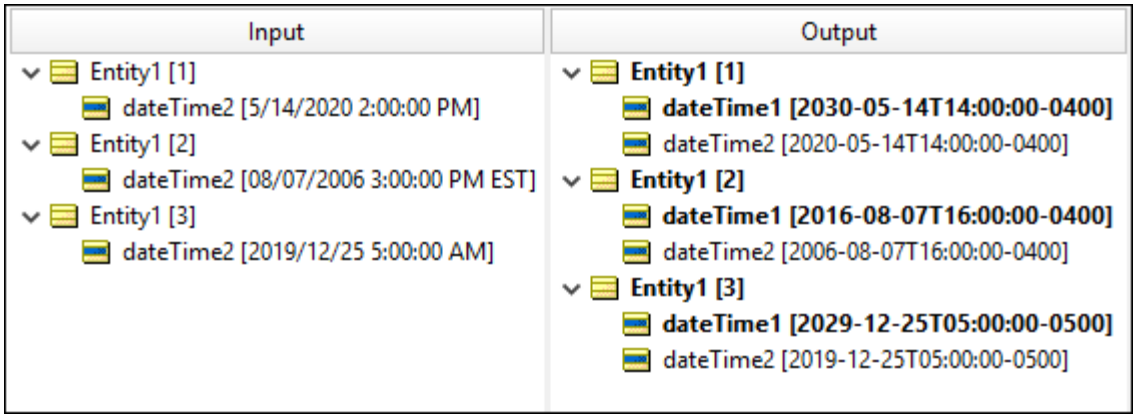
RULESHEET EXAMPLE

This sample Rulesheet uses **.addYears** in a Nonconditional rule to add `10` years to the value of `dateTime2` and assign the result to `dateTime1`.



SAMPLE RULETEST

A sample Ruletest provides values of `dateTime2` for three instances of `Entity1`. Input and Output panels are shown below.



After date

SYNTAX

`<DateTime1>.afterDate(<DateTime2>)`

DESCRIPTION

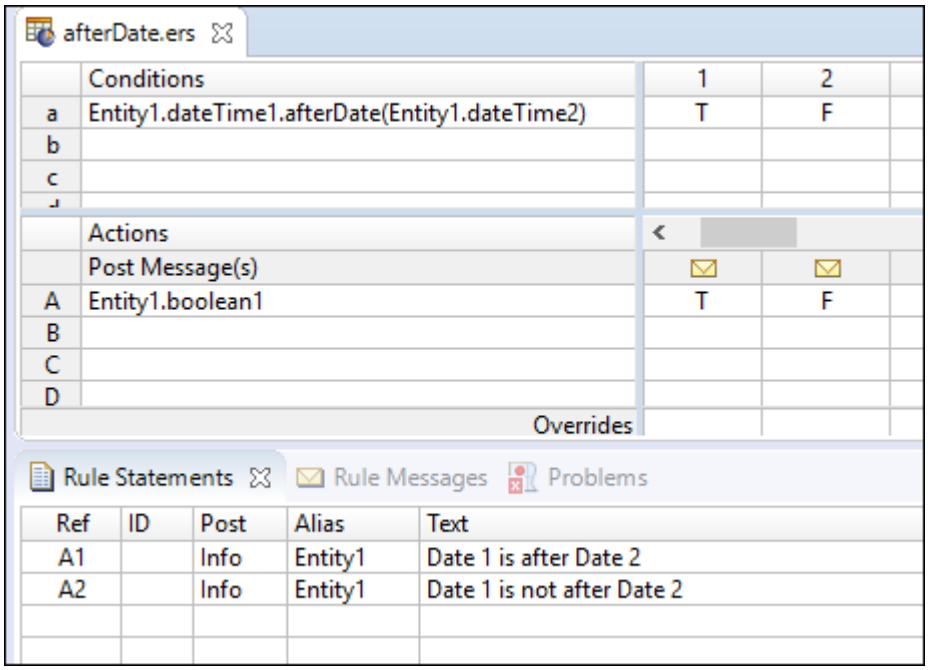
Returns boolean. True if DateTime1 is after DateTime2, ignoring the time part.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

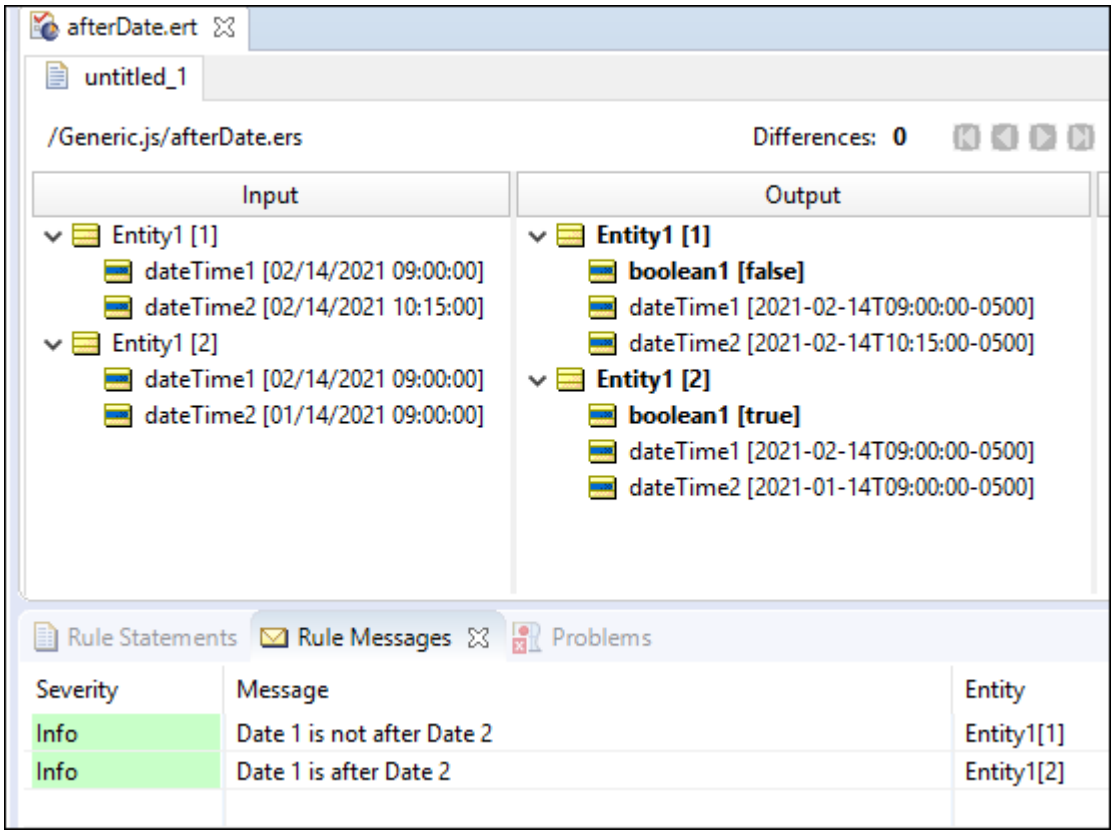
RULESHEET EXAMPLE

The following Rulesheet uses **.afterDate** to determine whether a dateTime date value is a later date than another.



SAMPLE RULETEST

A sample Ruletest provides `dateTime1` and `dateTime2` for two examples. Input and Output panels are shown below.



After time

SYNTAX

`<DateTime1>afterTime(<DateTime2>)`

DESCRIPTION

Returns boolean. True if DateTime1 is after DateTime2, ignoring the date part.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

RULESHEET EXAMPLE

The following Rulesheet uses **.afterTime** to determine whether a dateTime time value is a later time than another.

afterTime.ers

	Conditions	1	2
a	Entity1.dateTime1.afterTime(Entity1.dateTime2)	T	F
b			
c			
	Actions		
	Post Message(s)		
A	Entity1.boolean1	T	F
B			
C			
D			
	Overrides		

Rule Statements | Rule Messages | Problems

Ref	ID	Post	Alias	Text
A1		Info	Entity1	Time 1 is after Time 2
A2		Info	Entity1	Time 1 is not after Time 2

SAMPLE RULETEST

A sample Ruletest provides `dateTime1` and `dateTime2` for two examples. Input and Output panels are shown below.

*afterTime.ert

untitled_1

/Generic.js/afterTime.ers Differences: 0

Input	Output
Entity1 [1]	**Entity1 [1]**
dateTime1 [06/14/2015 09:15:00]	**boolean1 [false]**
dateTime2 [06/14/2014 09:30:00]	dateTime1 [2015-06-14T09:15:00-0400]
Entity1 [2]	dateTime2 [2014-06-14T09:30:00-0400]
dateTime1 [09/14/2015 09:45:00]	**Entity1 [2]**
dateTime2 [06/14/2014 09:30:00]	**boolean1 [true]**
	dateTime1 [2015-09-14T09:45:00-0400]
	dateTime2 [2014-06-14T09:30:00-0400]

Rule Statements | Rule Messages | Problems

Severity	Message	Entity
Info	Time 1 is not after Time 2	Entity1[1]
Info	Time 1 is after Time 2	Entity1[2]

Associate elements

SYNTAX

`<Collection1> += <Collection2>`

`<Collection1> += <Entity>`

DESCRIPTION

Associates all elements of `<Collection2>` or a single element named `<Entity>` with `<Collection1>`, provided such an association is allowed by the Vocabulary. Every collection must be uniquely identified with an alias or role.

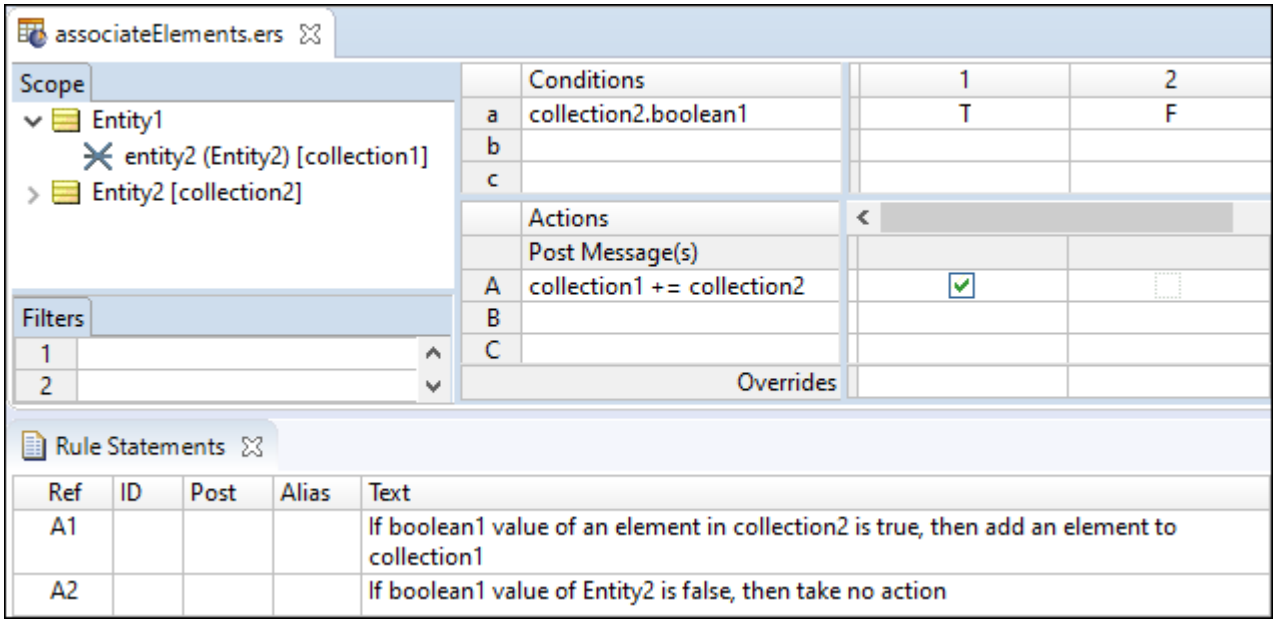
If the cardinality of the association between the parent entity of `<Collection>` and the `<Entity>` being added is “one-to-one” (a straight line icon beside the association in the Rule Vocabulary), then this **associate element** syntax is not used. Instead, replace element syntax is used, since the collection can contain only one element, and any element present will be replaced by the new element.

USAGE RESTRICTIONS

The Operators row of the table in Vocabulary usage restrictions does not apply. Special exceptions: **associate element** may only be used in Action Rows (section 5 in Sections of Rulesheet that correlate with usage restrictions).

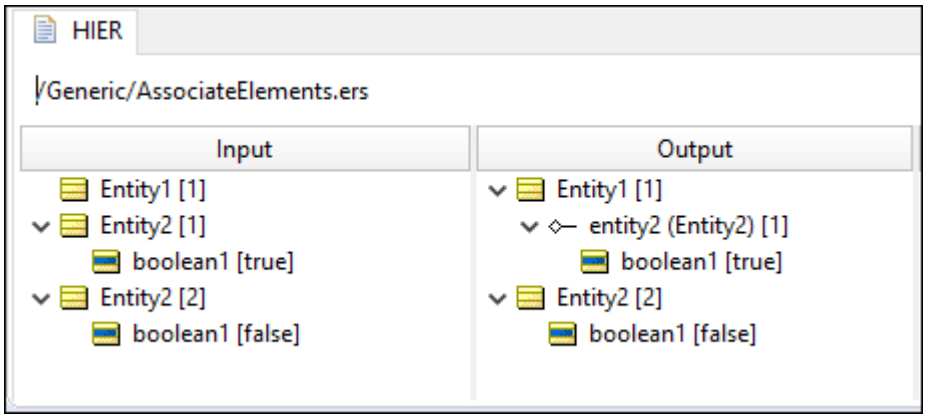
RULESHEET EXAMPLE

The following Rulesheet uses **associate element** to associate an element of `collection2` to `collection1` when `boolean1` value of any element in `collection2` is `true`. Note that the Action is not associating *all* elements in `collection2` with `collection1`, *only* those elements within `collection2` that satisfy the condition.



SAMPLE RULETEST: HIER

A sample Ruletest provides two examples of `Entity2` with `boolean1` values, and a single `Entity1`. Input and Output panels shows the association embedded in the parent entity:

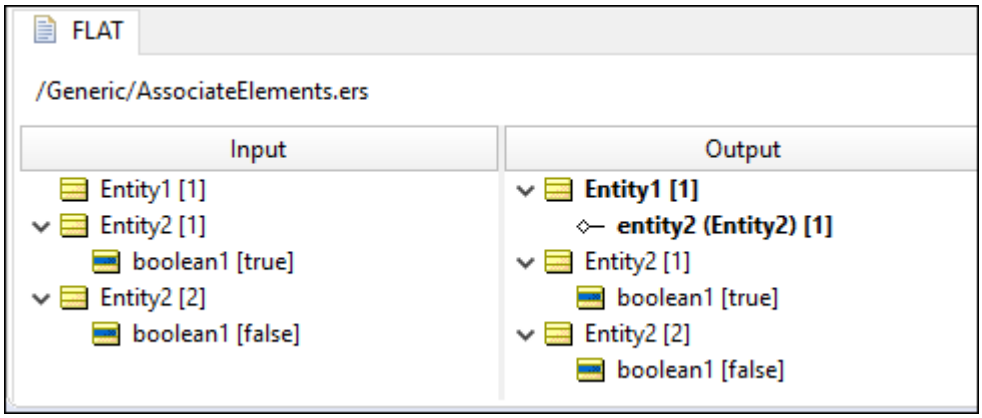


SAMPLE RULETEST: FLAT

Setting two properties in the Studio's `brms.properties` file enables a Flat payload:

```
com.corticon.tester.ccserver.execute.format=XML
com.corticon.designer.tester.xmlmessagingstyle=Flat
```

After restarting Studio, running the same sample Ruletest shows the association dropping to the root with an href entity:



At

SYNTAX

<Sequence> ->at(<Integer>).<Attribute1>

DESCRIPTION

Returns the value of <Attribute1> for the element at position <Integer> in <Sequence>. Another operator, such as ->sortedBy, must be used to transform a <Collection> into a <Sequence> before **->at** may be used. <Sequence> must be expressed as a unique alias. See *"Advanced collection sorting syntax" in the Rule Modeling Guide* for more examples of usage.

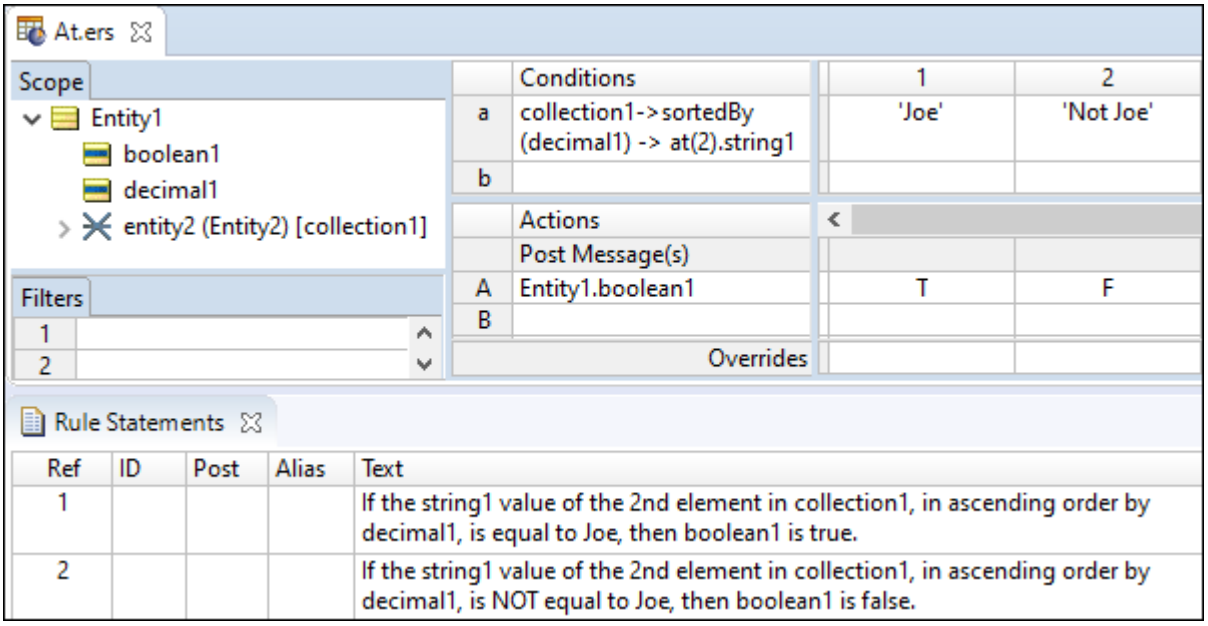
<Attribute1> may be of any data type.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

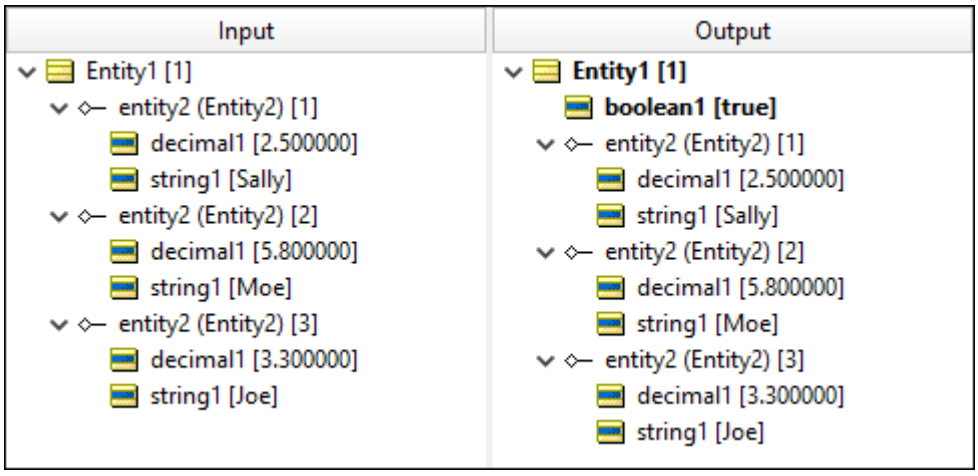
RULESHEET EXAMPLE

This sample Rulesheet uses **->at(2)** to identify the second element of the sequence created by applying sortedBy to `collection1`. Once identified, the value of the `string1` attribute belonging to this second element is evaluated. If the value of `string1` is `Joe`, then `boolean1` attribute of `Entity1` is assigned the value of `true`.



SAMPLE RULETEST

A sample Ruletest provides a collection of three elements, each with a `decimal1` value. Input and Output panels are shown below.



Average

SYNTAX

`<Collection.attribute>` ->avg

DESCRIPTION

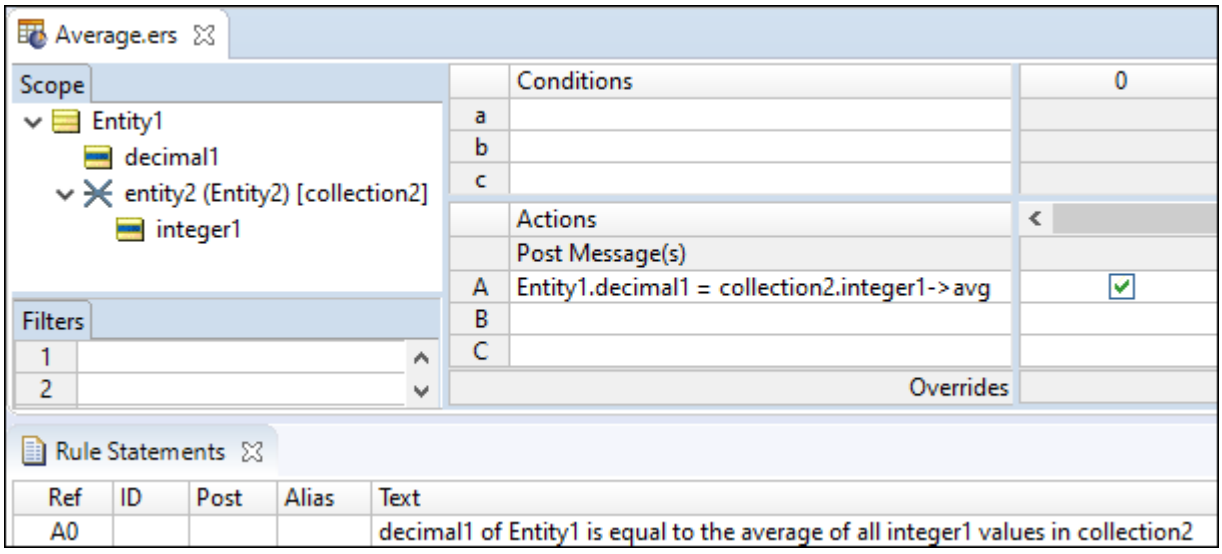
Averages the values of all of the specified attributes in `<Collection>`. `<Collection>` must be expressed as a unique alias. `<attribute>` must be a numeric data type.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

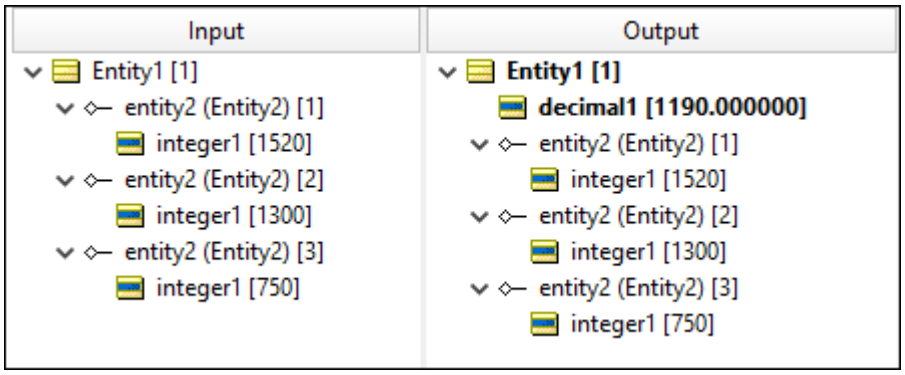
RULESHEET EXAMPLE

This sample Rulesheet uses **->avg** to average the `integer1` values of all elements in `collection2`, then assigns the resulting value to `decimal1` in `Entity1`. Note the use of the alias `collection2` to represent the collection of `Entity2` elements associated with `Entity1`.



SAMPLE RULETEST

A sample Ruletest provides `integer1` values for three elements in `collection2`. The following illustration shows Input and Output panels:



Before date

SYNTAX

`<DateTime1>`.beforeDate(`<DateTime2>`)

DESCRIPTION

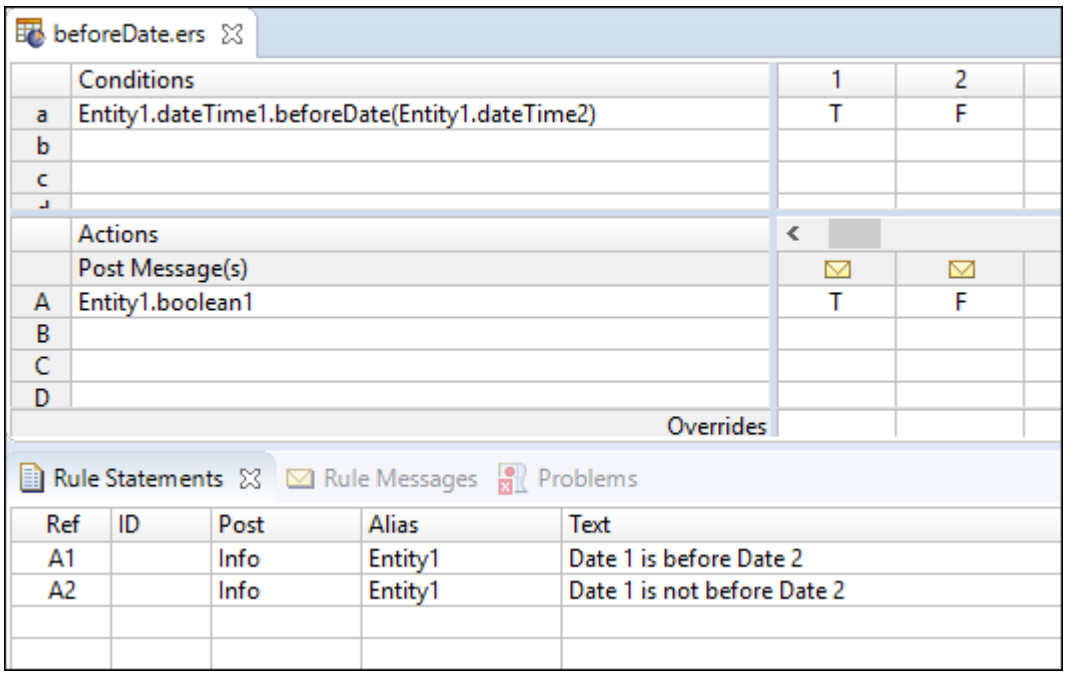
Returns boolean. True if DateTime1 is an earlier date than DateTime2, ignoring the time part.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

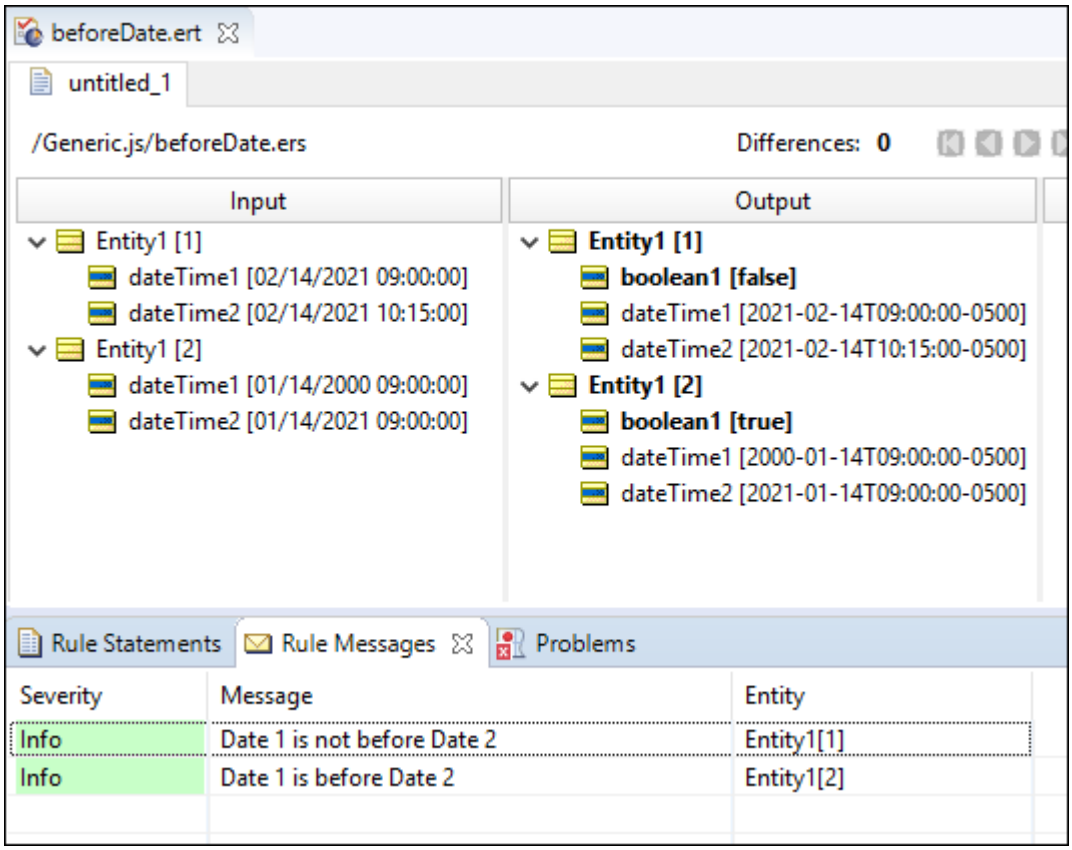
RULESHEET EXAMPLE

The following Rulesheet uses **.beforeDate** to determine whether a dateTime value is an earlier date than another.



SAMPLE RULETEST

A sample Ruletest provides `dateTime1` and `dateTime2` for two examples. Input and Output panels are shown below.



Before time

SYNTAX

`<DateTime1>.beforeTime(<DateTime2>)`

DESCRIPTION

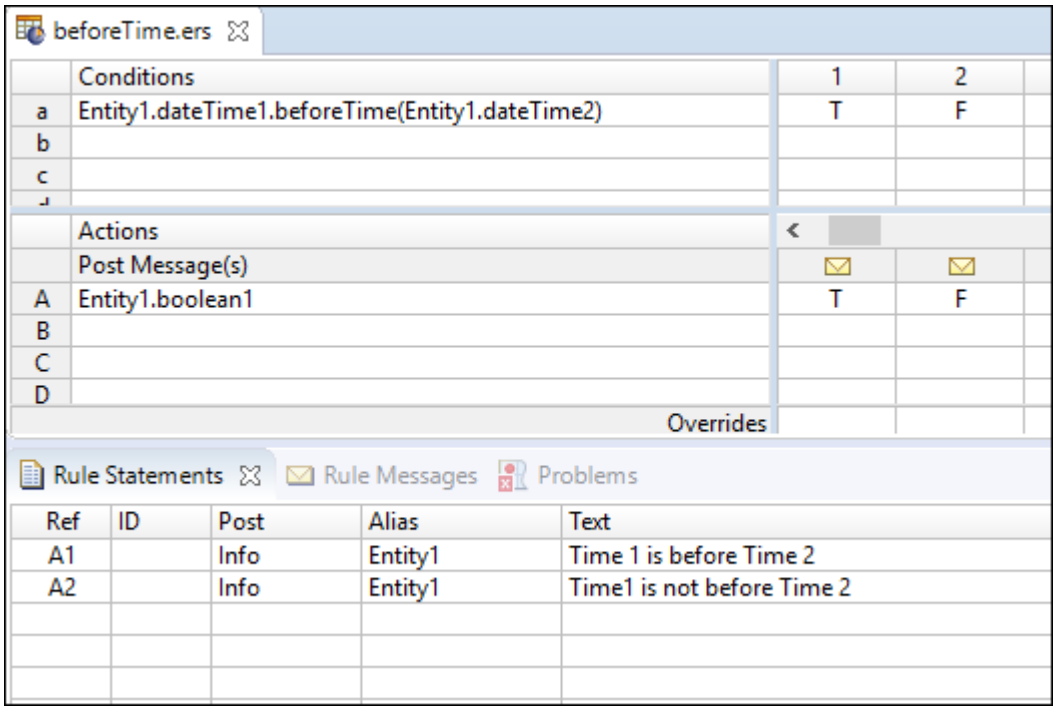
Returns boolean. True if DateTime1 is before DateTime2, ignoring the date part.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

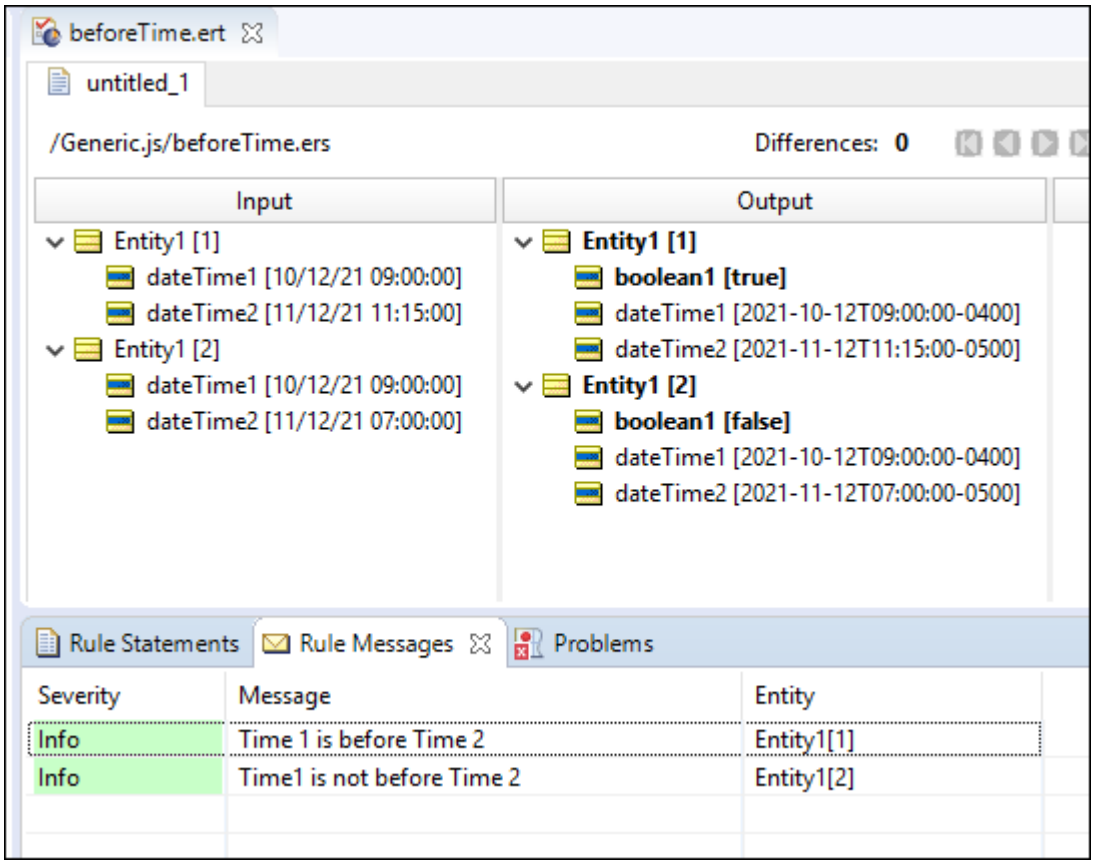
RULESHEET EXAMPLE

The following Rulesheet uses **.beforeTime** to determine whether a dateTime time value is an earlier date than another.



SAMPLE RULETEST

A sample Ruletest provides `dateTime1` and `dateTime2` for two examples. Input and Output panels are shown below.



Ceiling

SYNTAX

`<Decimal>`.ceiling

DESCRIPTION

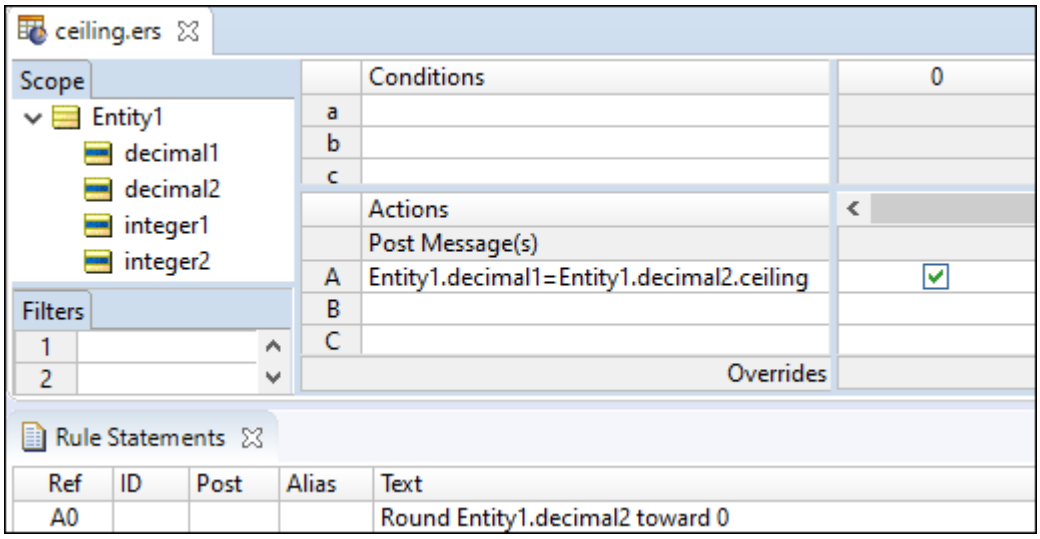
Returns the Decimal furthest from `<Decimal>`. **.ceiling** may also be thought of as a rounding to a whole number in the direction of positive infinity.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

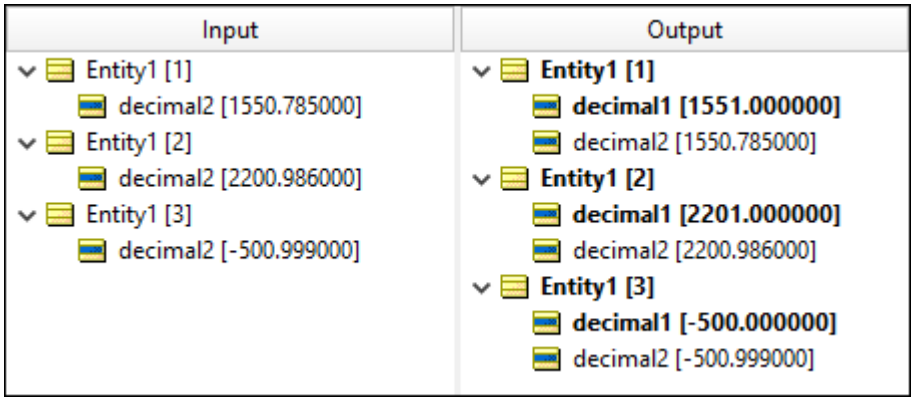
RULESHEET EXAMPLE

The Rulesheet uses **.ceiling** to assign Decimal values to `decimal1` that are closer to zero than the input `decimal2` values.



SAMPLE RULETEST

A sample Ruletest provides three `decimal2` values. Input and Output panels are shown below:



CellValue

SYNTAX

Various, see Examples below

DESCRIPTION

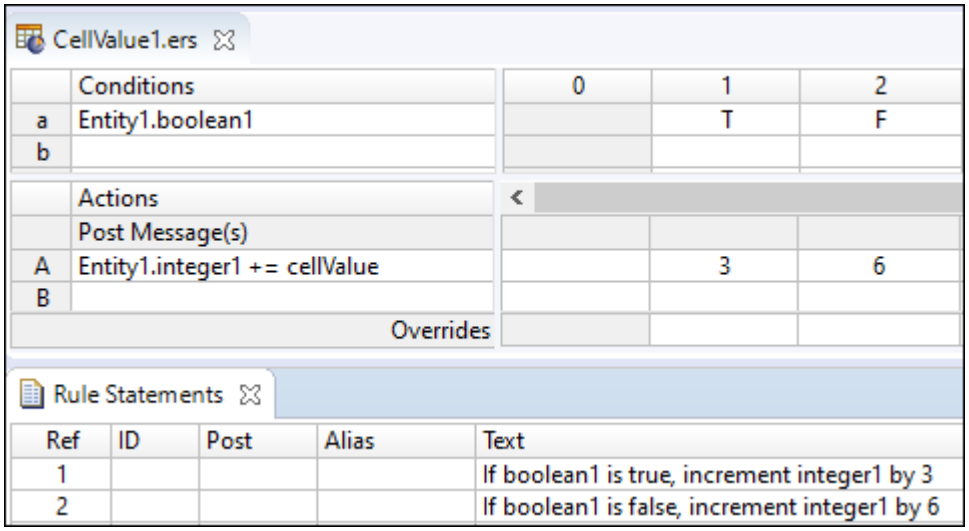
When used in an expression, **cellValue** performs text replacement where the value is determined by the rule Column that executes. Using **cellValue** in a Condition or Action expression eliminates the need for multiple, separate Rows to express the same logic.

USAGE RESTRICTIONS

The Operators row of the table in Vocabulary usage restrictions does not apply. Special exceptions: **cellValue** may only be used in Condition and Action Rows (sections 3 and 5 in Sections of Rulesheet that correlate with usage restrictions).

RULESHEET EXAMPLE 1

This sample Rulesheet uses **cellValue** to increment `integer1` by the amount in the Action Cell of the rule Column that fires. An equivalent Rulesheet which does not use cellValue is also shown for comparison purposes.



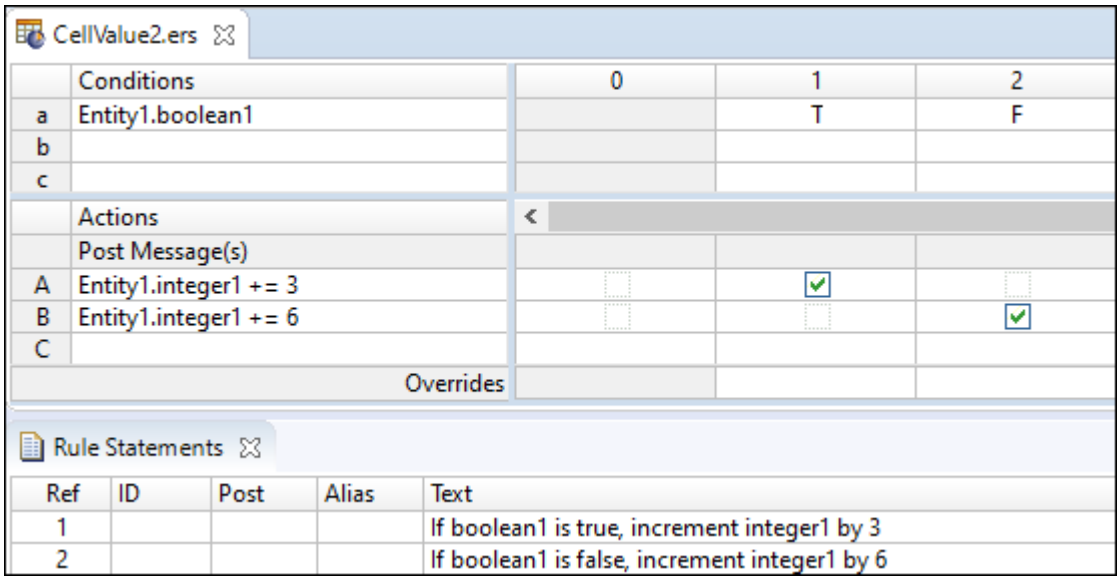
CellValue1.ers

	Conditions	0	1	2
a	Entity1.boolean1		T	F
b				
	Actions			
	Post Message(s)			
A	Entity1.integer1 += cellValue		3	6
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1				If boolean1 is true, increment integer1 by 3
2				If boolean1 is false, increment integer1 by 6

Equivalent Rulesheet without using **cellValue**:



CellValue2.ers

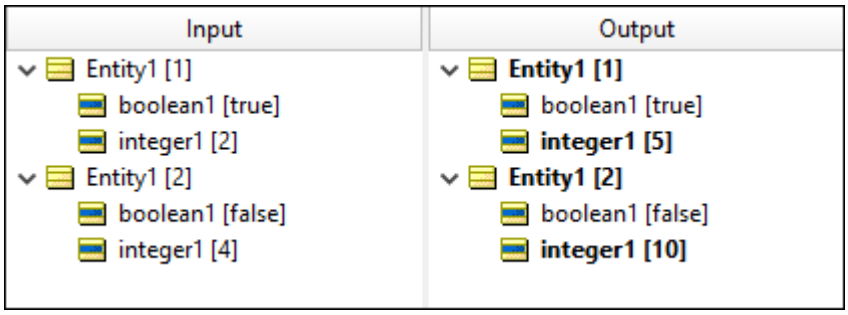
	Conditions	0	1	2
a	Entity1.boolean1		T	F
b				
c				
	Actions			
	Post Message(s)			
A	Entity1.integer1 += 3	☐	☑	☐
B	Entity1.integer1 += 6	☐	☐	☑
C				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1				If boolean1 is true, increment integer1 by 3
2				If boolean1 is false, increment integer1 by 6

SAMPLE RULETEST 1

A sample Ruletest provides two examples of `boolean1`. The following table shows Input and Output panels.



Input	Output
Entity1 [1]	**Entity1 [1]**
boolean1 [true]	boolean1 [true]
integer1 [2]	**integer1 [5]**
Entity1 [2]	**Entity1 [2]**
boolean1 [false]	boolean1 [false]
integer1 [4]	**integer1 [10]**

Character at

SYNTAX

<String>.characterAt(*index*:Integer)

DESCRIPTION

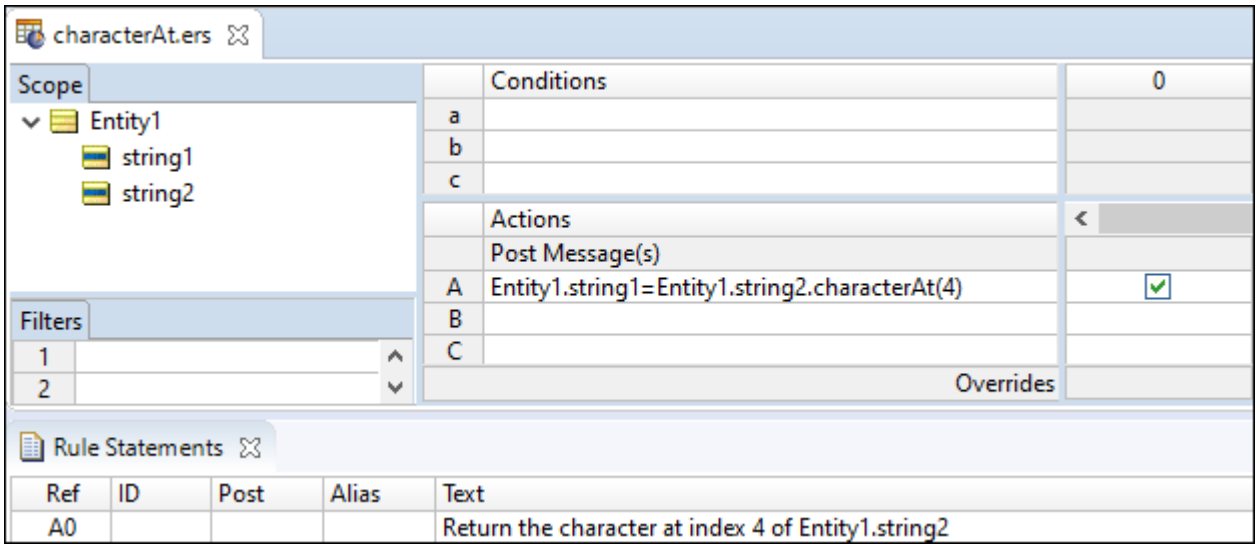
Returns the character at the specified position in the String.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

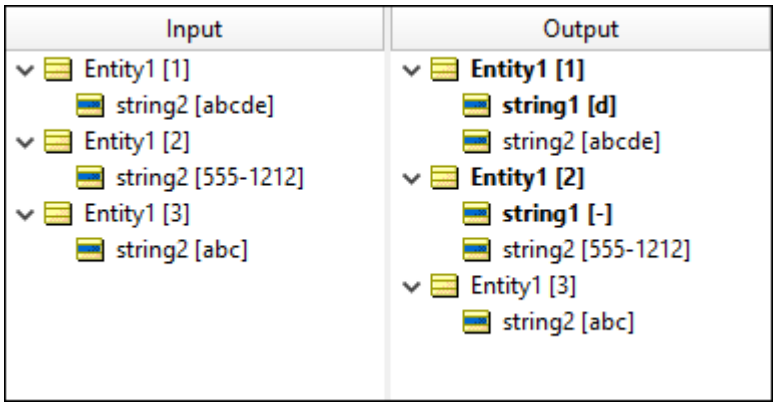
RULESHEET EXAMPLE

This action-only operator parses the specified string, and then returns that character to the return character string.



SAMPLE RULETEST

A sample Ruletest provides three elements that point out (1) the expected behavior, (2) the result when the character is not alphanumeric, and (3) a null when there is no character at that position in the String.



Clone

SYNTAX

`<Entity>.clone[<Expression1>,<Expression2>...]`

DESCRIPTION

Copies the specified `Entity` and its attribute values to a new `Entity` where `Expressions` (in the form *`attribute=value`*) override the corresponding cloned attribute values. The new `Entity` has no associations. Where an Entity specifies an Entity Identity, that identity is not copied to its clone entity. For each `Entity` in `Collection`, the operator creates a duplicate of `Entity`. The implementation is a *shallow clone* -- associations are not duplicated.

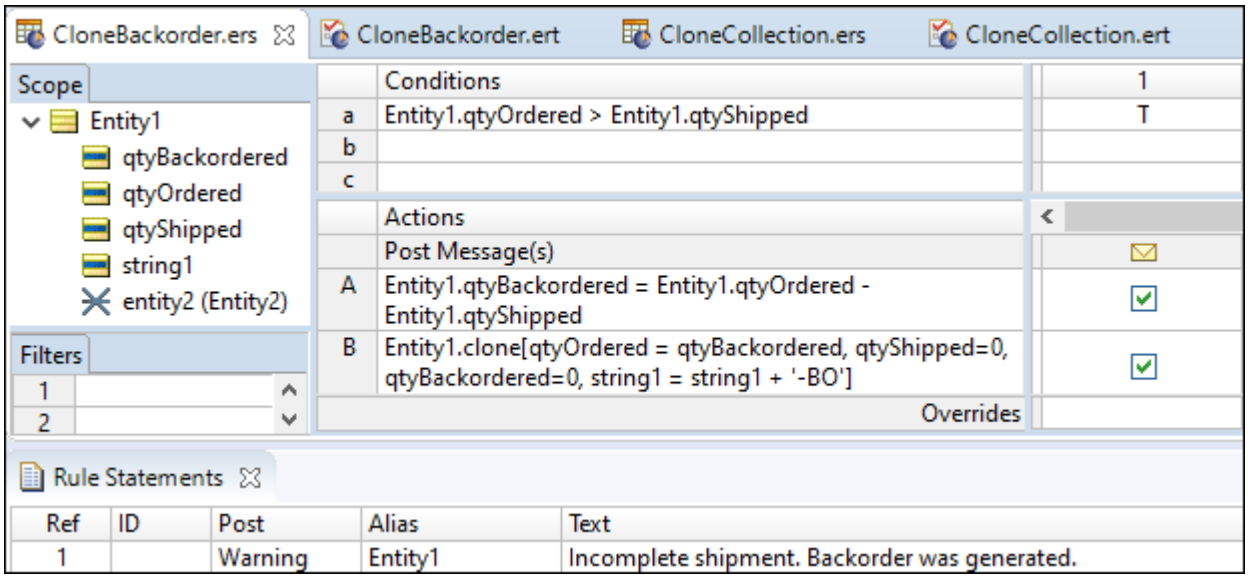
USAGE RESTRICTIONS

The Operators row in the table of Summary Table of Vocabulary Usage Restriction does not apply. Special exceptions: **clone** may only be used in Action Rows (section 5 in Sections of Rulesheet that correlate with usage restrictions).

Nested clone calls are not supported, such as `E1.clone[assoc1 += E1.assoc1.clone[…]]`.

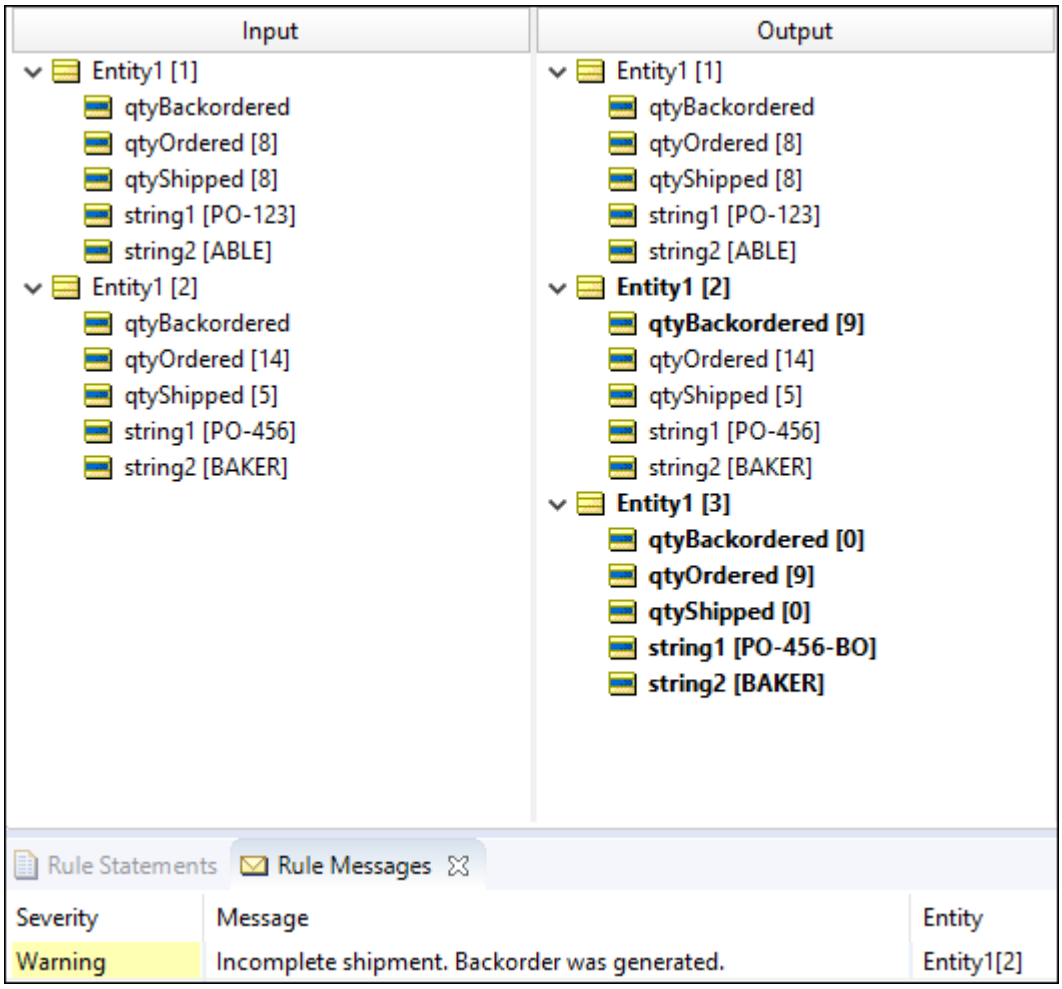
RULESHEET EXAMPLE

The following Rulesheet uses **.clone** to create a new `Entity2` element when the value of `qtyOrdered` in `Entity1` is greater than the `qtyShipped` value. An alias is not required by the **.clone** operator, because it is possible to create a new entity at the root level, without inserting it into a collection.



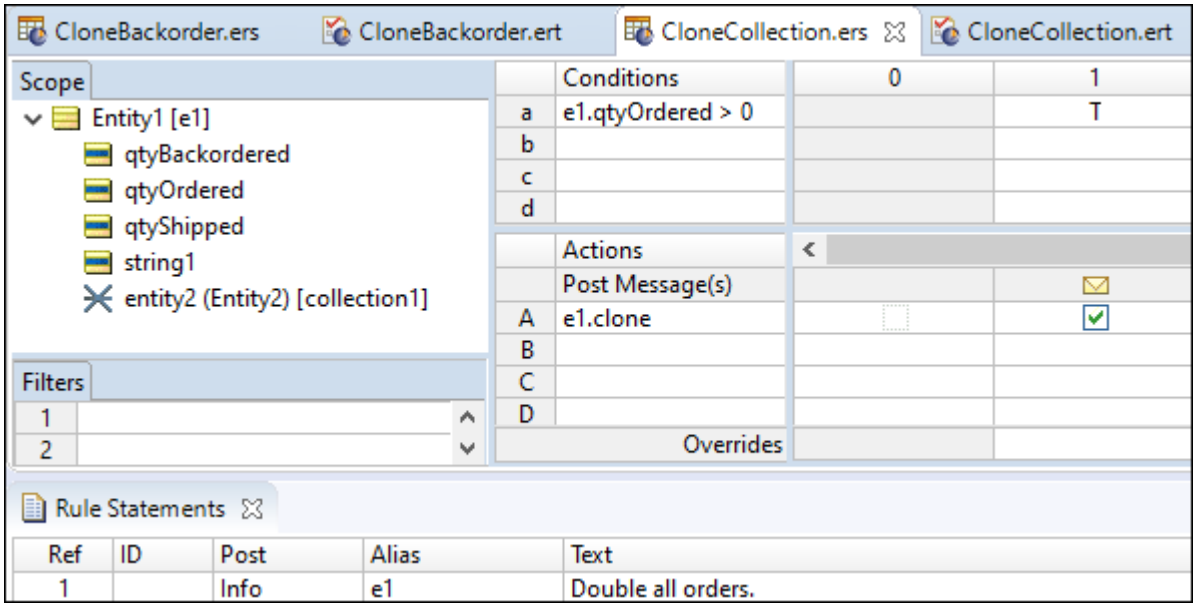
SAMPLE RULETEST

A sample Ruletest provides two collections of `Entity1`. Input, Output, and Expected panels are as follows:



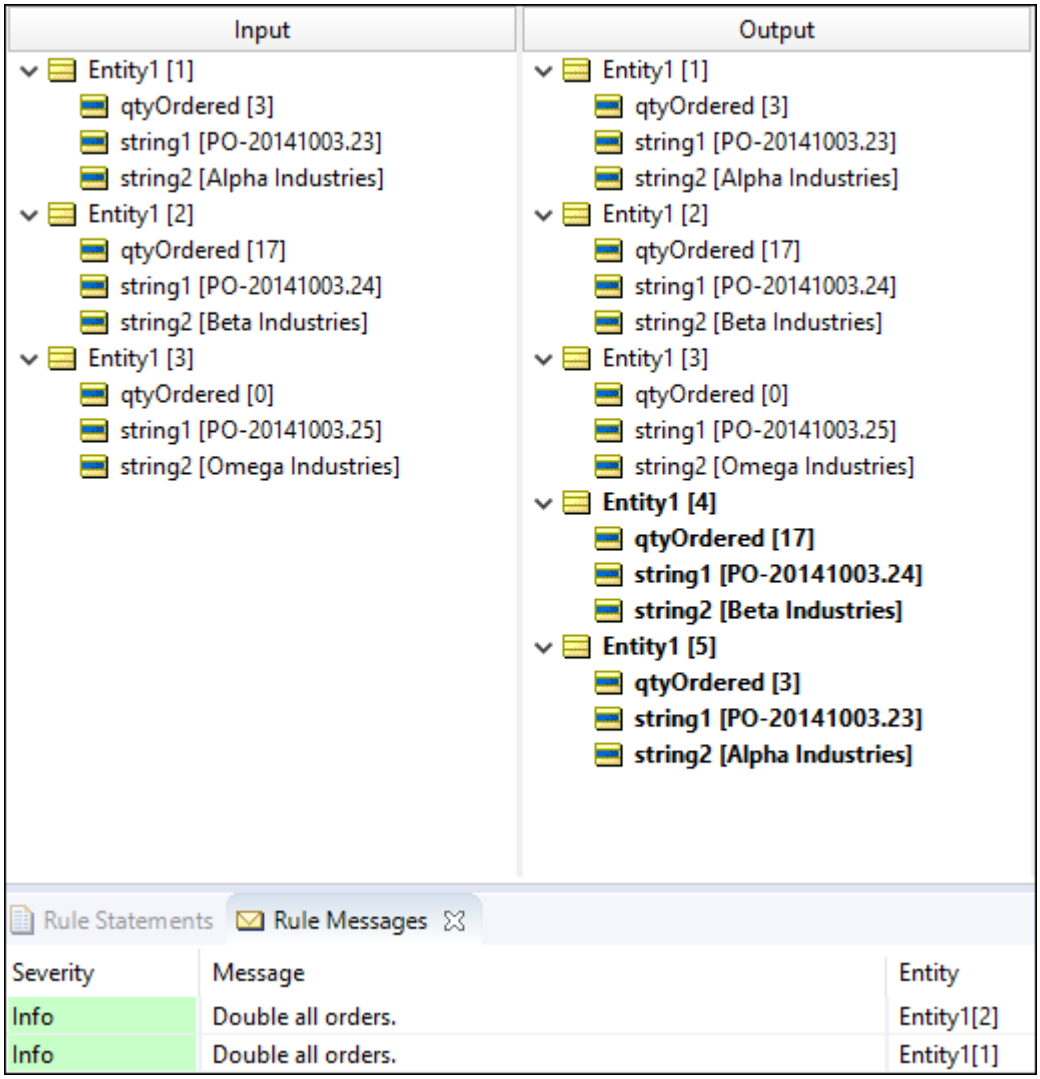
RULESHEET EXAMPLE: COLLECTION

The following Rulesheet uses **.clone** to create a new `Entity2` element in `collection1` when `Entity1` has a non-zero `qtyOrdered` value.



SAMPLE RULETEST: COLLECTION

A sample Ruletest provides three collections of `Entity1`. Input and Output panels are illustrated below:



Concatenate

SYNTAX

`<String1>.concat(<String2>)`

DESCRIPTION

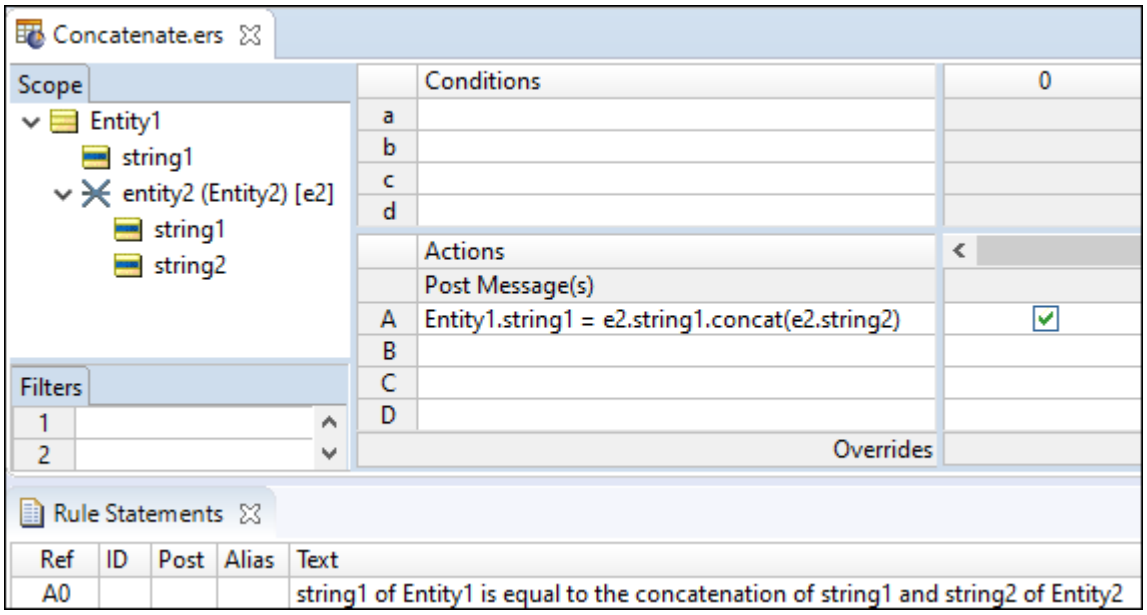
Concatenates `<String1>` to `<String2>`, placing `<String2>` at the end of `<String1>`

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

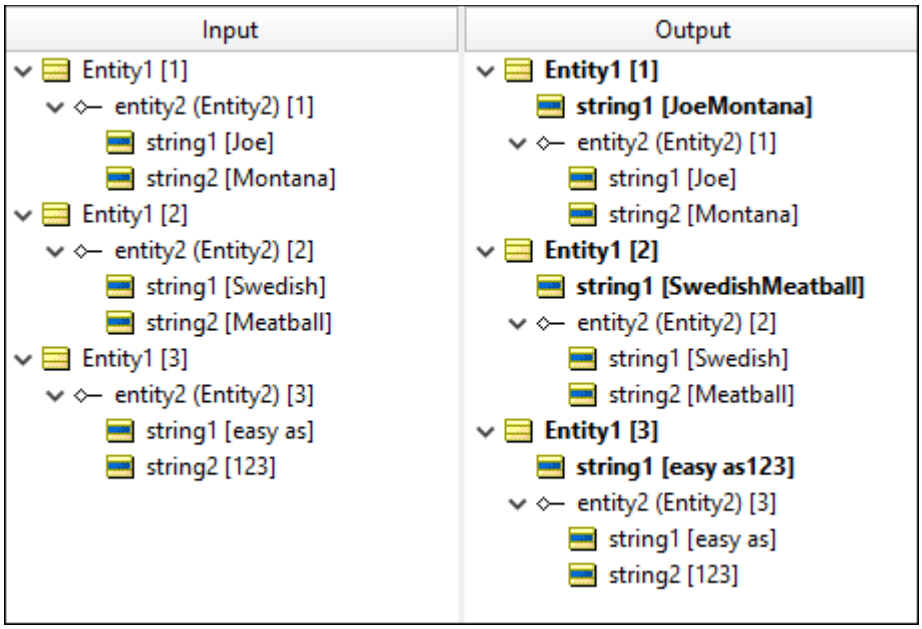
RULESHEET EXAMPLE

This sample Rulesheet uses **.concat** to create `string1` by combining `string1` and `string2` from `Entity1.entity2`.



SAMPLE RULETEST

A sample Ruletest provides three examples of `string1` and `string2`. Input and Output panels are shown below.



Contains

SYNTAX

`<String1>`.contains(`<String2>`)

DESCRIPTION

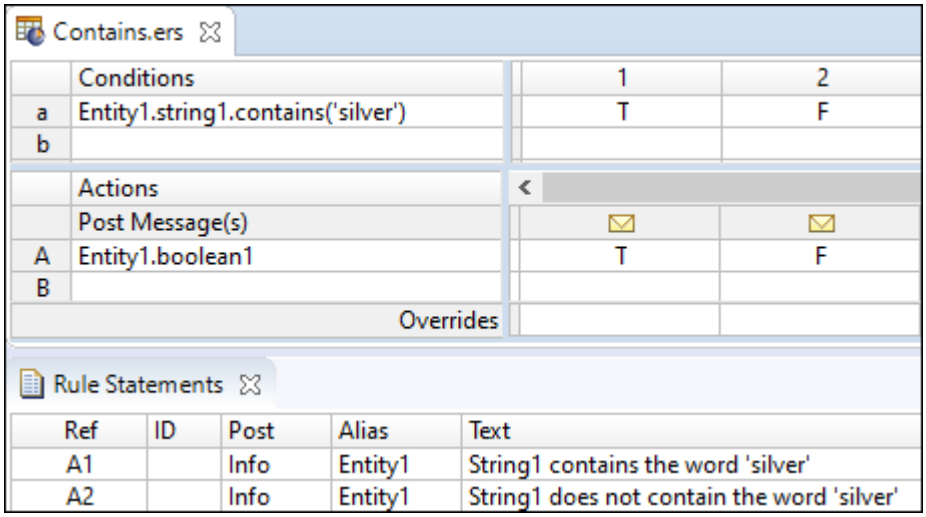
Evaluates `<String1>` and returns a value of true if it contains or includes the exact (case-sensitive) characters specified in `<String2>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

RULESHEET EXAMPLE 1

The following uses **.contains** to evaluate whether `string1` includes the characters `silver` and assigns a value to `boolean1` for each outcome.



Contains.ers

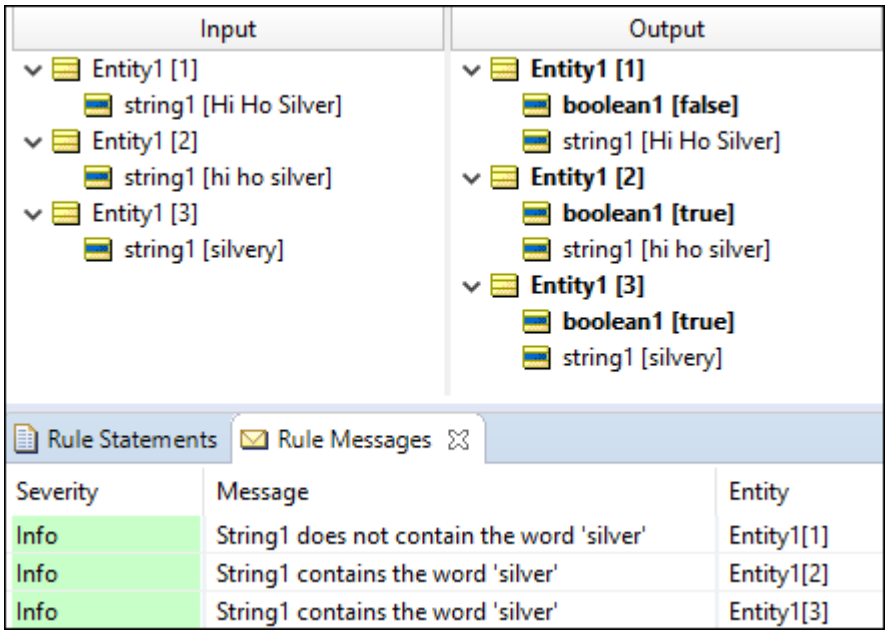
	Conditions	1	2
a	Entity1.string1.contains('silver')	T	F
b			
	Actions		
	Post Message(s)	✉	✉
A	Entity1.boolean1	T	F
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
A1		Info	Entity1	String1 contains the word 'silver'
A2		Info	Entity1	String1 does not contain the word 'silver'

SAMPLE RULETEST 1

A sample Ruletest provides `string1` values for three examples. Input and Output panels are shown below. Note case sensitivity in these examples. Posted messages are not shown.



Day

SYNTAX

`<DateTime>`.day

`<Date>`.day

DESCRIPTION

Returns the day portion of `<DateTime>` or `<Date>` as an Integer between 1 and 31.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.day** to assign a value to `string1` and post a message.

day.ers

	Conditions	0	1	2
a	Entity1.dateTime1.day		< 15	>= 15
b				
	Actions			
	Post Message(s)			
A	Entity1.string1		'Hold'	'Ship'
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
A1		Warning	Entity1	If the day of dateTime1 is earlier than the 15th, then assign string1 a value of 'Hold' and issue a Warning Message
A2		Info	Entity1	If the day of dateTime1 is on or after the 15th, then assign string1 a value of 'Ship' and issue an Info Message

SAMPLE RULETEST

A sample Ruletest provides `dateTime1` values for three examples. Input and Output panels are shown below. Posted messages are not shown.

Input	Output
Entity1 [1]	**Entity1 [1]**
dateTime1 [5/14/2020 2:00:00 PM]	dateTime1 [2020-05-14T14:00:00-0400]
Entity1 [2]	**string1 [Hold]**
dateTime1 [08/07/2006 3:00:00 PM EST]	**Entity1 [2]**
Entity1 [3]	dateTime1 [2006-08-07T16:00:00-0400]
dateTime1 [2019/12/25 5:00:00 AM]	**string1 [Hold]**
	Entity1 [3]
	dateTime1 [2019-12-25T05:00:00-0500]
	string1 [Ship]

Day of week

SYNTAX

`<DateTime>`.dayOfWeek

`<Date>`.day

DESCRIPTION

Returns an Integer between 1 and 7, corresponding to the table below:

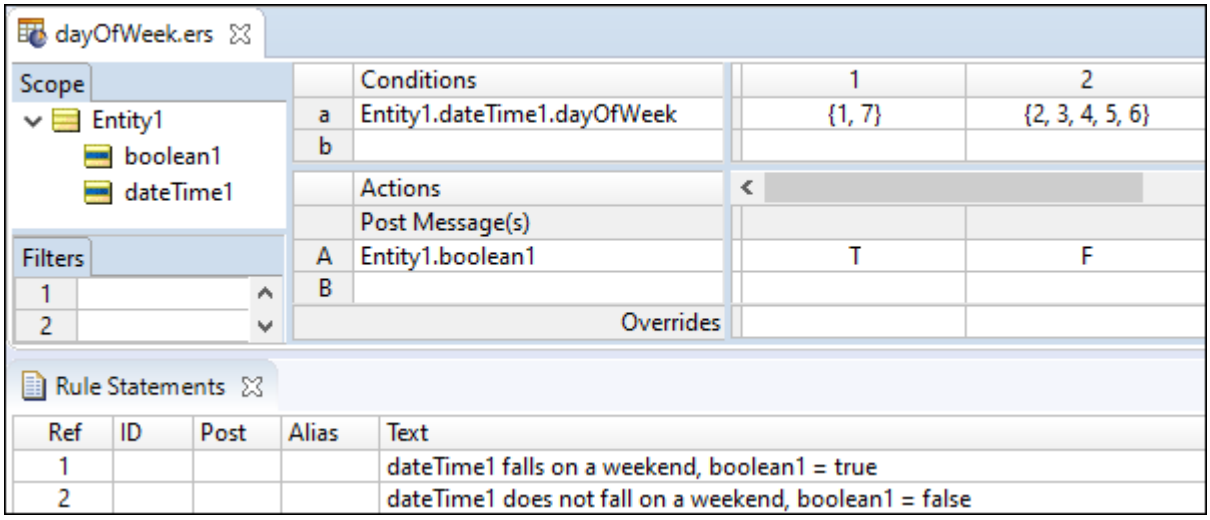
returned Integer	day of the week
1	Sunday
2	Monday
3	Tuesday
4	Wednesday
5	Thursday
6	Friday
7	Saturday

USAGE RESTRICTIONS

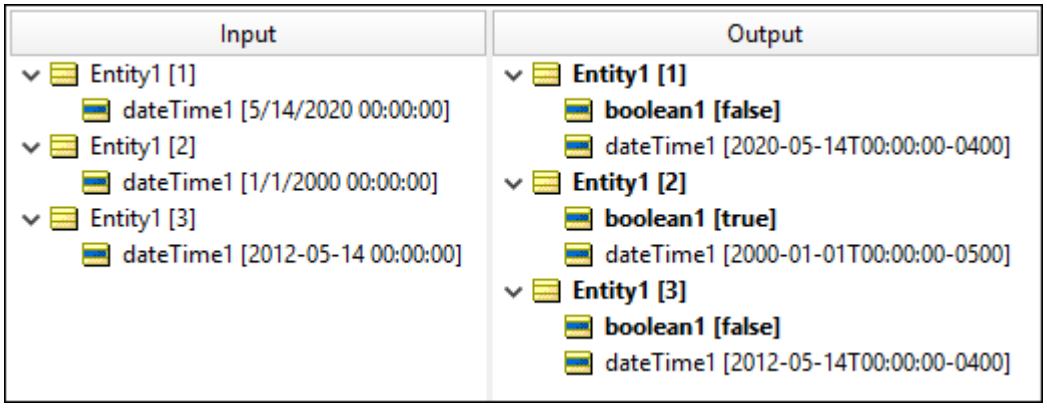
The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

RULESHEET EXAMPLE

The following Rulesheet uses **.dayOfWeek** to assign a value to `boolean1`.



SAMPLE RULETEST



Day of year

SYNTAX

`<DateTime>`.dayOfYear

`<Date>`.dayOfYear

DESCRIPTION

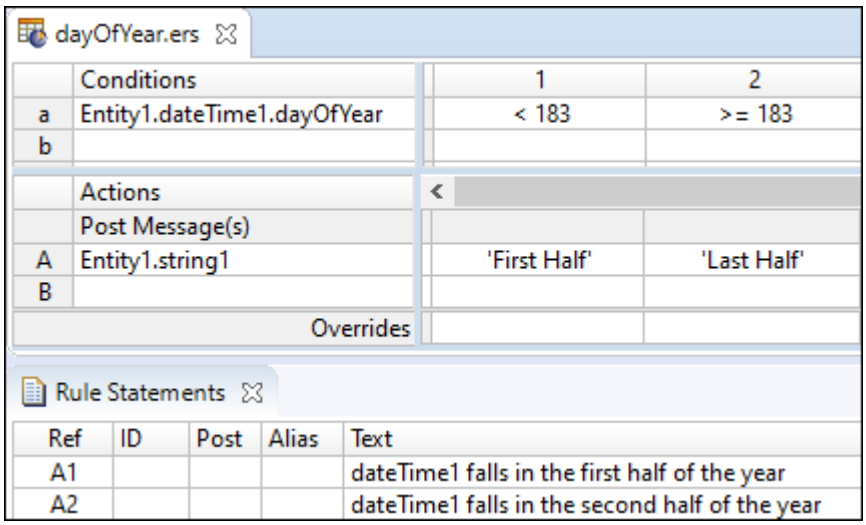
Returns an Integer from 1 to 366, equal to the day number within the year.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

RULESHEET EXAMPLE

The following Rulesheet uses **.dayOfYear** to assign a value to `string1`.



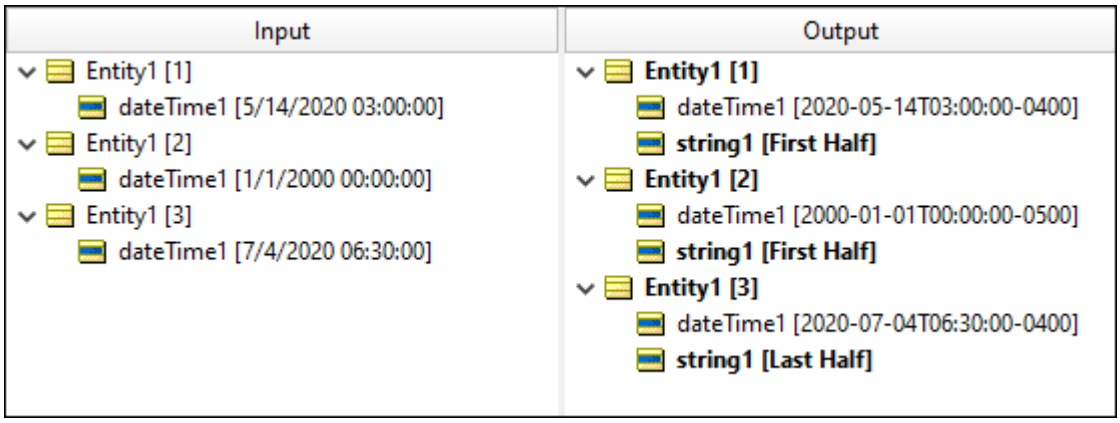
dayOfYear.ers

	Conditions	1	2
a	Entity1.dateTime1.dayOfYear	< 183	>= 183
b			
	Actions		
	Post Message(s)		
A	Entity1.string1	'First Half'	'Last Half'
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
A1				dateTime1 falls in the first half of the year
A2				dateTime1 falls in the second half of the year

SAMPLE RULETEST



Input	Output
Entity1 [1]	**Entity1 [1]**
dateTime1 [5/14/2020 03:00:00]	dateTime1 [2020-05-14T03:00:00-0400]
Entity1 [2]	**string1 [First Half]**
dateTime1 [1/1/2000 00:00:00]	**Entity1 [2]**
Entity1 [3]	dateTime1 [2000-01-01T00:00:00-0500]
dateTime1 [7/4/2020 06:30:00]	**string1 [First Half]**
	Entity1 [3]
	dateTime1 [2020-07-04T06:30:00-0400]
	string1 [Last Half]

Days between

SYNTAX

<DateTime1>.daysBetween(<DateTime2>)

<Date1>.daysBetween(<Date2>)

DESCRIPTION

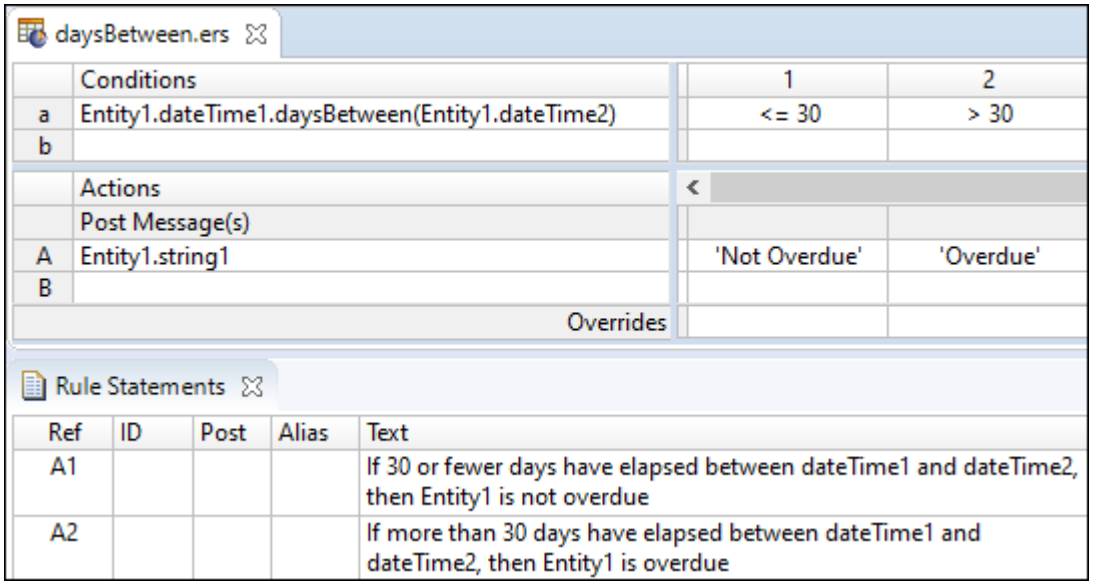
Returns the Integer number of days between DateTimes or Dates. This function calculates the number of milliseconds between the date values and divides that number by 86,400,000 (the number of milliseconds in a day). Any fraction is truncated, leaving an Integer result. If the two dates differ by less than a full 24-hour period, the value returned is zero. A positive Integer value is returned when <DateTime2> occurs after <DateTime1>.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

RULESHEET EXAMPLE

The following Rulesheet uses **.daysBetween** to determine the number of days that have elapsed between dateTime1 and dateTime2, compare it to the values in the Condition cells, and assign a value to string1.



daysBetween.ers

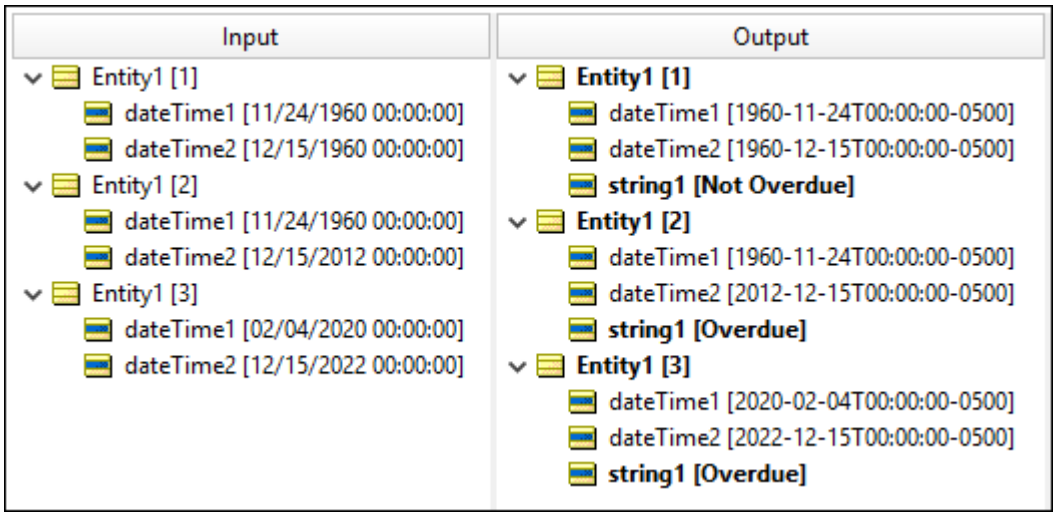
	Conditions	1	2
a	Entity1.dateTime1.daysBetween(Entity1.dateTime2)	<= 30	> 30
b			
	Actions		
	Post Message(s)		
A	Entity1.string1	'Not Overdue'	'Overdue'
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
A1				If 30 or fewer days have elapsed between dateTime1 and dateTime2, then Entity1 is not overdue
A2				If more than 30 days have elapsed between dateTime1 and dateTime2, then Entity1 is overdue

SAMPLE RULETEST

A sample Ruletest provides dateTime1 and dateTime2 for three examples. Input and Output panels are shown below.



Decrement

SYNTAX

`<Number1> -= <Number2>`

DESCRIPTION

Decrements `<Number1>` by the value of `<Number2>`. The data type of `<Number1>` must accommodate the subtraction of `<Number2>`. In other words, an Integer may not be decremented by a Decimal without using another operator (such as .toInteger or Floor on page 96) to first convert the Decimal to an Integer.

USAGE RESTRICTIONS

The Operators row of the table in Vocabulary usage restrictions does not apply. Special exceptions: **decrement** may only be used in Action Rows (section 5 in Sections of Rulesheet that correlate with usage restrictions).

RULESHEET EXAMPLE

This sample Rulesheet uses **decrement** to reduce `integer1` by the value of `integer2` when `boolean1` is `false`.

Decrement.ers

	Conditions	0	1
a	Entity1.boolean1		F
b			
	Actions		
	Post Message(s)		
A	Entity1.integer1 -= Entity1.integer2	☐	☑
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
A1				If boolean1 is false, then decrement integer1 by the value of integer2

SAMPLE RULETEST

A sample Ruletest provides three examples of `integer1`, `integer2`, and `boolean1`. Input and Output panels are shown below.

Input	Output
Entity1 [1]	Entity1 [1]
boolean1 [true]	boolean1 [true]
integer1 [10]	integer1 [10]
integer2 [5]	integer2 [5]
Entity1 [2]	**Entity1 [2]**
boolean1 [false]	boolean1 [false]
integer1 [12]	**integer1 [8]**
integer2 [4]	integer2 [4]
Entity1 [3]	Entity1 [3]
boolean1 [true]	boolean1 [true]
integer1 [25]	integer1 [25]
integer2 [10]	integer2 [10]

Disassociate elements

SYNTAX

`<Collection1> -= <Collection2>`

DESCRIPTION

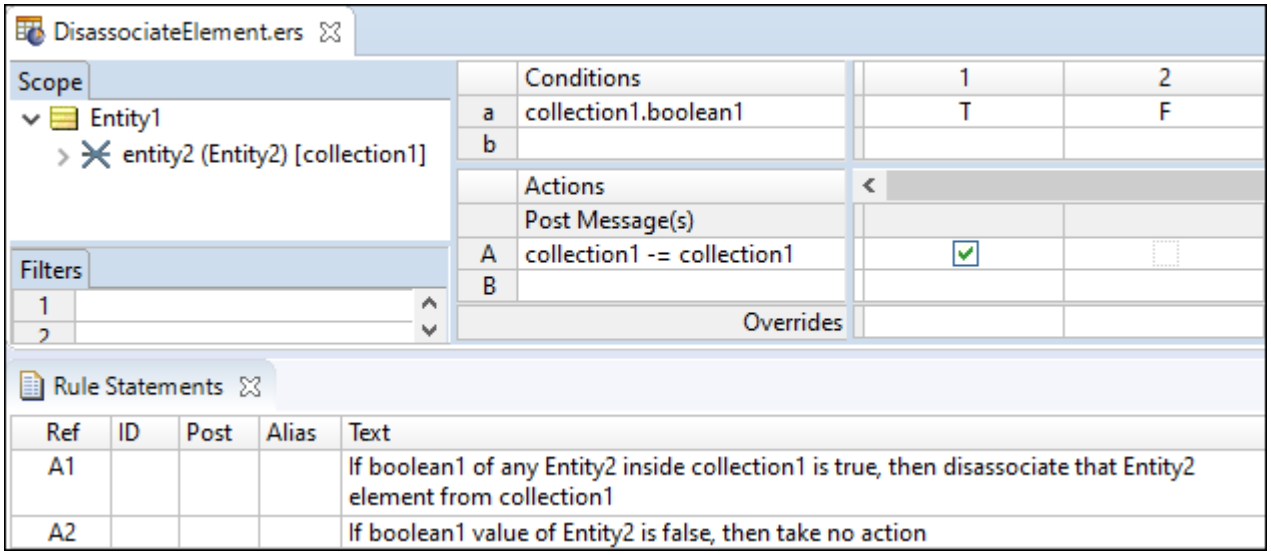
Disassociates all elements of `<Collection2>` from `<Collection1>`. Elements are not deleted, but once disassociated from `<Collection1>`, they are moved to the root level of the data. `<Collection1>` must be expressed as a unique alias. Contrast this behavior with remove, which deletes elements entirely.

USAGE RESTRICTIONS

The Operators row of the table in Vocabulary usage restrictions does not apply. Special exceptions: **disassociate element** may only be used in Action Rows (section 5 in Sections of Rulesheet that correlate with usage restrictions).

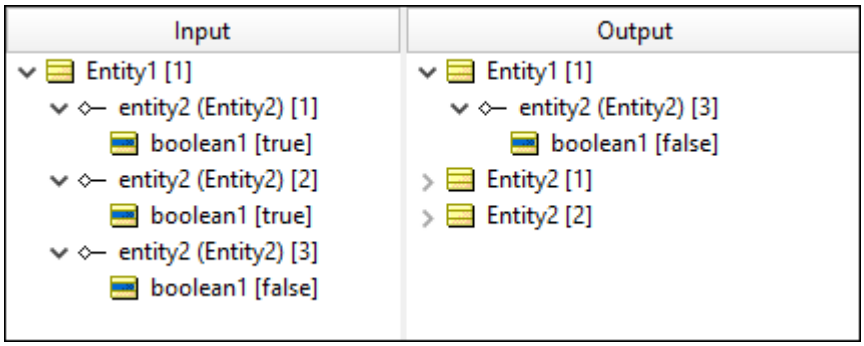
RULESHEET EXAMPLE

This sample Rulesheet removes those elements from `collection1` whose `boolean1` value is true.



SAMPLE RULETEST

A sample Ruletest provides a collection with three elements. The illustration shows Input and Output panels:



Divide

SYNTAX

`<Number1>/<Number2>`

DESCRIPTION

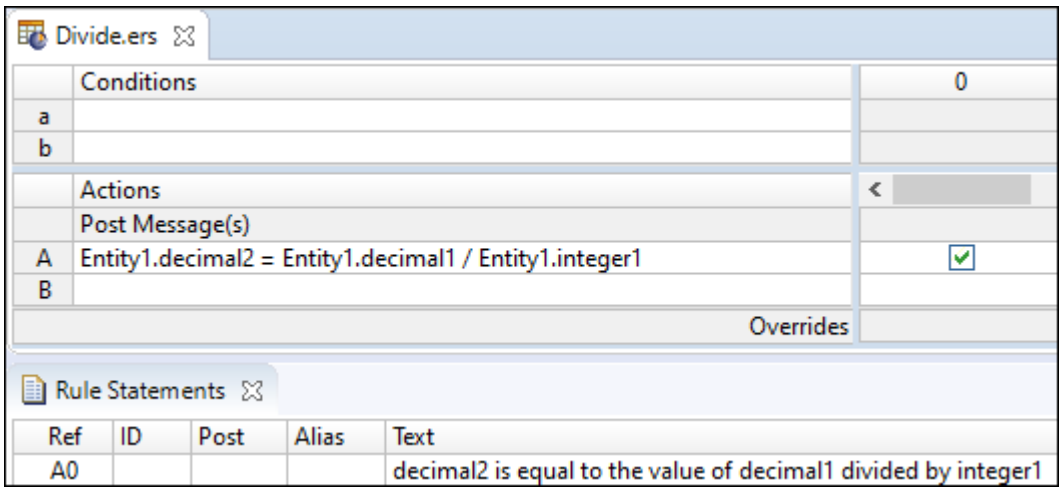
Divides `<Number1>` by `<Number2>`. The resulting data type is the more expansive of those of `<Number1>` and `<Number2>`.

USAGE RESTRICTIONS

The Operators row in the table of Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

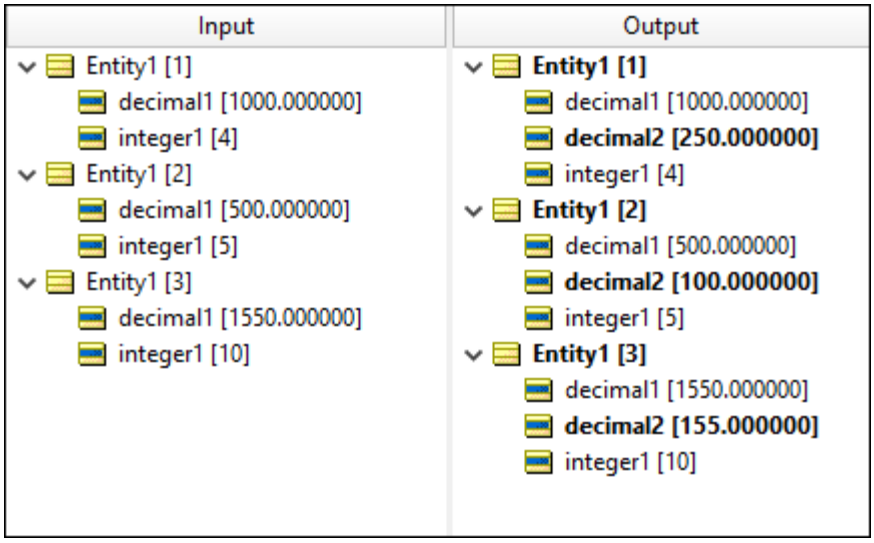
RULESHEET EXAMPLE

This sample Rulesheet uses **divide** to divide `decimal1` by `integer1` and assign the resulting value to `decimal2`



SAMPLE RULETEST

A sample Ruletest provides `decimal1` and `integer1` values for three examples. Input and Output panels are shown below.



Ends with

SYNTAX

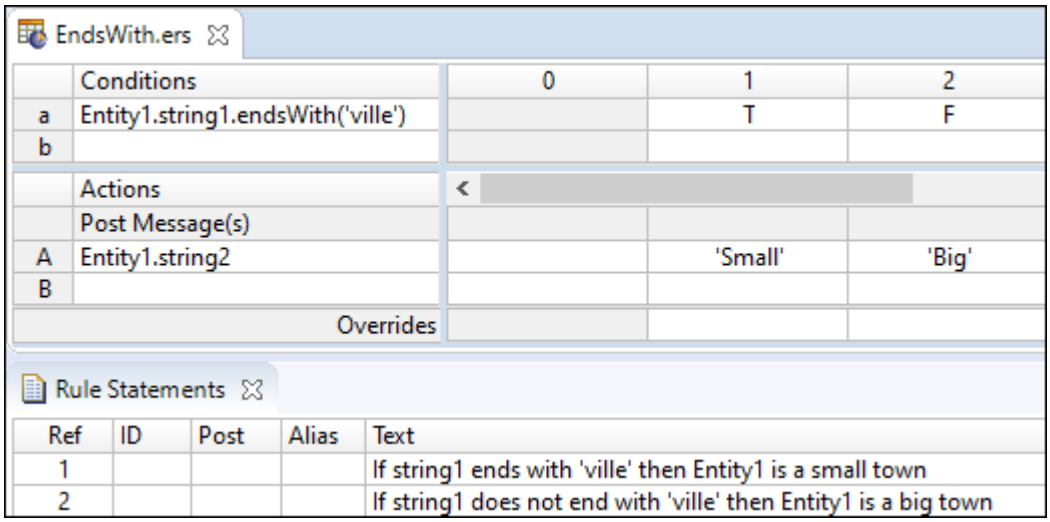
`<String1>.endsWith(<String2>)`

DESCRIPTION
Evaluates <String1> and returns a value of true if it ends with the characters specified in <String2>.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE
The following Rulesheet uses **.endsWith** to evaluate whether `string1` ends with the characters `ville` and assigns a different value to `string2` for each outcome.



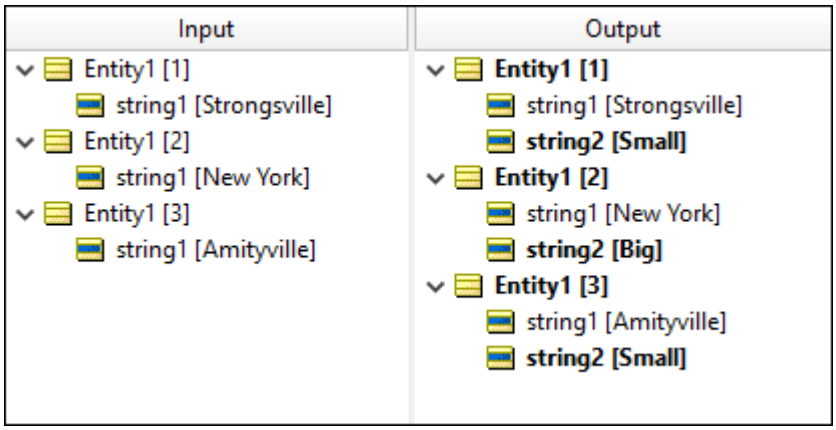
EndsWith.ers

	Conditions	0	1	2
a	Entity1.string1.endsWith('ville')		T	F
b				
	Actions			
	Post Message(s)			
A	Entity1.string2		'Small'	'Big'
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1				If string1 ends with 'ville' then Entity1 is a small town
2				If string1 does not end with 'ville' then Entity1 is a big town

SAMPLE RULETEST
A sample Ruletest provides `string1` values for three examples. Input and Output panels are shown below.



Input	Output
Entity1 [1]	**Entity1 [1]**
string1 [Strongsville]	string1 [Strongsville]
Entity1 [2]	**string2 [Small]**
string1 [New York]	**Entity1 [2]**
Entity1 [3]	string1 [New York]
string1 [Amityville]	**string2 [Big]**
	Entity1 [3]
	string1 [Amityville]
	string2 [Small]

Equals ignoring case

SYNTAX
<String1>.equalsIgnoreCase(<String2>)

DESCRIPTION

Returns a value of true if <String1> is the same as <String2>, irrespective of case.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

This sample Rulesheet uses **.equalsIgnoreCase** to compare the values of `string1` and `string2`, and assign a value to `boolean1` based on the results of the comparison.

EqualsIgnoringCase.ers

	Conditions	0	1	2
a	Entity1.string1.equalsIgnoreCase(Entity1.string2)		T	F
b				
	Actions			
	Post Message(s)			
A	Entity1.boolean1		T	F
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1				boolean1 must be true if string1 and string2 are the same (ignoring case)
2				boolean1 must be false if string1 and string2 are not the same (ignoring case)

SAMPLE RULETEST

A sample Ruletest provides the plane type for three sets of `string1` and `string2`. Input and Output panels are shown below. Notice how these results differ from those shown in the `equals` example.

Input	Output
Entity1 [1]	**Entity1 [1]**
string1 [McDonnell-Douglas]	**boolean1 [true]**
string2 [McDONNell-DOUGlas]	string1 [McDonnell-Douglas]
Entity1 [2]	string2 [McDONNell-DOUGlas]
string1 [LOCKHEED]	**Entity1 [2]**
string2 [lockheed]	**boolean1 [true]**
Entity1 [3]	string1 [LOCKHEED]
string1 [boeing]	string2 [lockheed]
string2 [boing]	**Entity1 [3]**
	boolean1 [false]
	string1 [boeing]
	string2 [boing]

Equals when used as an assignment

SYNTAX

Boolean	`<Boolean1>` = `<Expression1>`
DateTime*	`<DateTime1>` = `<DateTime2>`
Number	`<Number1>` = `<Number2>`
String	`<String1>` = `<String2>`

DESCRIPTION

Boolean	Assigns the truth value of `<Expression1>` to `<Boolean1>`.
DateTime*	Assigns the value of `<DateTime2>` to `<DateTime1>`.
Number	Assigns the value of `<Number2>` to `<Number1>`. Automatic *casting* (the process of changing a value's data type) will occur when assigning an Integer data type to a Decimal data type. To assign a Decimal value to an Integer value, use the .toInteger operator.
String	Assigns the value of `<String2>` to `<String1>`.

* includes DateTime and Date data types.

USAGE RESTRICTIONS

The Operators row of the table in Vocabulary usage restrictions does not apply. Special exceptions: **equals** used as an assignment may only be used in Action Rows (section 5 in Sections of Rulesheet that correlate with usage restrictions).

RULESHEET EXAMPLE

The following Rulesheet uses **equals** twice: in an Action row to assign a value to `decimal1`, and in an Action row to assign a value to `string1` based on the value of `boolean1`.

EqualsUsedAsAnAssignment.ers

	Conditions	0	1	2
a	Entity1.boolean1		T	F
b				
	Actions			
	Post Message(s)			
A	Entity1.decimal1 = 5.0		☑	
B	Entity1.string1 = 'yes'		☑	
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
A0				decimal1 is assigned a value of 5
B0				If boolean1 is true, assign the value of [yes] to string1
2				If boolean1 is false, then take no action

SAMPLE RULETEST

A sample Ruletest provides two examples of `boolean1`. Input and Output panels are shown below:

Input	Output
Entity1 [1]	**Entity1 [1]**
boolean1 [true]	boolean1 [true]
Entity1 [2]	**decimal1 [5.000000]**
boolean1 [false]	**string1 [yes]**
	Entity1 [2]
	boolean1 [false]

Equals when used as a comparison

SYNTAX

Boolean	`<Expression1> = <Expression2>`
DateTime	`<DateTime1> = <DateTime2>`
Number	`<Number1> = <Number2>`
String	`<String1> = <String2>`

DESCRIPTION

Boolean	Returns a value of true if <Expression1> is the same as <Expression2>.
DateTime	Returns a value of true if <DateTime1> is the same as <DateTime2>, including both the Date and the Time portions
Number	Returns a value of true if <Number1> is the same as <Number2>. Different numeric data types may be compared in the same expression.
String	Returns a value of true if <String1> is the same as <String2>. Both case and length are examined to determine equality. Corticon.js Studio uses the ISO character precedence in comparing String values.

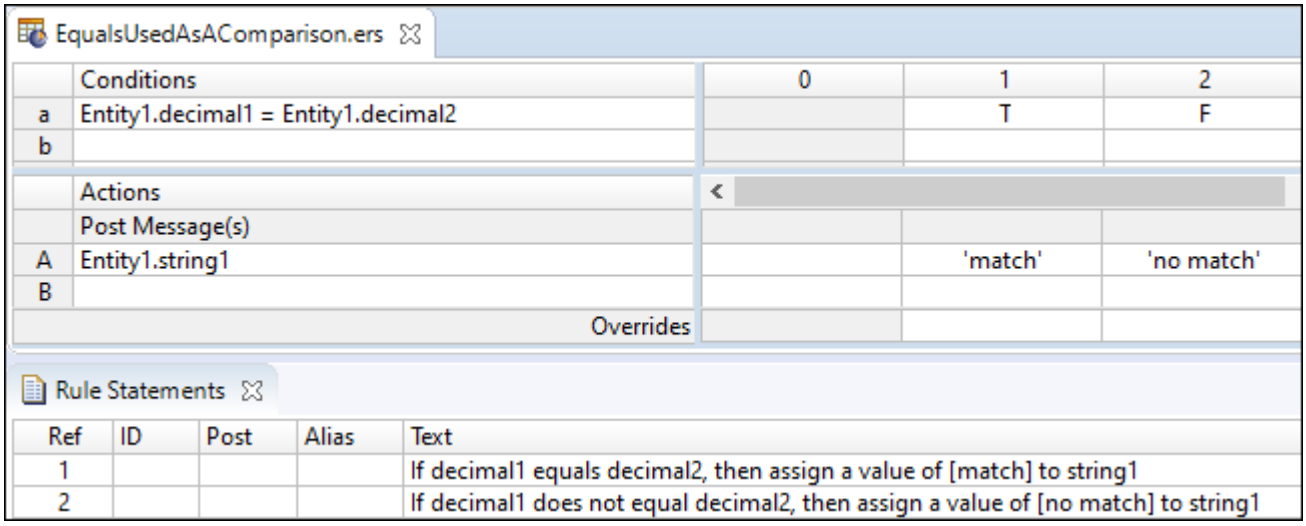
*includes DateTime and Date data types

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **equals** to Ruletest whether `decimal1` equals `decimal2`, and assign a value to `string1` based on the result of the comparison.



EqualsUsedAsAComparison.ers

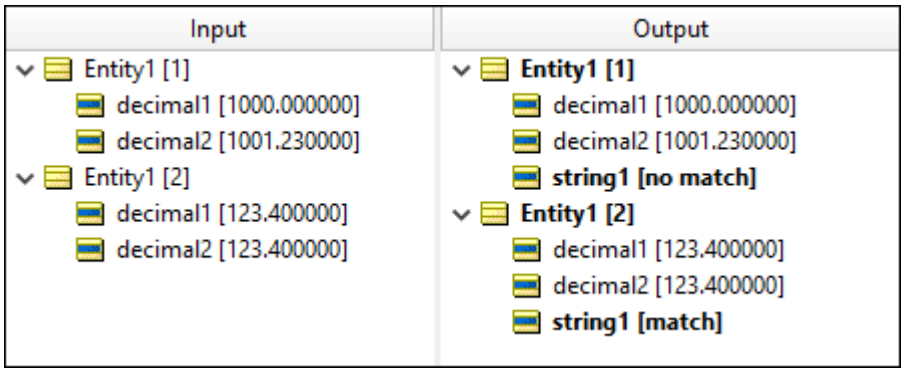
	Conditions	0	1	2
a	Entity1.decimal1 = Entity1.decimal2		T	F
b				
	Actions			
	Post Message(s)			
A	Entity1.string1		'match'	'no match'
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1				If decimal1 equals decimal2, then assign a value of [match] to string1
2				If decimal1 does not equal decimal2, then assign a value of [no match] to string1

SAMPLE RULETEST

A sample Ruletest provides two examples. Input and Output panels are shown below:



Equals when using Strings

SYNTAX

`<String1>.equals(<String2>)`

DESCRIPTION

Returns a value of true if `<String1String2>`, including character case. This is alternative syntax to > is exactly the same as <equals (used as a comparison).

USAGE RESTRICTIONS

The Operators row in the table Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

This sample Rulesheet uses **.equals** to compare the contents of `string1` and `string2`, and assign a value to `boolean1` as a result.

EqualsStringsOnly.ers

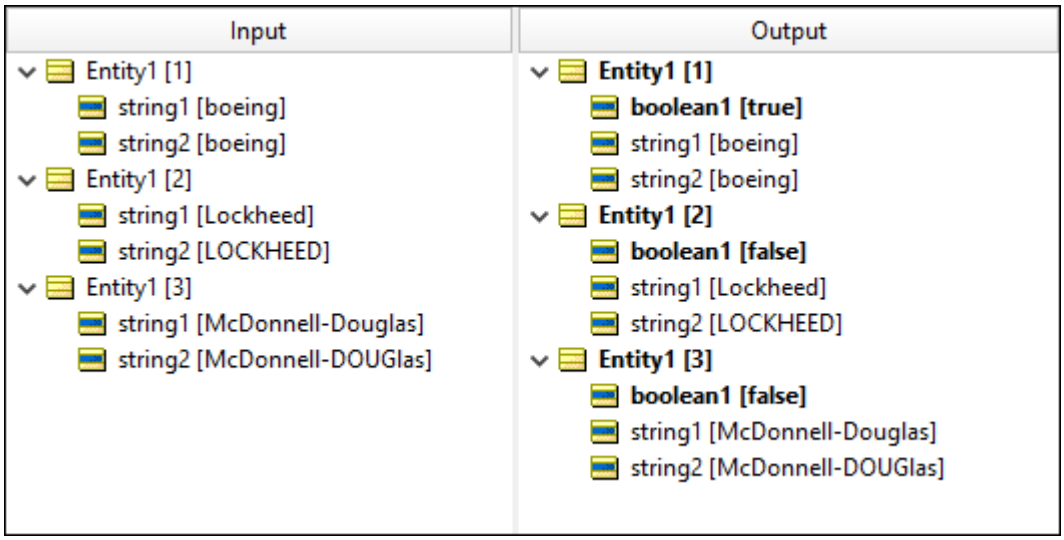
	Conditions	0	1	2
a	Entity1.string1.equals(Entity1.string2)		T	F
b				
	Actions			
	Post Message(s)			
A	Entity1.boolean1		T	F
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1				boolean1 must be true if string1 and string2 are the same
2				boolean1 must be false if string1 and string2 are not the same

SAMPLE RULETEST

>A sample Ruletest provides three sets of `string1` and `string2`. Input and Output panels are shown below. Notice how these results differ from those shown in the .equalsIgnoreCase example.



Exists

SYNTAX

`<Collection>` ->exists(`<Expression1>`,`<Expression2>`,…)

`<Collection>` ->exists(`<Expression1>` or `<Expression2>` or …)

DESCRIPTION

Returns a value of true if `<Expression>` holds true for *at least one* element of `<Collection>`. `<Collection>` must be expressed as a unique alias. Multiple `<Expressions>` are optional, but at least one is required.

Both **AND** (indicated by commas between `<Expressions>`) and **OR** syntax (indicated by `or` between `<Expressions>`) are supported within the parentheses `(..)`. However, take care to ensure invariant expressions are not inadvertently created. For example:

```
<Collection> -> exists(integer1=5, integer1=8)
```

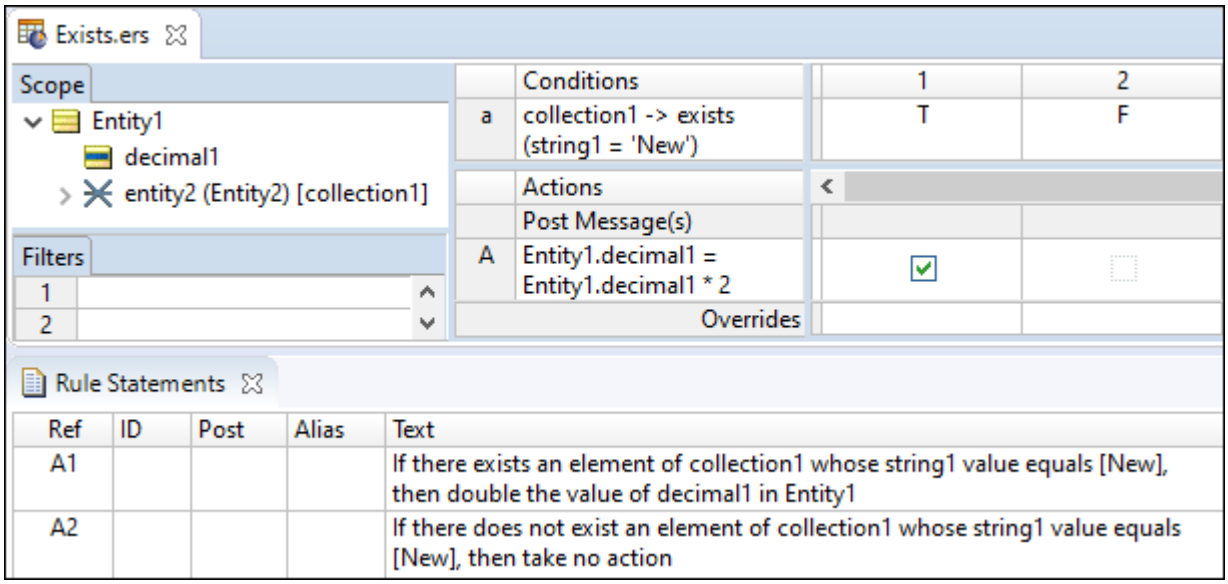
will always evaluate to `false` because no `integer1` value can be both `5` **AND** `8` simultaneously.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

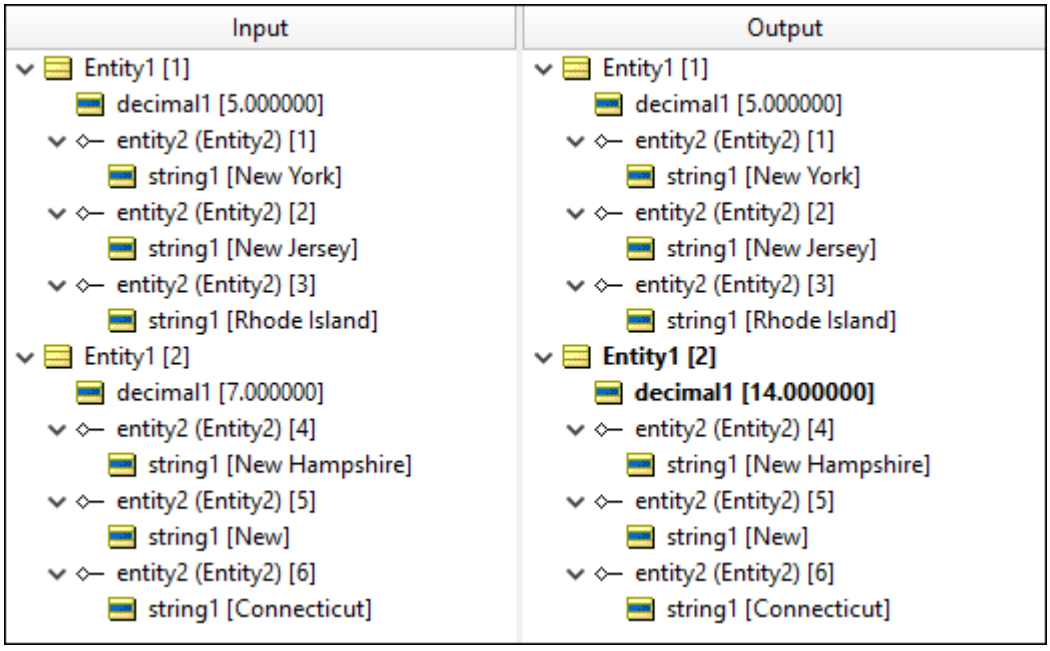
RULESHEET EXAMPLE

This sample Rulesheet uses **->exists** to check for the existence of an element in `collection1` whose `string1` value equals `New`, and assigns a value to `decimal1` based on the results of the test. Note the use of unique alias `collection1` to represent the collection of `Entity2` associated with `Entity1`.



SAMPLE RULETEST

A sample Ruletest provides 2 separate collections of `Entity2` elements and `Entity1.decimal1` values. Input and Output panels are shown below.



Exponent

SYNTAX

`<Number1>` ** `<Number2>`

DESCRIPTION

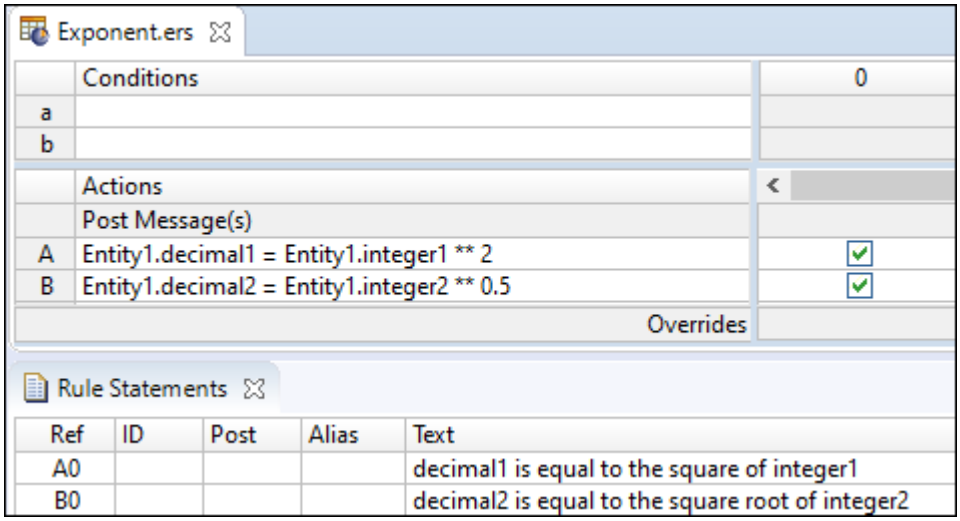
Raises <Number1> by the power of <Number2>. The resulting data type is the more expansive of those of <Number1> and <Number2>. To find a root, <Number2> can be expressed as a decimal value, such as 0.5 for a square root, or -- for greater accuracy in larger roots -- in decimal format within parentheses, such as **(1.0/3.0) for a cube root.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

This sample Rulesheet uses **exponent** to raise integer1 and integer2 by the power of 2 and 0.5, respectively, and assign the resulting value to decimal1 and decimal2, respectively.



Exponent.ers

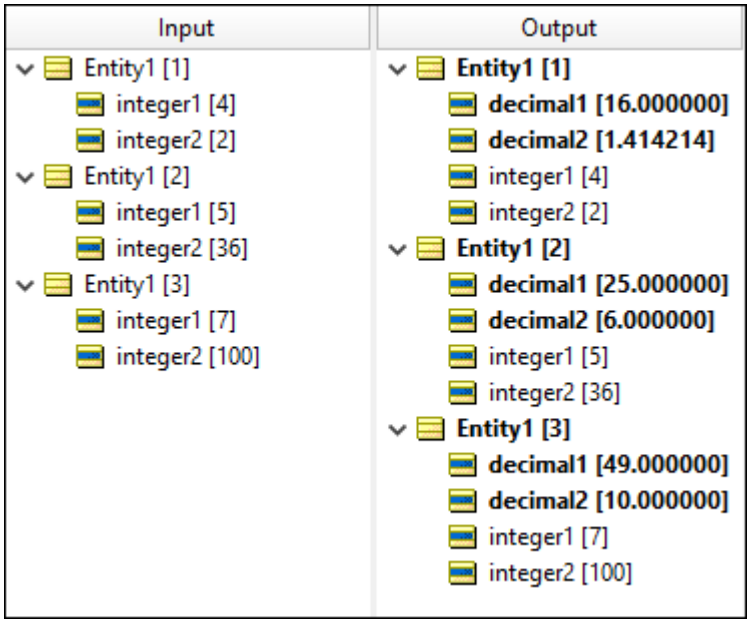
	Conditions	0
a		
b		
	Actions	
	Post Message(s)	
A	Entity1.decimal1 = Entity1.integer1 ** 2	☑
B	Entity1.decimal2 = Entity1.integer2 ** 0.5	☑
	Overrides	

Rule Statements

Ref	ID	Post	Alias	Text
A0				decimal1 is equal to the square of integer1
B0				decimal2 is equal to the square root of integer2

SAMPLE RULETEST

A sample Ruletest provides decimal1 and integer1 values for three examples.



Input	Output
Entity1 [1]	**Entity1 [1]**
integer1 [4]	**decimal1 [16.000000]**
integer2 [2]	**decimal2 [1.414214]**
Entity1 [2]	integer1 [4]
integer1 [5]	integer2 [2]
integer2 [36]	**Entity1 [2]**
Entity1 [3]	**decimal1 [25.000000]**
integer1 [7]	**decimal2 [6.000000]**
integer2 [100]	integer1 [5]
	integer2 [36]
	Entity1 [3]
	decimal1 [49.000000]
	decimal2 [10.000000]
	integer1 [7]
	integer2 [100]

False

SYNTAX

false or F

DESCRIPTION

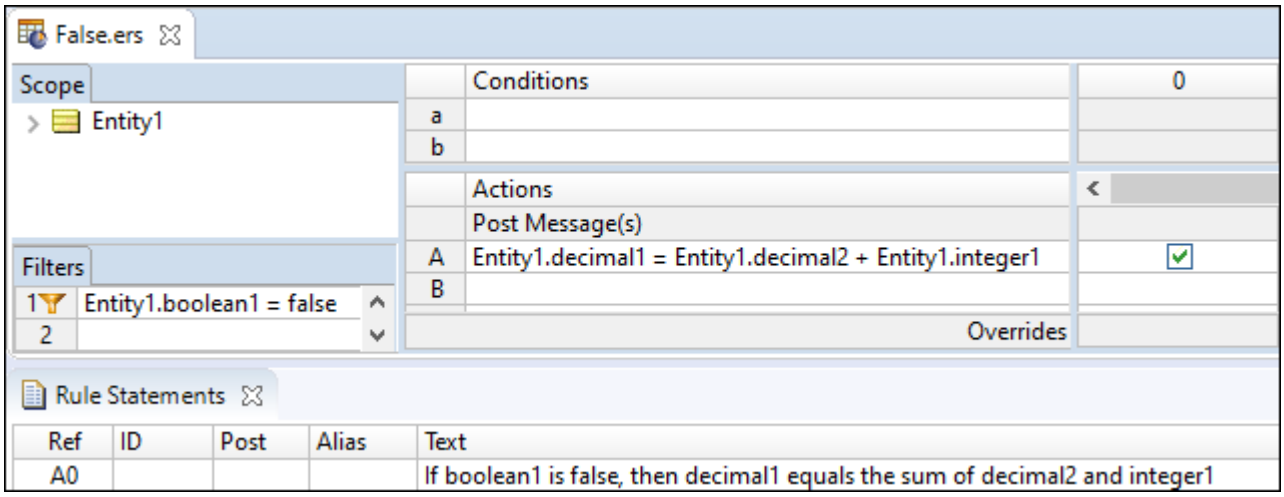
Represents the Boolean value false. Recall from discussion of truth values that an <expression> is evaluated for its truth value, so the expression `Entity1.boolean1=false` evaluates to `true` only when `boolean1=false`. But since `boolean1` is Boolean and has a truth value all by itself without any additional syntax, we could simply state `not Entity1.boolean1`, with the same effect. Many examples in the documentation use explicit syntax like `boolean1=true` or `boolean2=false` for clarity and consistency, even though `boolean1` or `not boolean2` are equivalent, respectively, to the explicit syntax.

USAGE RESTRICTIONS

The Operators row of the table in Vocabulary usage restrictions applies. No special exceptions.

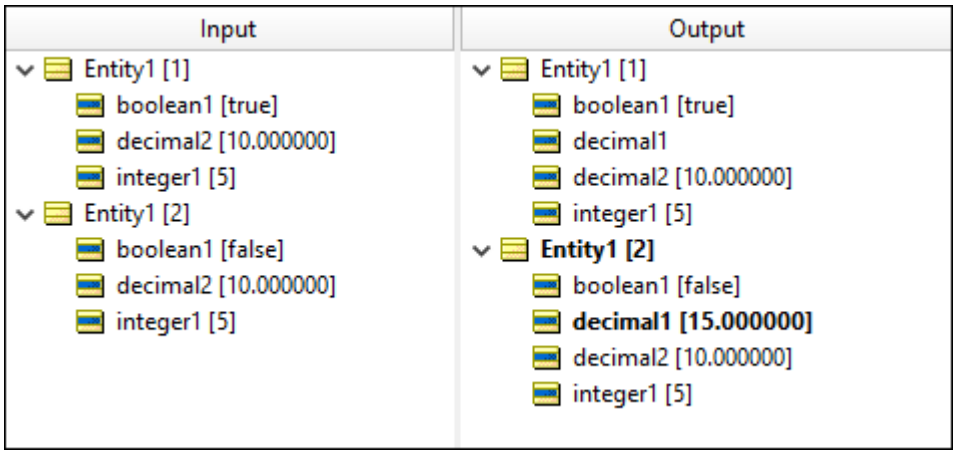
RULESHEET EXAMPLE

The following Rulesheet uses **false** in a Filter row to test whether `boolean1` is false, and perform the Nonconditional computation if it is. As discussed above, the alternative expression `not Entity1.boolean1` is logically equivalent.



SAMPLE RULETEST

A sample Ruletest provides three examples. Assume `decimal2=10.0` and `integer1=5` for all examples. Input and Output panels are shown below:



First

SYNTAX

<Sequence> ->first.<attribute1>

DESCRIPTION

Returns the value of <attribute1> of the first element in <Sequence>. Another operator, such as ->sortedBy, must be used to transform a <Collection> into a <Sequence> before **->first** may be used. <Sequence> must be expressed as a unique alias. See *"Advanced collection sorting syntax" in the Rule Modeling Guide* for more examples of usage.

<attribute1> may be of any data type.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

This sample Rulesheet uses **->first** to identify the first element of the sequence created by applying ->sortedBy to collection1. Once identified, the value of the string1 attribute belonging to this first element is evaluated. If the value of string1 is Joe, then boolean1 attribute of Entity1 is assigned the value of true.

First.ers

Scope

- Entity1
  - boolean1
  - entity2 (Entity2) [collection1]
    - decimal1
    - string1

Filters

1	
2	

	Conditions	1	2
a	collection1.string1 -> sortedBy (decimal1) -> first	'Joe'	not 'Joe'
b			
	Actions		
	Post Message(s)		
A	Entity1.boolean1	T	F
B			
C			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If the string1 value of the first element in collection1, in ascending order by decimal1, is equal to Joe, then boolean1 is true.
2				If the string1 value of the first element in collection1, in ascending order by decimal1, is NOT equal to Joe, then boolean1 is false.

SAMPLE RULETEST

A sample Ruletest provides a collection of three elements, each with a `decimal1` value. Input and Output panels are shown below.

Input	Output
Entity1 [1]	**Entity1 [1]**
entity2 (Entity2) [1]	**boolean1 [true]**
decimal1 [2.500000]	entity2 (Entity2) [1]
string1 [Joe]	decimal1 [2.500000]
entity2 (Entity2) [2]	string1 [Joe]
decimal1 [5.800000]	entity2 (Entity2) [2]
string1 [Mary]	decimal1 [5.800000]
entity2 (Entity2) [3]	string1 [Mary]
decimal1 [3.300000]	entity2 (Entity2) [3]
string1 [Sue]	decimal1 [3.300000]
	string1 [Sue]

Floor

SYNTAX

<Decimal>.floor

DESCRIPTION

Returns the Decimal closest to zero from <Decimal>. **.floor** may also be thought of as a truncation of <Decimal>.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The Rulesheet uses **.floor** to assign decimal values to `decimal` that are closer to zero than the input `decimal2` values.

floor.ers

	Conditions	0	1
a			
b			
	Actions		
	Post Message(s)		
A	Entity1.integer1=Entity1.decimal2.floor	☑	☐
B			
C			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
A0				Integer1 is equal to the highest integer value that does not exceed the decimal value of decimal1

SAMPLE RULETEST

A sample Ruletest provides three `decimal2` values. Input and Output panels are shown below:

/Generic.js/floor.ers

Input	Output
Entity1 [1]	**Entity1 [1]**
decimal2 [1550.785000]	**decimal1 [1550.000000]**
Entity1 [2]	decimal2 [1550.785000]
decimal2 [2200.986000]	**Entity1 [2]**
Entity1 [3]	**decimal1 [2200.000000]**
decimal2 [-500.999000]	decimal2 [2200.986000]
	Entity1 [3]
	decimal1 [-501.000000]
	decimal2 [-500.999000]

Note: Notice how these results differ from those shown in the Round example.

For all

SYNTAX

<Collection> ->forAll(<Expression1>, <Expression2>,…)

`<Collection>` ->forAll(`<Expression1>` or `<Expression2>` or …)

DESCRIPTION

Returns a value of true if *every* `<Expression>` holds true for *every* element of `<Collection>`. `<Collection>` must be expressed as a unique alias. Multiple `<Expressions>` are optional, but at least one is required.

Both **AND** (indicated by commas between `<Expressions>`) and **OR** syntax (indicated by `or` between `<Expressions>`) is supported within the parentheses `(..)`. However, take care to ensure invariant expressions are not inadvertently created. For example:

```
<Collection> -> forAll(integer1=5, integer1=8)
```

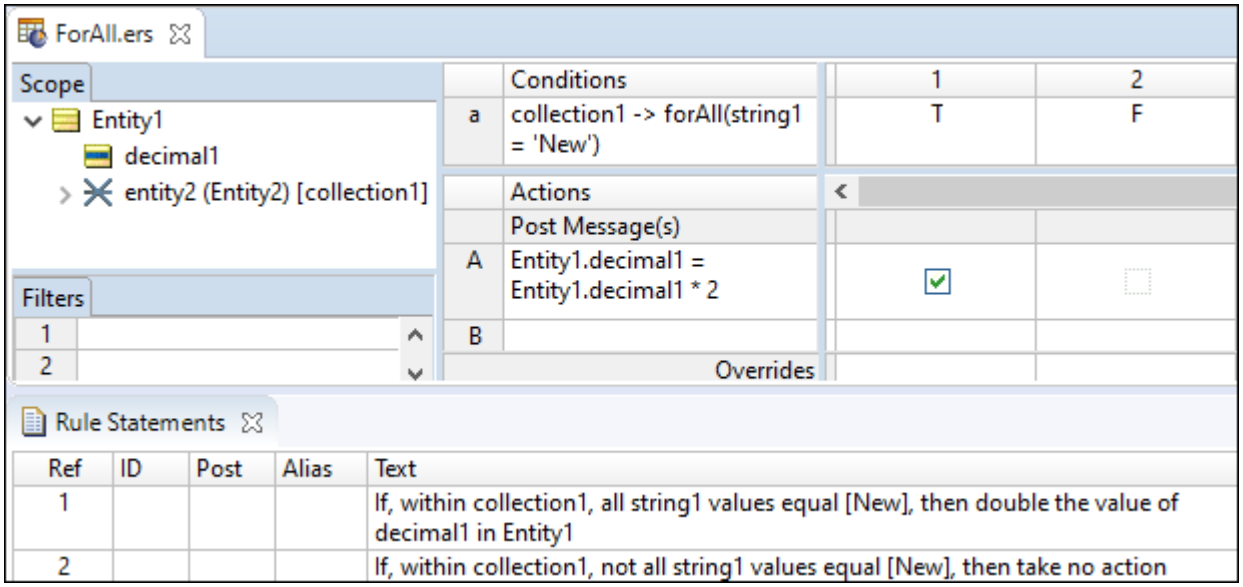
will always evaluate to `false` because no single `integer1` value can be both `5` **AND** `8` simultaneously, let alone all of them.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

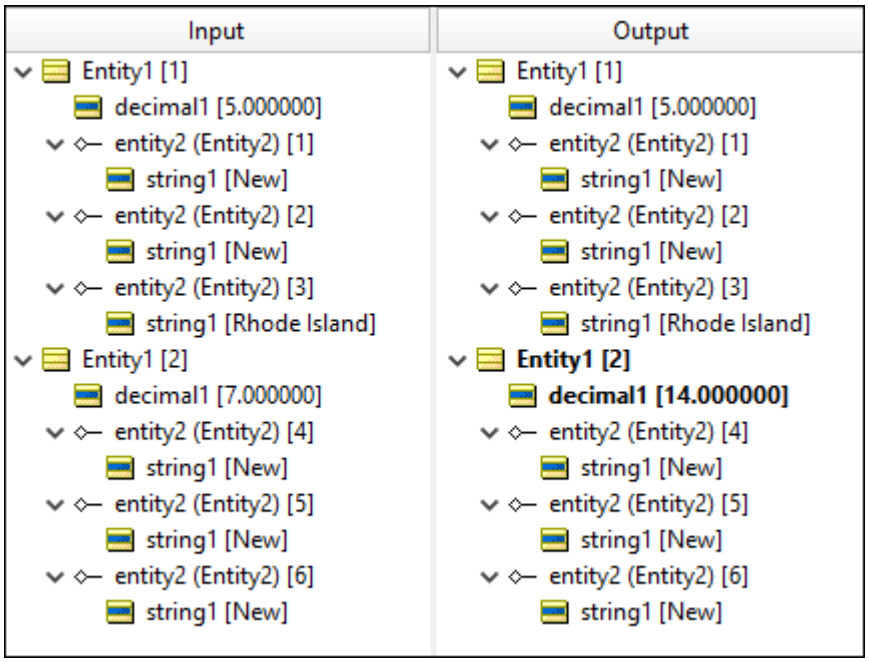
RULESHEET EXAMPLE

This sample Rulesheet uses **->forAll** to check for the existence of an element in `collection1` whose `string1` value equals `New`, and assigns a value to `decimal1` based on the results of the test. Note the use of unique alias `collection1` to represent the collection of `Entity2` associated with `Entity1`.



SAMPLE RULETEST

A sample Ruletest provides 2 separate collections of `Entity2` elements and `Entity1.decimal1 values`. The following illustration shows Input and Output panel



Get Milliseconds

SYNTAX

`<DateTime>`.getMilliseconds

DESCRIPTION

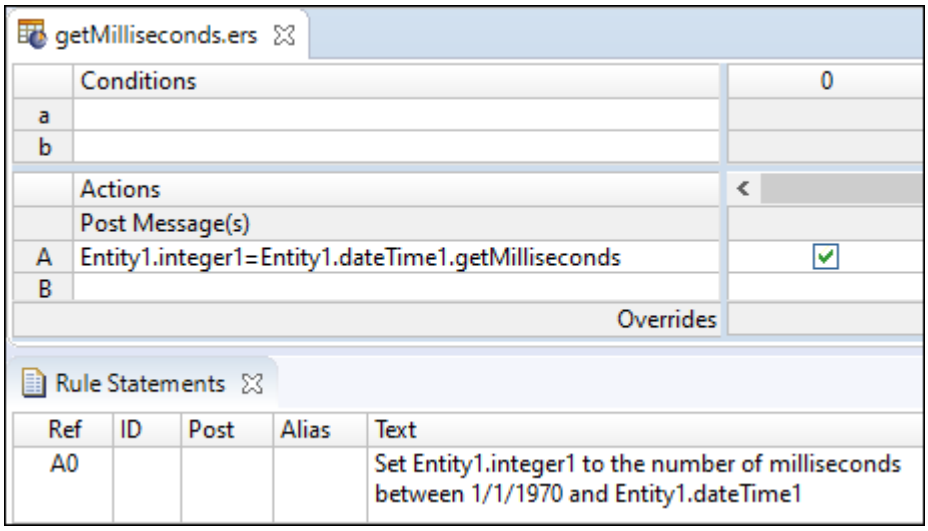
Returns the number of milliseconds elapsed since the epoch: January 1, 1970.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

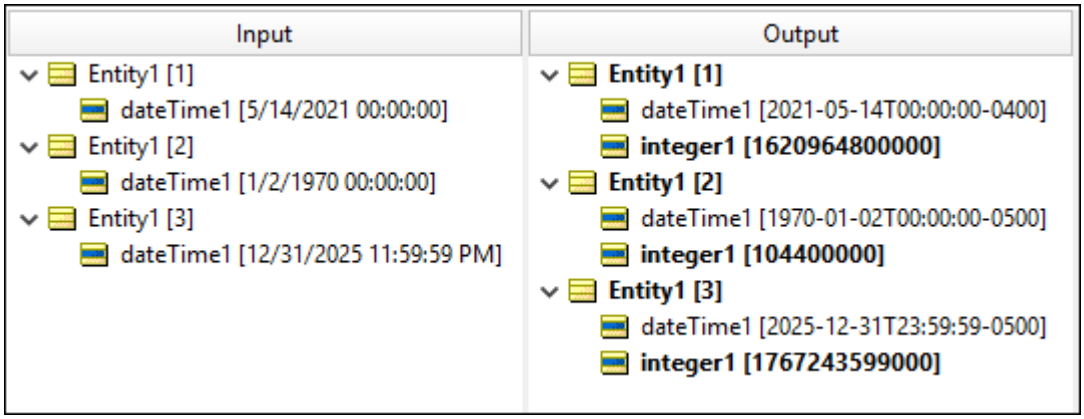
RULESHEET EXAMPLE

This sample Rulesheet uses **.getMilliSeconds** in a Nonconditional rule to evaluate the number of milliseconds between the epoch and `dateTime1`, and return the number as `integer1`.



SAMPLE RULETEST

A sample Ruletest provides values of `dateTime2` for three instances of `Entity1`. Input and Output panels are shown below.



Greater than

SYNTAX

DateTime*	`<DateTime1> > <DateTime2>`
Number	`<Number1> > <Number2>`
String	`<String1>  > <String2>`

DESCRIPTION

DateTime*	Returns a value of true if <DateTime1> is greater than or equal to <DateTime2>. This is equivalent to <DateTime1> occurring after <DateTime2>
Number	Returns a value of true if <Number1> is greater than <Number2>. Different numeric data types may be compared in the same expression.
String	Returns a value of true if <String1> is greater than <String2>.

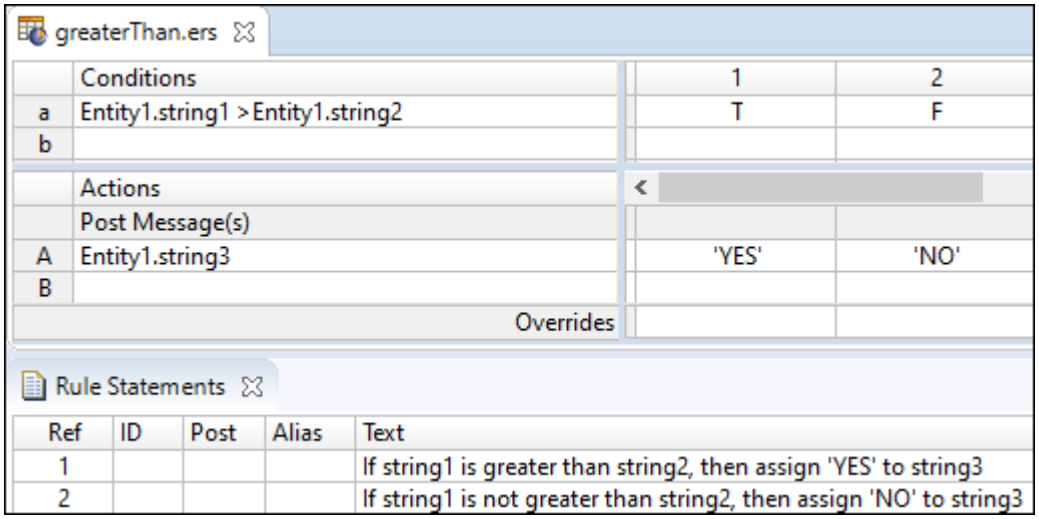
*includes DateTime and Date data types

USAGE RESTRICTIONS

The Operators row of the table in Vocabulary usage restrictions applies, with the following exception: **greater than** may also be used in Conditional Value Sets & Cells (section 5 in Sections of Rulesheet that correlate with usage restrictions).

RULESHEET EXAMPLE

The following Rulesheet uses **greater than** to test whether `string1` is greater than `string2`, and assigns YES or NO to `string3`.



greaterThan.ers

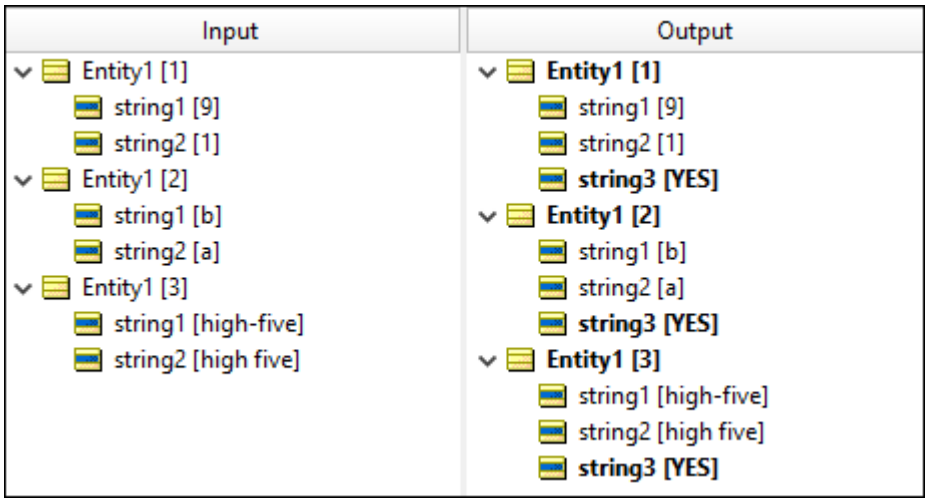
	Conditions	1	2
a	Entity1.string1 >Entity1.string2	T	F
b			
	Actions		
	Post Message(s)		
A	Entity1.string3	'YES'	'NO'
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If string1 is greater than string2, then assign 'YES' to string3
2				If string1 is not greater than string2, then assign 'NO' to string3

SAMPLE RULETEST

A sample Ruletest provides three examples. Input and Output panels are shown below:



Greater than or equal to

SYNTAX

DateTime*	`<DateTime1> >= <DateTime2>`
Number	`<Number1> >= <Number2>`
String	`<String1> >= <String2>`

DESCRIPTION

DateTime*	Returns a value of true if `<DateTime1>` is greater than or equal to `<DateTime2>`. This is equivalent to `<DateTime1>` occurring on or after `<DateTime2>`
Number	Returns a value of true if `<Number1>` is greater than or equal to `<Number2>`. Different numeric data types may be compared in the same expression.
String	Returns a value of true if `<String1>` is greater than or equal to `<String2>`.

*includes DateTime and Date data types

USAGE RESTRICTIONS

The Operators row of the table in Vocabulary usage restrictions applies, with the following exception: **greater than or equal to** may also be used in Conditional Value Sets & Cells (section 5 in Sections of Rulesheet that correlate with usage restrictions).

RULESHEET EXAMPLE

The following Rulesheet uses **greater than or equal to** to test whether `string1` is greater than or equal to `string2`, and assigns YES or NO to `string3`.

greaterThanEq.ers

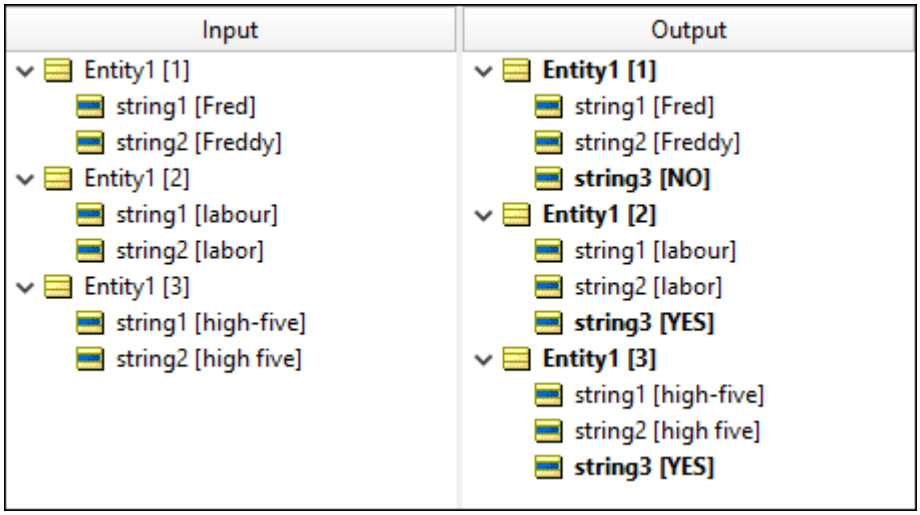
	Conditions	1	2
a	Entity1.string1 >= Entity1.string2	T	F
b			
	Actions		
	Post Message(s)		
A	Entity1.string3	'YES'	'NO'
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If string1 is greater than or equal to string2, then assign 'YES' to string3
2				If string1 is not greater than or equal to string2, then assign 'NO' to string3

SAMPLE RULETEST

A sample Ruletest provides two examples. Input and Output panels are shown below:



Hour

SYNTAX

`<DateTime>`.hour

DESCRIPTION

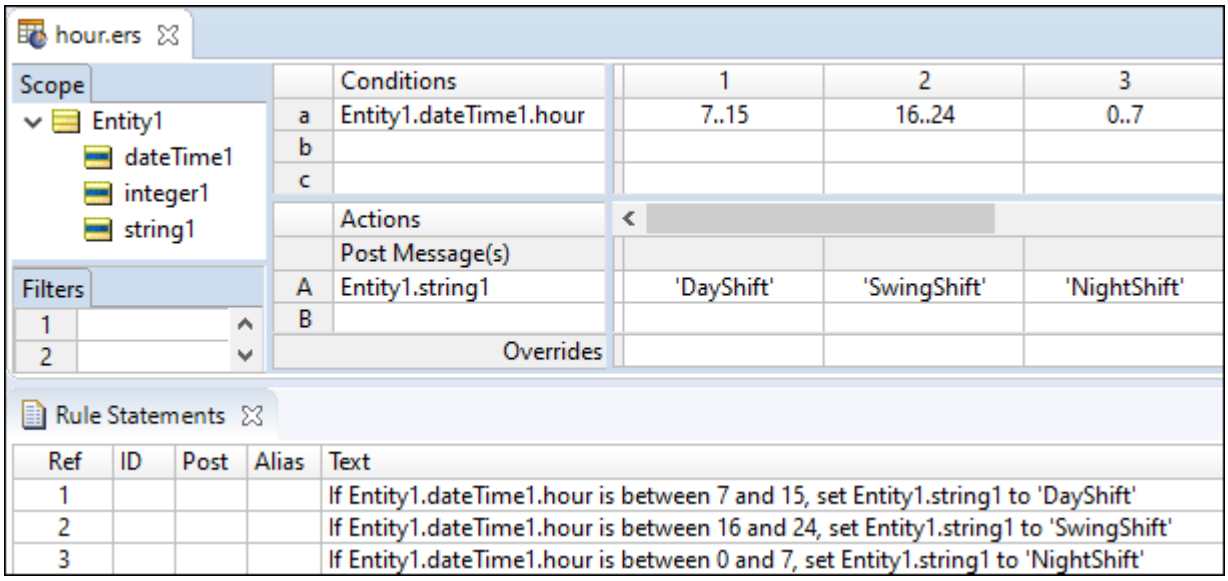
Returns the hour portion of `<DateTime>`. The returned value is based on a 24-hour clock. For example, 10:00 PM (22:00 hours) is returned as 22.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

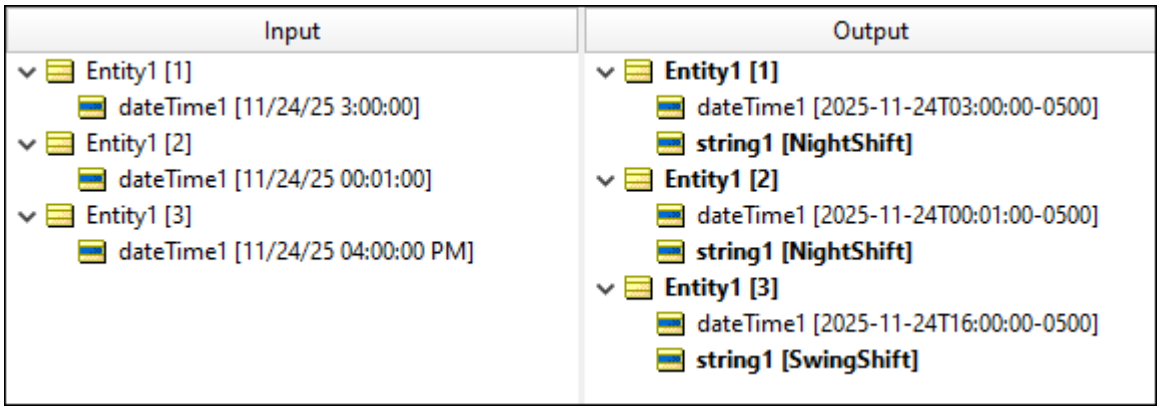
RULESHEET EXAMPLE

The following Rulesheet uses **.hour** to evaluate `dateTime1` and assign the hour value to `integer1`.



SAMPLE RULETEST

A sample Ruletest provides three examples of `dateTime1`. Input and Output panels are shown below. Notice that the hour returned is dependent upon the timezone of the machine executing the rule. The hour returned is independent of the machine running the Ruletest and only depends on the locale/timezone of the data itself.



Hours between

SYNTAX

`<DateTime1>.hoursBetween(<DateTime2>)`

DESCRIPTION

Returns the Integer number of hours between any two DateTimes. The function calculates the number of milliseconds between the two values and divides that number by 3,600,000 (the number of milliseconds in an hour). The decimal portion is then truncated. If the two dates differ by less than a full hour, the value is zero. This function returns a positive number if `<DateTime2>` is later than `<DateTime1>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.hoursBetween** to determine the number of hours that have elapsed between `dateTime1` and `dateTime2`, compare it to the Values set, and assign a value to `string1`.

HoursBetween.ers

	Conditions	1	2
a	Entity1.dateTime1.hoursBetween(Entity1.dateTime2)	<= 24	> 24
b			
	Actions		
	Post Message(s)		
A	Entity1.string1	'Not Overdue'	'Overdue'
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If 24 or fewer hours have elapsed between dateTime1 and dateTime2, then Entity1 is not overdue
2				If more than 24 hours have elapsed between dateTime1 and dateTime2, then Entity1 is not overdue

SAMPLE RULETEST

A sample Ruletest provides `dateTime1` and `dateTime2` for two examples. Input and Output panels are shown below.

Input	Output
Entity1 [1]	**Entity1 [1]**
dateTime1 [11/24/1960 00:00:00]	dateTime1 [1960-11-24T00:00:00-0500]
dateTime2 [12/15/1960 00:00:00]	dateTime2 [1960-12-15T00:00:00-0500]
Entity1 [2]	**string1 [Overdue]**
dateTime1 [11/24/1960 00:00:00]	**Entity1 [2]**
dateTime2 [12/15/2012 00:00:00]	dateTime1 [1960-11-24T00:00:00-0500]
Entity1 [3]	dateTime2 [2012-12-15T00:00:00-0500]
dateTime1 [11/24/2013 00:00:00]	**string1 [Overdue]**
dateTime2 [12/15/2022 00:00:00]	**Entity1 [3]**
	dateTime1 [2013-11-24T00:00:00-0500]
	dateTime2 [2022-12-15T00:00:00-0500]
	string1 [Overdue]

In LIST

SYNTAX

Date	`<Date1>` in {`<Date2>`,`<Date3>`,...}
DateTime	`<DateTime1>` in {`<DateTime2>`,`<DateTime3>`,...}
Decimal	`<Decimal1>` in {`<Decimal2>`,`<Decimal3>`,...}
Integer	`<Integer1>` in {`<Integer2>`,`<Integer3>`,...}
String	`<String1>` in {`<String2>`,`<String3>`,...}

DESCRIPTION

Returns the value `true` if the attribute type is contained in the set of valid values for the attribute.

USAGE RESTRICTIONS

- The set of values is always enclosed in braces: { }
- For integer and decimal data types, a list of literals or enumerated values without labels requires that the values are not in single quotes, such as `{3,1,2}`.
- For date and String data types, a list of literals or enumerated values without labels requires that the values are in single quotes, such as `{'B','A','C'}`.
- The list can be in any order.
- Duplicate values or labels in a list are tolerated.

When enumerated datatypes with labels are used:

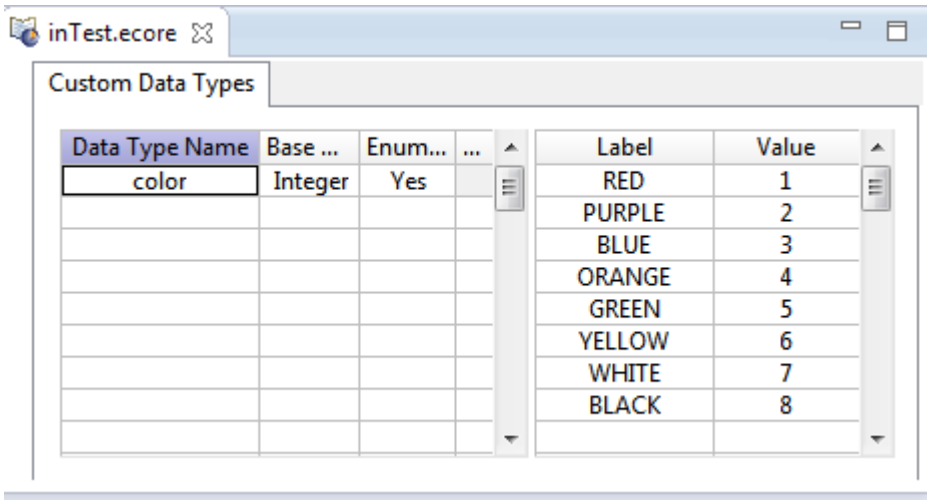
- The labels are listed without delimiters, such as `{B,A,C}`
- Values and labels can be mixed, such as `{A,B,'C_value'}`.

Note: While literal values in the enumeration table are accepted in a list, only existing label values will be exposed and accepted as valid.

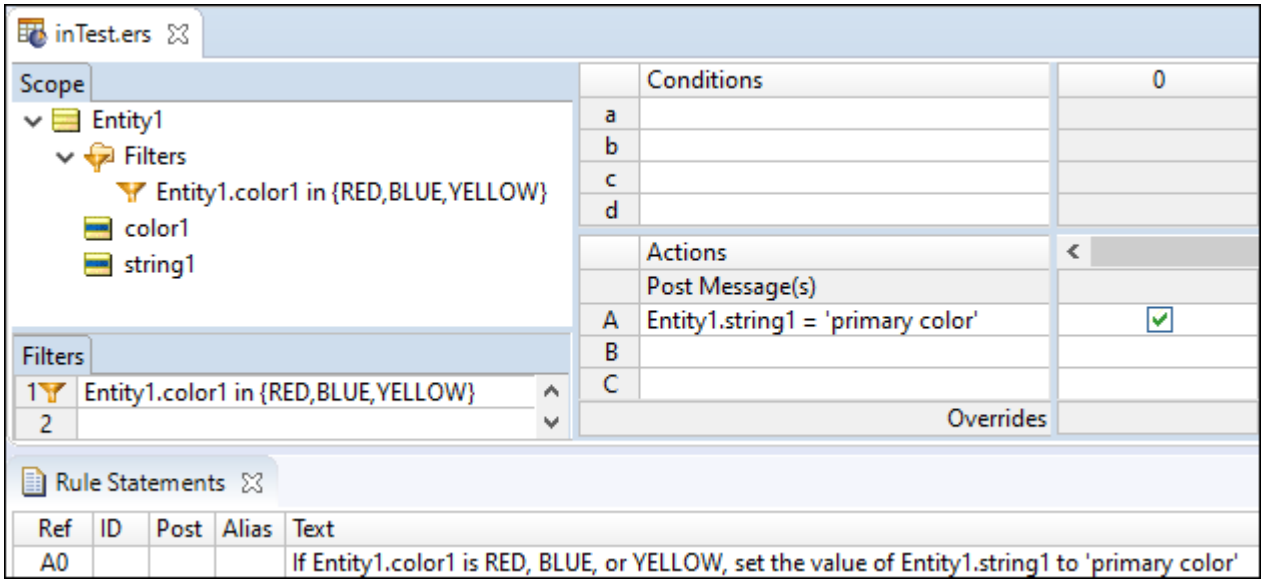
The Operators row of the table in Vocabulary Usage Restriction does not apply. The `in` operator can be used in Conditions and Filters, but not in Actions.

RULESHEET EXAMPLE

The example's Vocabulary defined an enumerated list:

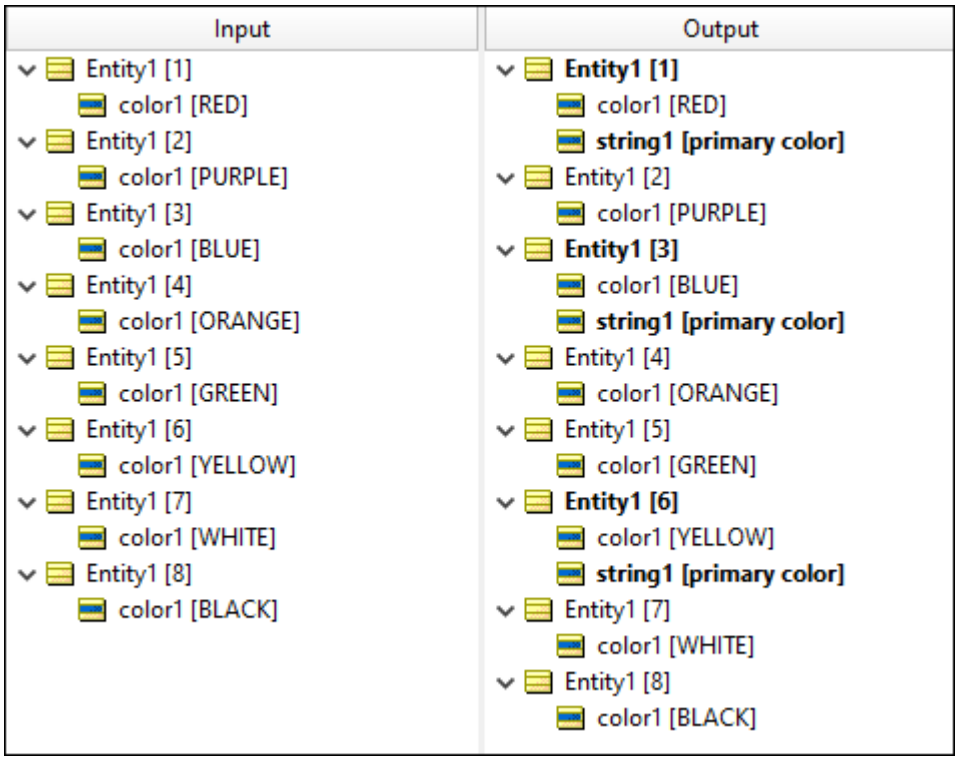


The following Rulesheet uses **in** to filter certain labels to be tested against request data:



SAMPLE TEST

A sample Ruletest provides examples. Input and Output panels are shown below.



In RANGE

SYNTAX

Date	`<Date1>` in `(<earlierDate2>..<laterDate3>)`
DateTime	`<DateTime1>` in `(<earlierDateTime2>..<laterDateTime3>)`
Decimal	`<Decimal1>` in `(<smallerDecimal2>..<largerDecimal3>)`
Integer	`<Integer1>` in `(<smallerInteger2>..<largerInteger3>)`
String	`<String1>` in `(<startString2>..<endString3>)`

A square bracket on either end of the expression indicates that the start or end value is to be included in the range.

DESCRIPTION

Returns the value `true` if the attribute type is contained in the range of valid values for the attribute.

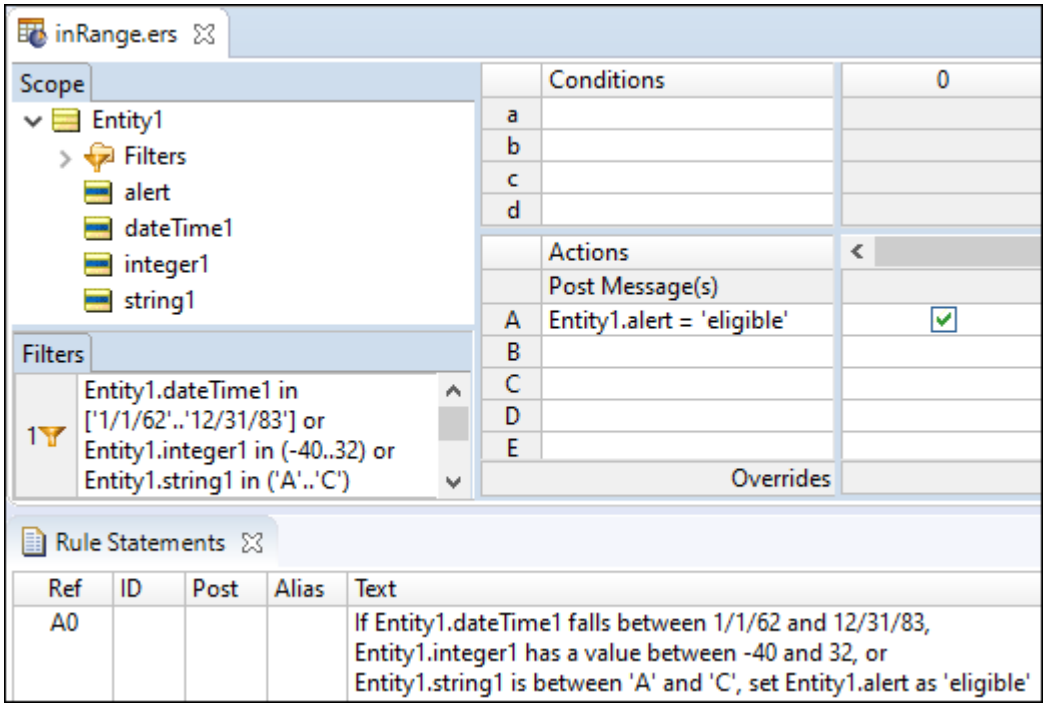
USAGE RESTRICTIONS

- For integer and decimal data types, the range of values are not in single quotes. For example, `(1..3)`.
- For date and String data types, the range of values are in single quotes. For example, `('A'..'C')`.

The Operators row of the table in Vocabulary Usage Restriction does not apply. The `in` operator can be used in Conditions and Filters, but not in Actions.

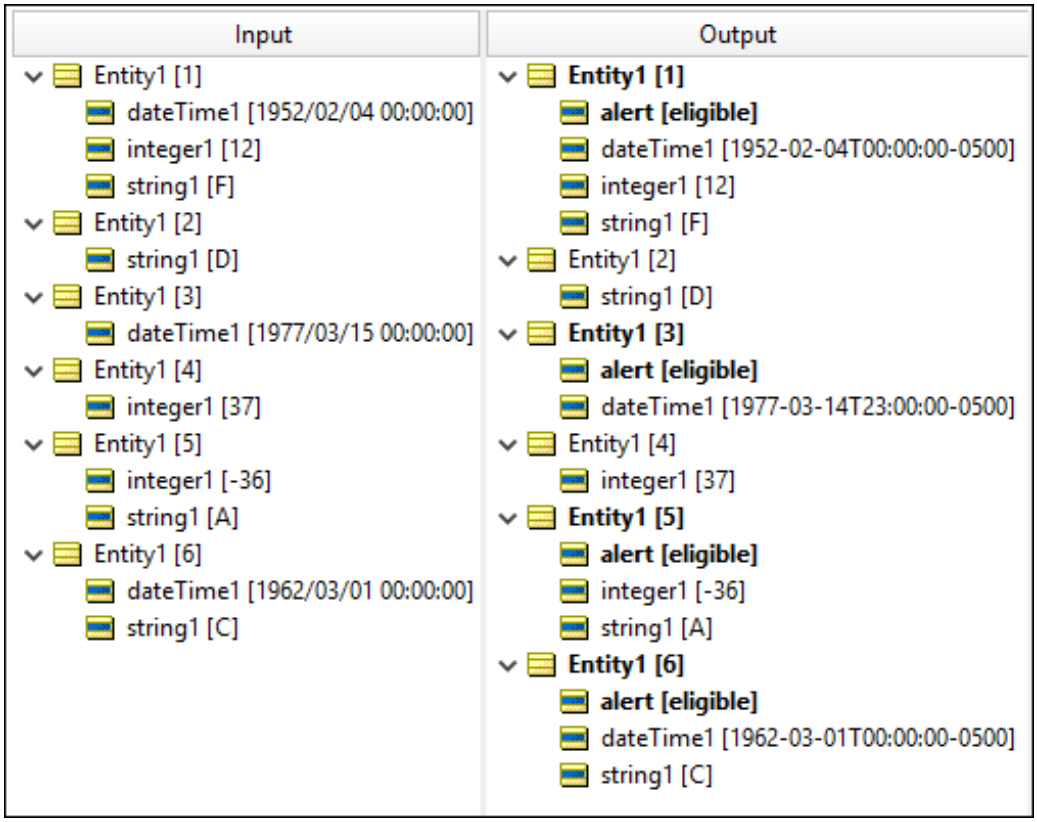
RULESHEET EXAMPLE

The following Rulesheet uses **in** ranges for three data types OR'ed together in a filter to be tested against request data:



SAMPLE TEST

A sample Ruletest provides examples. Input and Output panels are shown below.



Increment

SYNTAX

`<Number1> += <Number2>`

DESCRIPTION

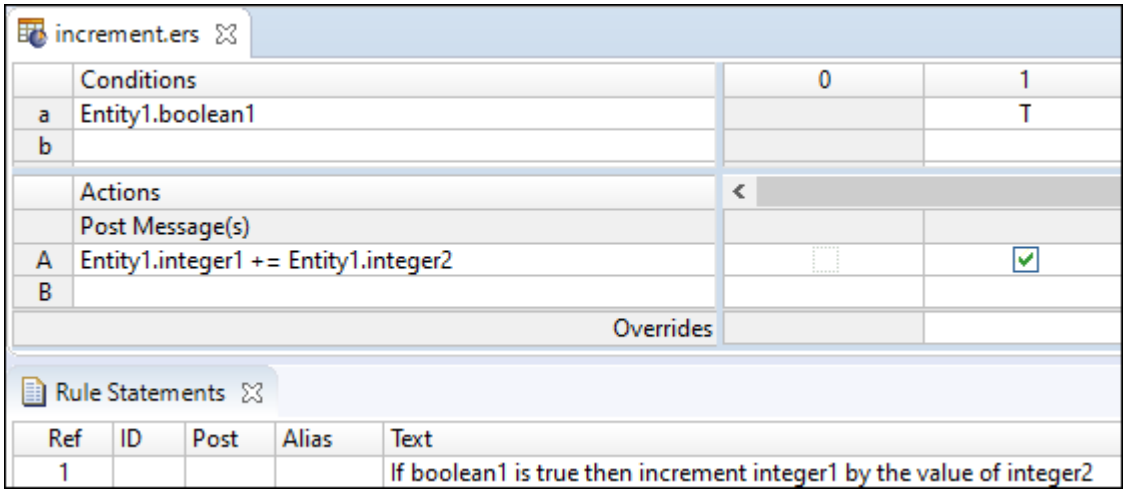
Increments `<Number1>` by the value of `<Number2>`. The data type of `<Number1>` must accommodate the addition of `<Number2>`. In other words, an Integer may not be incremented by a Decimal without using another operator (such as .toInteger or Floor on page 96.floor) to first convert the Decimal to an Integer.

USAGE RESTRICTIONS

The Operators row of the table in Vocabulary usage restrictions does not apply. Special exceptions: **increment** may only be used in Action Rows (section 5 in Sections of Rulesheet that correlate with usage restrictions).

RULESHEET EXAMPLE

This sample Rulesheet uses **increment** to increment `integer1` by the value of `integer2` when `boolean1` is `true`.



increment.ers

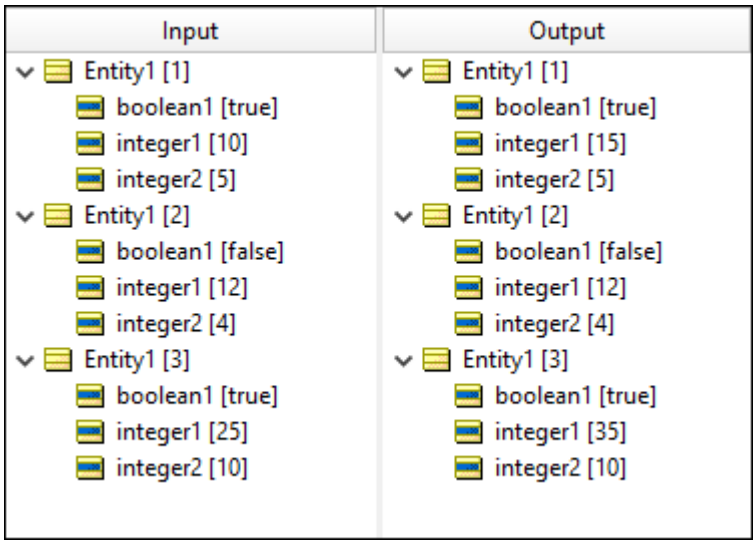
	Conditions	0	1
a	Entity1.boolean1		T
b			
	Actions		
	Post Message(s)		
A	Entity1.integer1 += Entity1.integer2	☐	☑
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If boolean1 is true then increment integer1 by the value of integer2

SAMPLE RULETEST

A sample Ruletest provides three examples of `integer1`, `integer2`, and `boolean1`. Input and Output panels are shown below.



Index of

SYNTAX

`<String1>`.indexOf(`<String2>`)

DESCRIPTION

Determines if `<String2>` is contained within `<String1>` and returns an Integer value equal to the beginning character position of the first occurrence of `<String2>` within `<String1>`. If `<String1>` does not contain `<String2>`, then a value of `0` (zero) is returned. This operator is similar to .contains but returns different results. A `0` result from **.indexOf** is equivalent to a `false` value returned by the .contains operator.

If `<String1>` contains more than one occurrence of `<String2>`, **.indexOf** returns the first character position of the first occurrence. For example: If `<String1>` holds the String value `'Mississippi'` and `<String2>` holds the String value `'ss'`, then the **.indexOf** operator returns 3. The second occurrence of `'ss'` beginning at position 6 is not identified.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.indexOf** to evaluate whether `string1` includes the characters `silver` and assigns a value to `integer1` corresponding to the beginning character position of the first occurrence.

IndexOf.ers

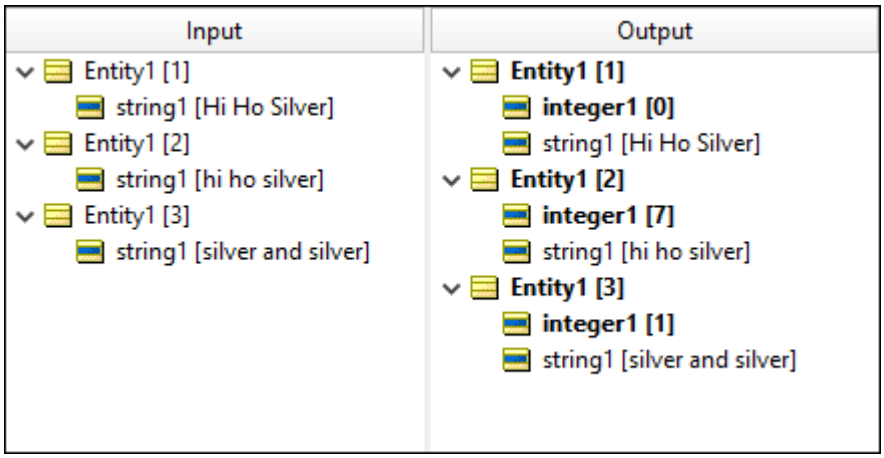
	Conditions	0	1
a			
b			
	Actions		
	Post Message(s)		
A	Entity1.integer1 = Entity1.string1.indexOf('silver')	☑	
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
A0				integer1 is assigned the value of the starting position of silver inside string1

SAMPLE RULETEST

A sample Ruletest provides `string1` values for three examples. Input and Output panels are shown below. Notice sensitivity to case in example 1.



Is empty

SYNTAX

<Collection> ->isEmpty

DESCRIPTION

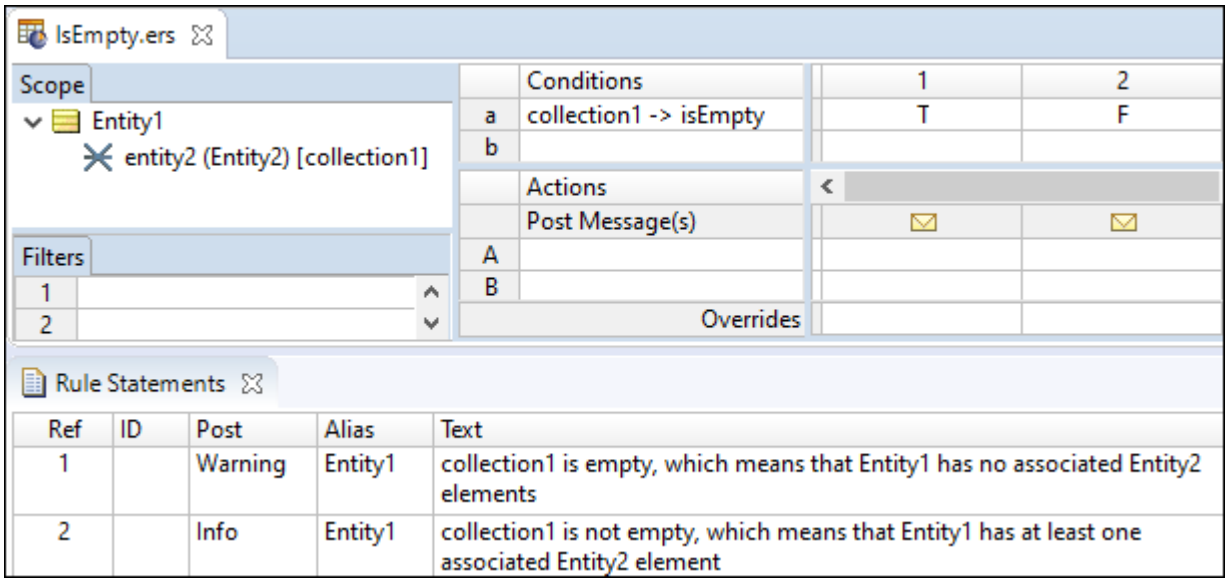
Returns a value of true if <Collection> contains *no* elements (that is, has no children). **->isEmpty** does not check for an empty or null value of an attribute, but instead checks for *existence* of elements within the collection. As such, a unique alias must be used to represent the <Collection> being tested.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

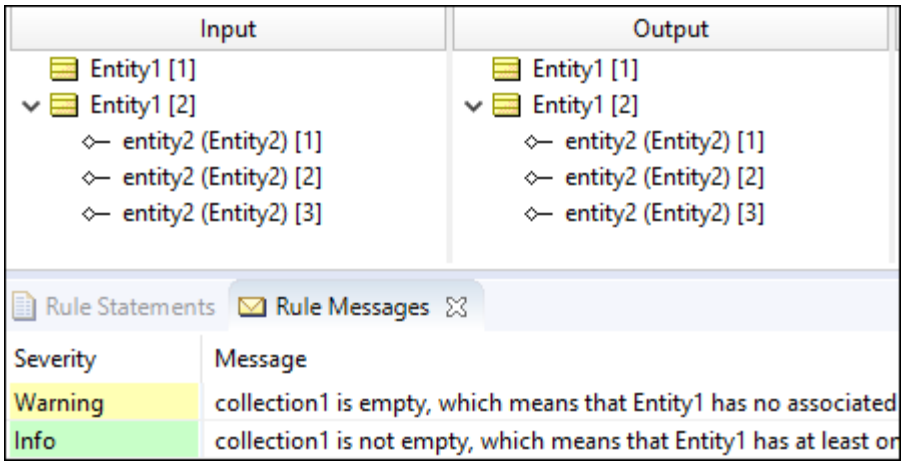
RULESHEET EXAMPLE

This sample Rulesheet uses **->isEmpty** to determine if `collection1` has any elements. Note the use of unique alias `collection1` to represent the collection of `Entity2` associated with `Entity1`.



SAMPLE RULETEST

A sample Ruletest provides two example `collection1`. The following illustration shows Input and Output panels



Is integer

SYNTAX

```
<String>.isInteger
```

DESCRIPTION

Returns true if string is an integer

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

This sample Rulesheet uses **isInteger**.

isInteger.ers

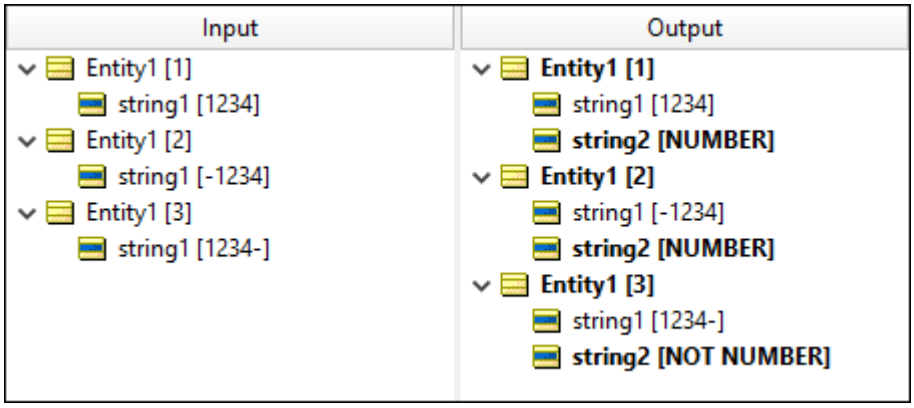
	Conditions	0	1	2
a	Entity1.string1.isInteger		T	F
b				
c				
	Actions			
	Post Message(s)			
A	Entity1.string2		'NUMBER'	'NOT NUMBER'
B				
C				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1				If Entity1.string1 is an integer, set Entity1.string2 to 'NUMBER'
2				If Entity1.string2 is not an integer, set Entity1.string2 to 'NOT NUMBER'

SAMPLE RULETEST

A sample Ruletest provides a collection of three elements, each with a `string11` value. Input and Output panels are shown below.



Last

SYNTAX

<Sequence> ->last.<Attribute1>

DESCRIPTION

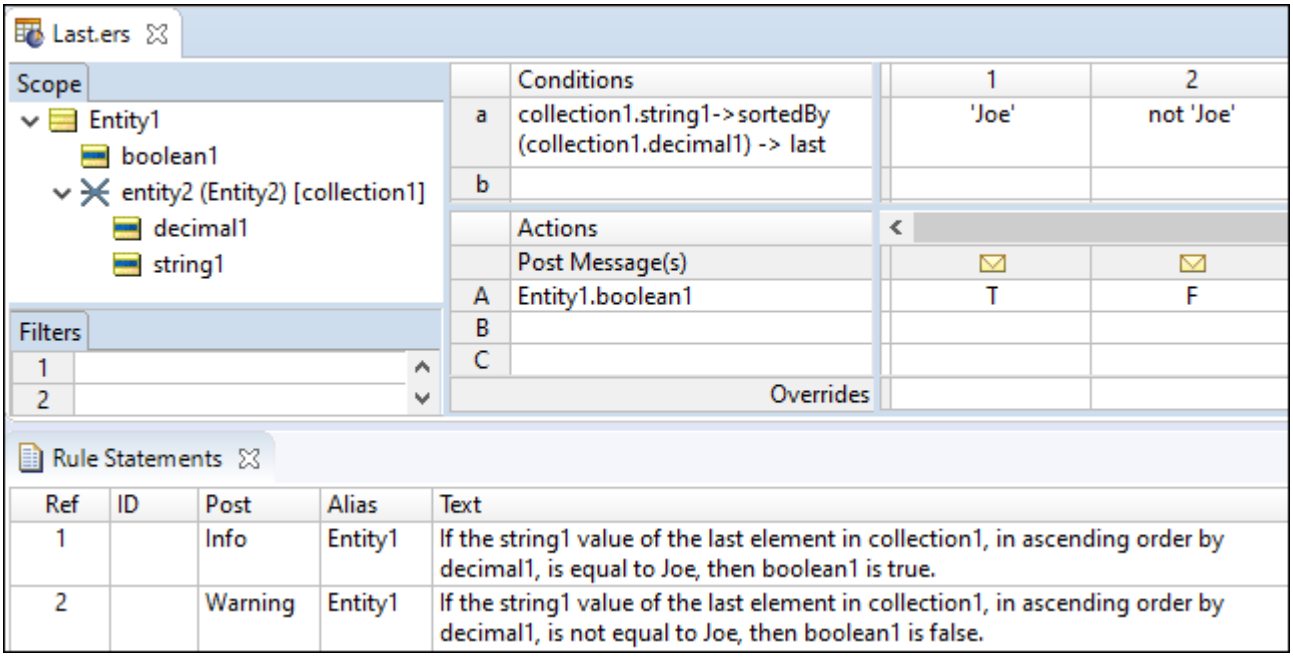
Returns the value of <Attribute1> of the last element in <Sequence>. Another operator, such as ->sortedBy, must be used to transform a <Collection> into a <Sequence> before **->last** may be used. <Sequence> must be expressed as a unique alias. <Attribute1> may be of any data type. See *"Advanced collection sorting syntax" in the Rule Modeling Guide* for more examples of usage.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

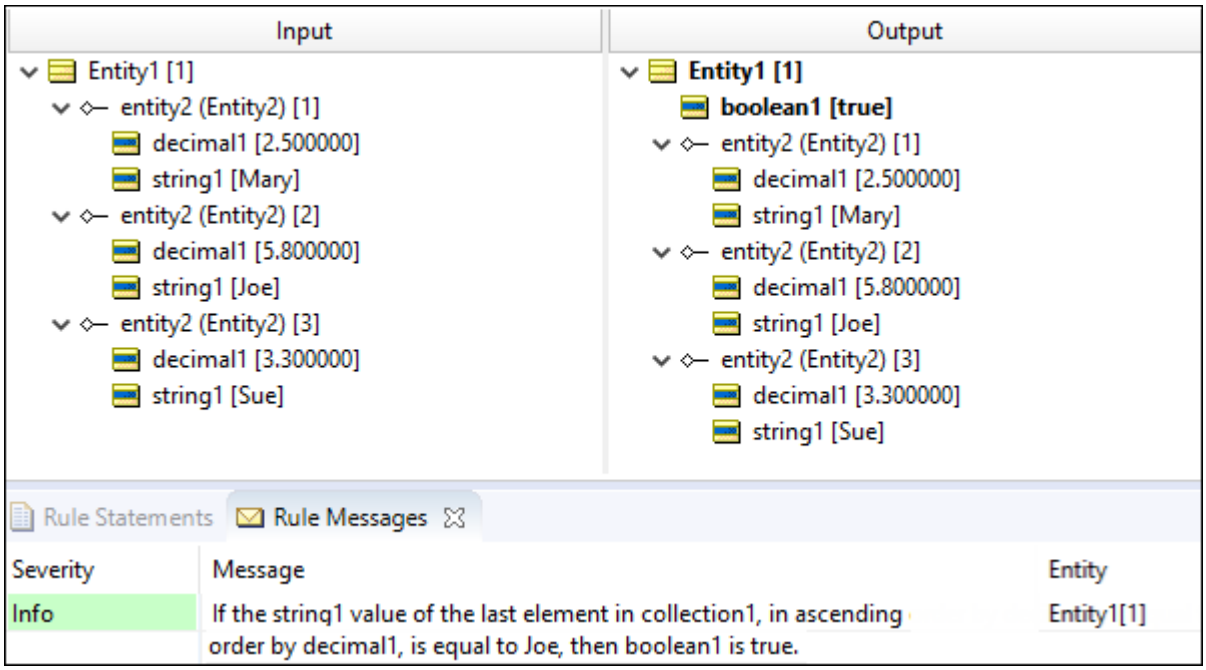
RULESHEET EXAMPLE

This sample Rulesheet uses **->last** to identify the last element of the sequence created by applying ->sortedBy to `collection1`. Once identified, the value of the `string1` attribute belonging to this last element is evaluated. If the value of `string1` is `Joe`, then `boolean1` attribute of `Entity1` is assigned the value of `true`.



SAMPLE RULETEST

A sample Ruletest provides a collection of three elements, each with a `decimal1` value. Input and Output panels are shown below.



Is same date

SYNTAX

`<DateTime1>.isSameDate(<DateTime2>)`

DESCRIPTION

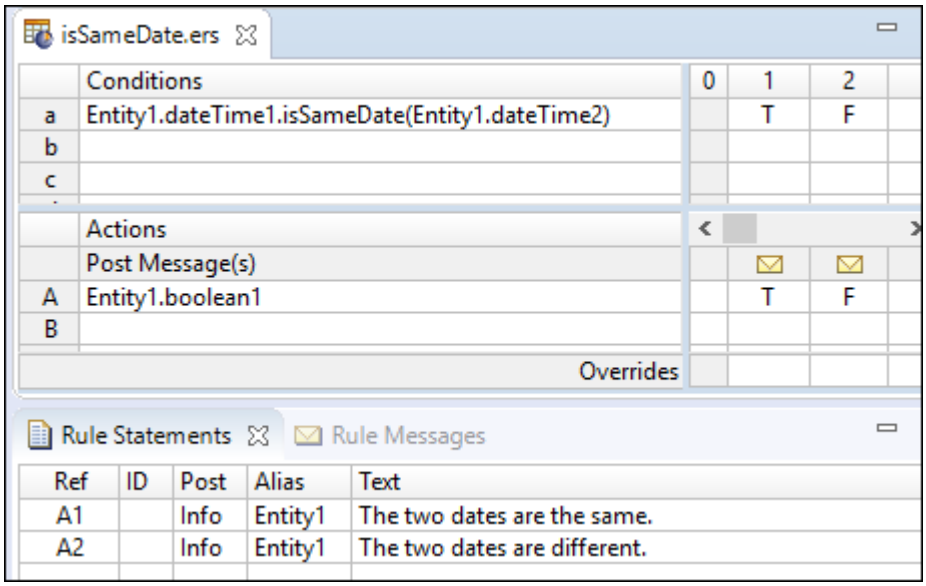
Returns boolean. True if the DateTime1 is the same as DateTime2, ignoring the time part.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

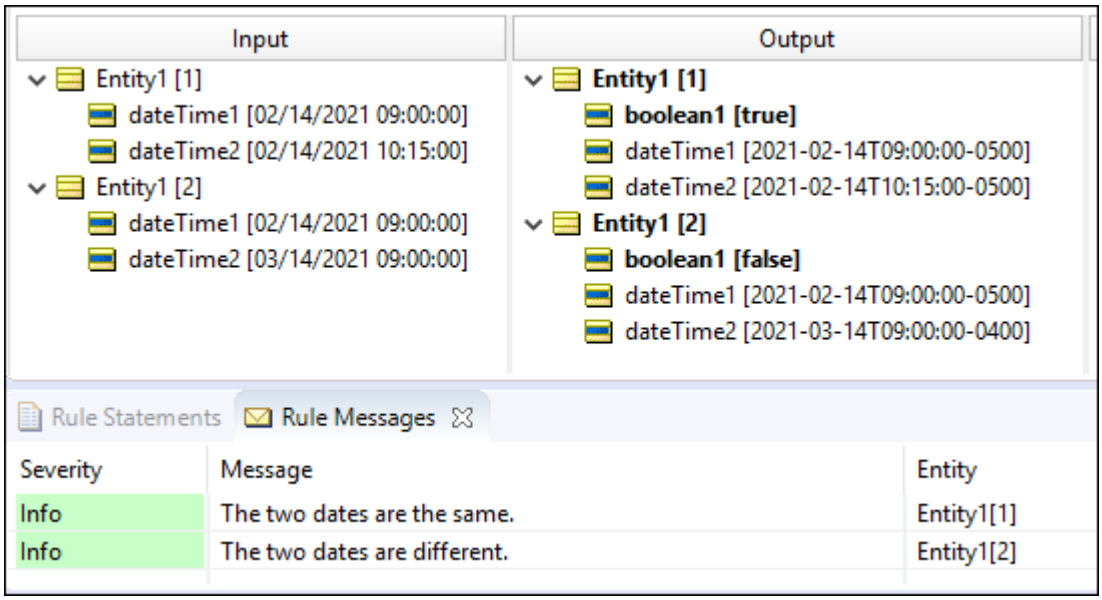
RULESHEET EXAMPLE

The following Rulesheet uses **.isSameDate** to determine whether two dateTime values are the same date.



SAMPLE RULETEST

A sample Ruletest provides `dateTime1` and `dateTime2` for two examples. Input and Output panels are shown below.



Is same time

SYNTAX

`<DateTime1>`.isSameTime(`<DateTime2>`)

DESCRIPTION

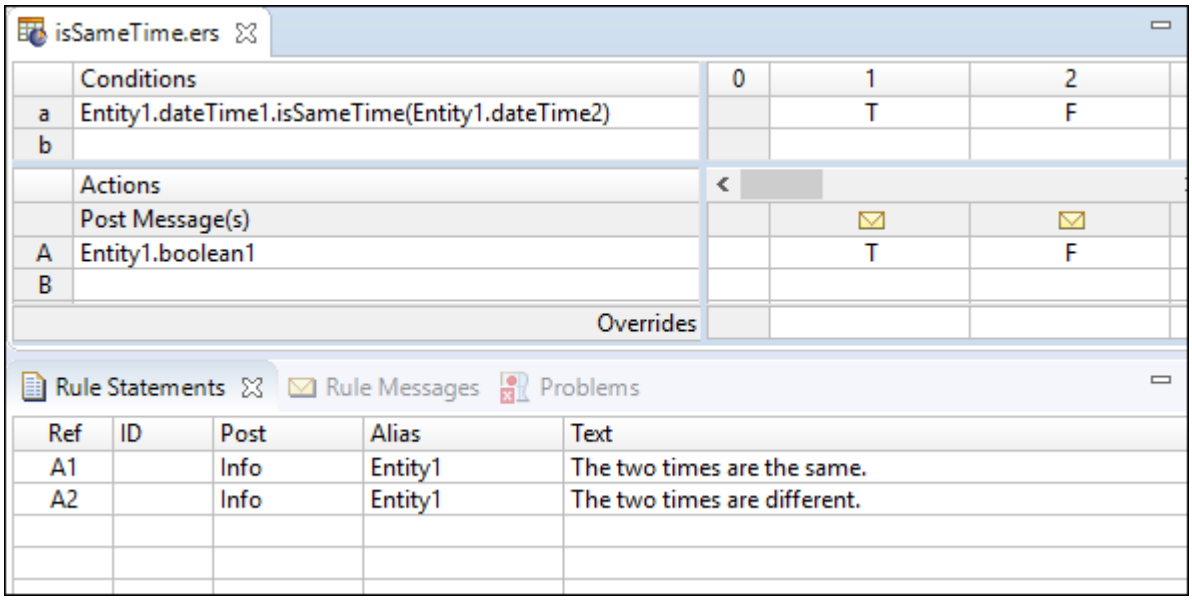
Returns boolean. True if DateTime1 is the same as DateTime2, ignoring the date part.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

RULESHEET EXAMPLE

The following Rulesheet uses **.isSameTime** to determine whether two dateTime values are the same time.



isSameTime.ers

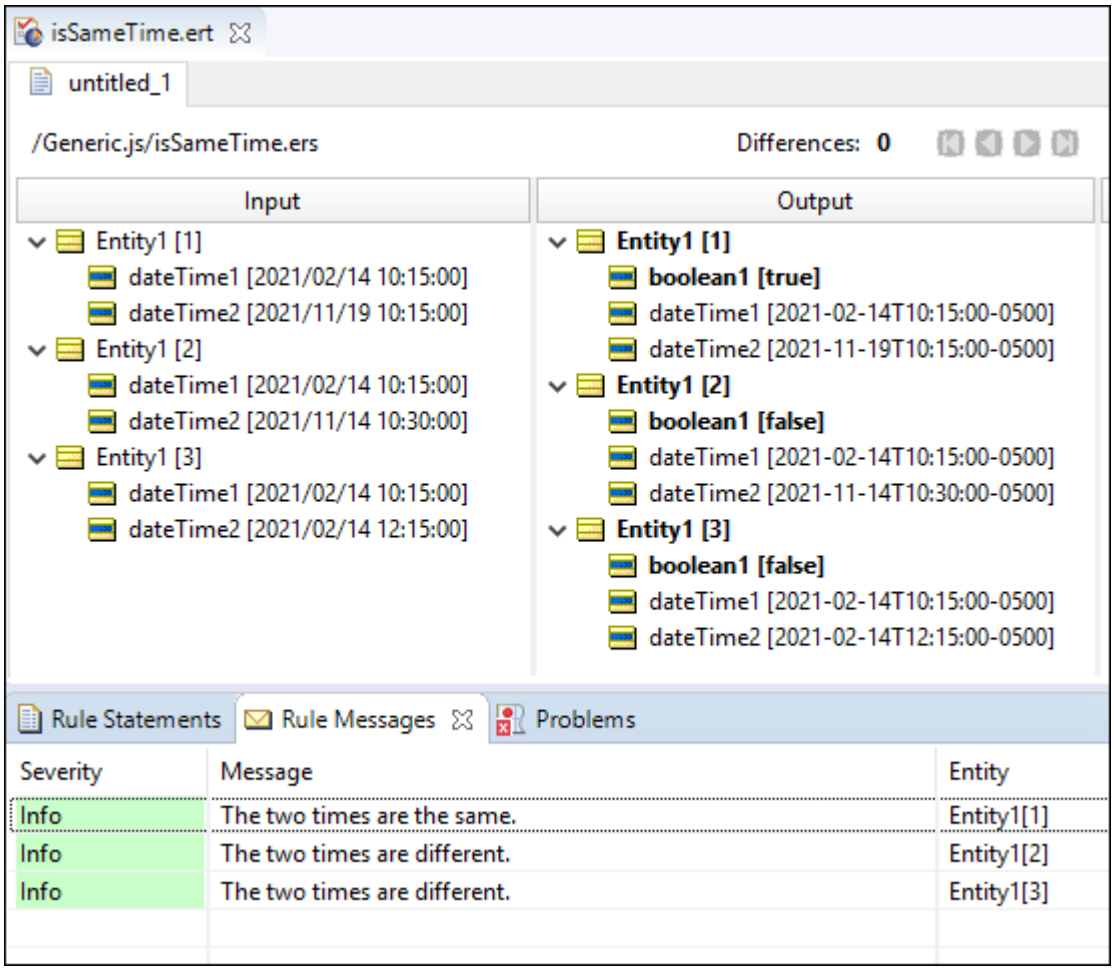
	Conditions	0	1	2
a	Entity1.dateTime1.isSameTime(Entity1.dateTime2)		T	F
b				
	Actions			
	Post Message(s)		✉	✉
A	Entity1.boolean1		T	F
B				
	Overrides			

Rule Statements | Rule Messages | Problems

Ref	ID	Post	Alias	Text
A1		Info	Entity1	The two times are the same.
A2		Info	Entity1	The two times are different.

SAMPLE RULETEST

A sample Ruletest provides `dateTime1` and `dateTime2` for two examples. Input and Output panels are shown below.



Less than

SYNTAX

DateTime*	`<DateTime1> < <DateTime2>`
Number	`<Number1> < <Number2>`
String	`<String1> < <String2>`

DESCRIPTION

DateTime*	Returns a value of true if <DateTime1> is less than <DateTime2>. This is equivalent to <DateTime1> occurring “before” <DateTime2>
Number	Returns a value of true if <Number1> is less than <Number2>. Different numeric data types may be compared in the same expression.
String	Returns a value of true if <String1> is less than <String2>.

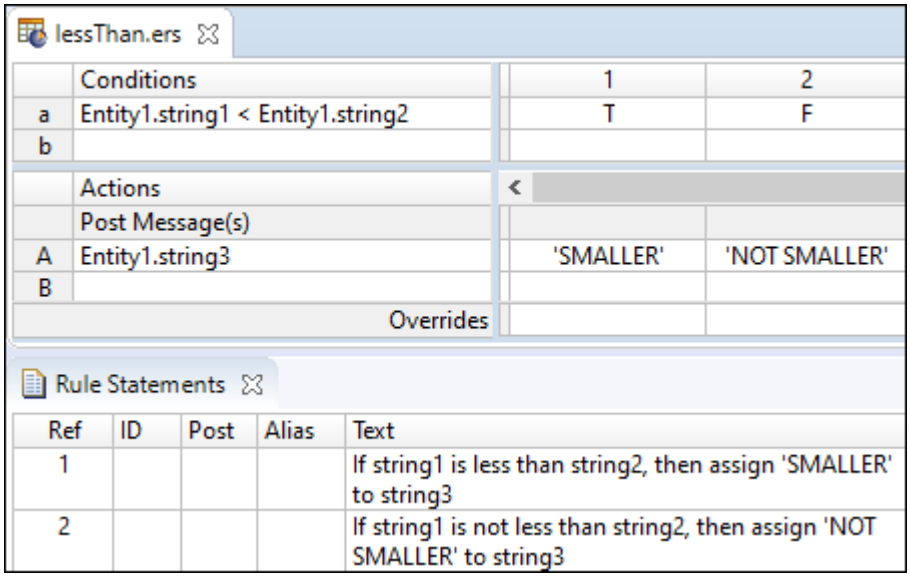
* includes DateTime and Date data types

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction**less than** may also be used in Conditional Value Sets & Cells (section 5 in applies, with the following exception: Sections of Rulesheet: Numbers Correlate with Table Above).

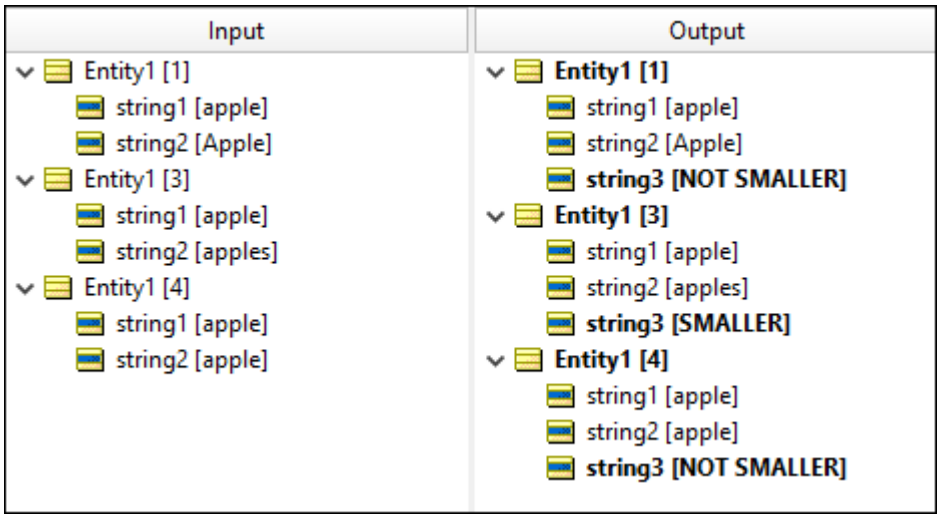
RULESHEET EXAMPLE

The following Rulesheet uses **less than** to test whether `string1` is less than `string2`.



SAMPLE RULETEST

A sample Ruletest provides two examples. Input and Output panels are shown below:



Less than or equal to

SYNTAX

DateTime*	`<DateTime1> <= <DateTime2>`
Number	`<Number1> <= <Number2>`
String	`<String1> <= <String2>`

DESCRIPTION

DateTime*	Returns a value of true if `<DateTime1>` is less than or equal to `<DateTime2>`. This is equivalent to `<DateTime1>` occurring “on or before” `<DateTime2>`
Number	Returns a value of true if `<Number1>` is less than or equal to `<Number2>`. Different numeric data types may be compared in the same expression.
String	Returns a value of true if `<String1>` is less than or equal to `<String2>`.

* includes DateTime and Date data types

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies, with the following exception: **less than or equal to** may also be used in Conditional Value Sets & Cells (section 5 of Sections of Rulesheet that correlate with usage restrictions).

RULESHEET EXAMPLE

The following Rulesheet uses **less than or equal to** to test whether `string1` is less than or equal to `string2`.

lessThanEqual.ers

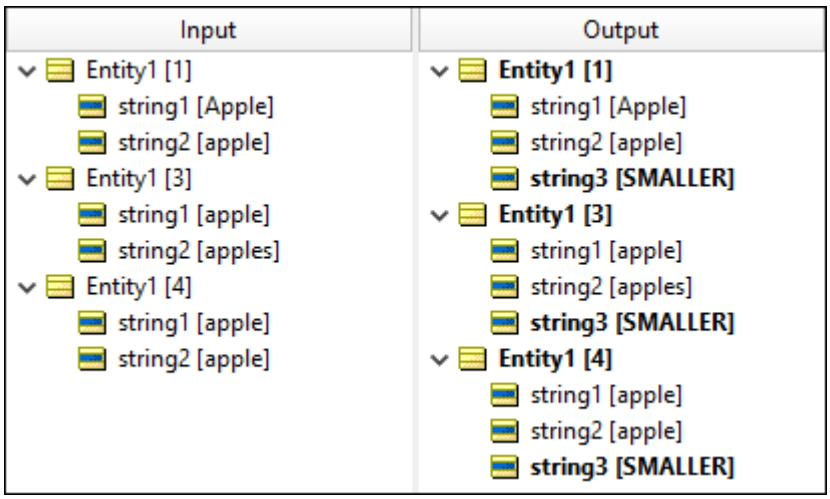
	Conditions	1	2
a	Entity1.string1 <= Entity1.string2	T	F
b			
	Actions		
	Post Message(s)		
A	Entity1.string3	'SMALLER'	'NOT SMALLER'
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If string1 is less than or equal to string2, then assign 'SMALLER' to string3
2				If string1 is not less than or equal to string2, then assign 'NOT SMALLER' to string3

SAMPLE RULETEST

A sample Ruletest provides two examples. Input and Output panels are shown below:



Logarithm BASE 10

SYNTAX

`<Number>`.log

DESCRIPTION

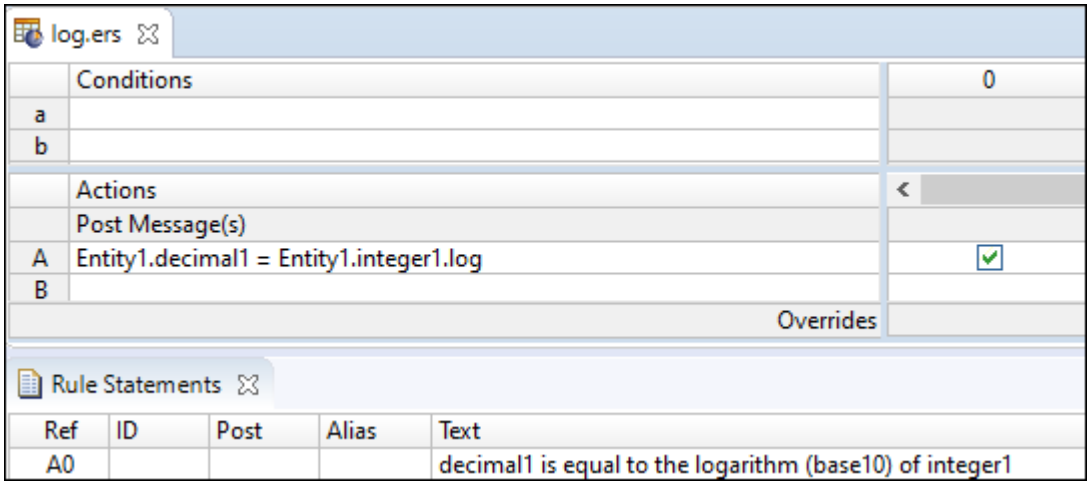
Returns a Decimal value equal to the logarithm (base 10) of `<Number>`. If `<Number>` is equal to `0` (zero) an error is returned when the rule is executed.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

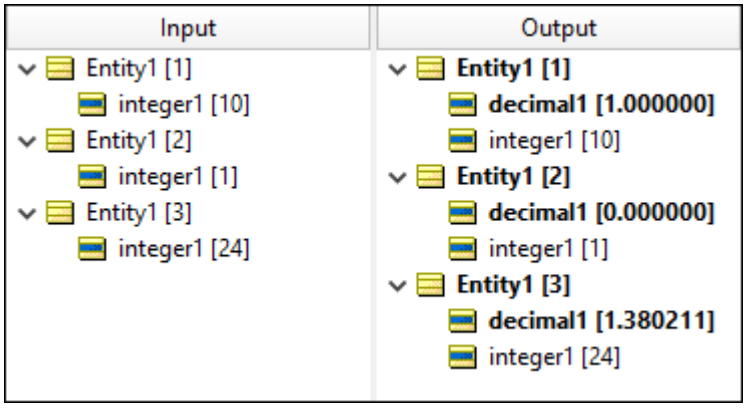
RULESHEET EXAMPLE

The following Rulesheet uses **.log** to calculate the logarithm (base 10) of `integer1` and assign it to `decimal1`.



SAMPLE RULETEST

A sample Ruletest provides results for three examples of `integer1`. Input and Output panels are shown below:



Note: In a case where the rule encounters `log(0)`, it throws an exception that halts execution. That's because the value of `log(0)` is undefined. If the rule is executing against multiple entities, the arbitrary order of execution might be different on subsequent runs before execution is halted.

Logarithm BASE X

SYNTAX

`<Number>.log(<Decimal>)`

DESCRIPTION

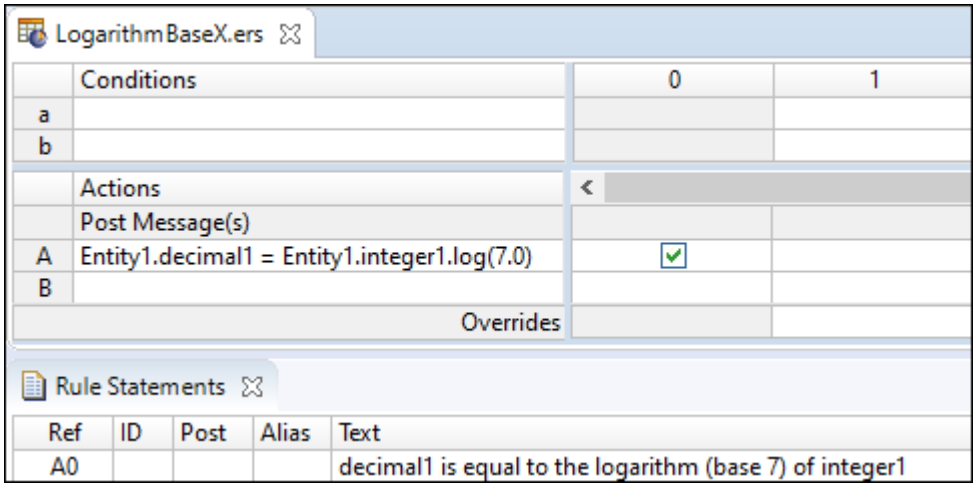
Returns a Decimal value equal to the logarithm (base `<Decimal>`) of `<Number>`. If `<Number>` is equal to `0` (zero) an error is returned when the rule is executed.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

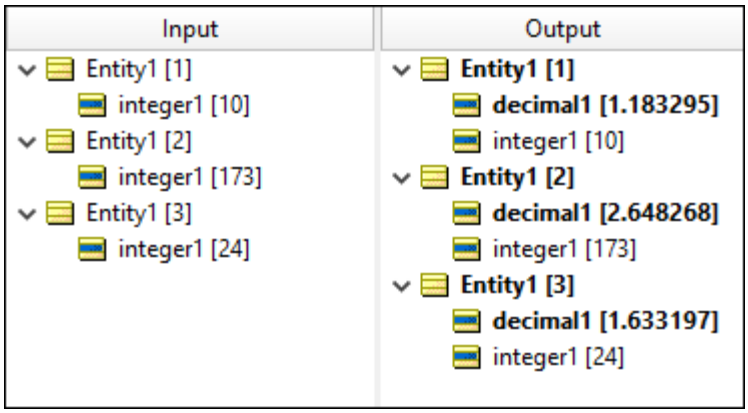
RULESHEET EXAMPLE

The following Rulesheet uses **.log** to calculate the logarithm (base 7.0) of `integer1` and assign it to `decimal1`.



SAMPLE RULETEST

A sample Ruletest provides results for three examples of `integer1`. Input and Output panels are shown below:



Note: In a case where the rule encounters `log(0)`, it throws an exception that halts execution. That's because the value of `log(0)` is undefined. If the rule is executing against multiple entities, the arbitrary order of execution might be different on subsequent runs before execution is halted.

Lowercase

SYNTAX

<String>.toLower

DESCRIPTION

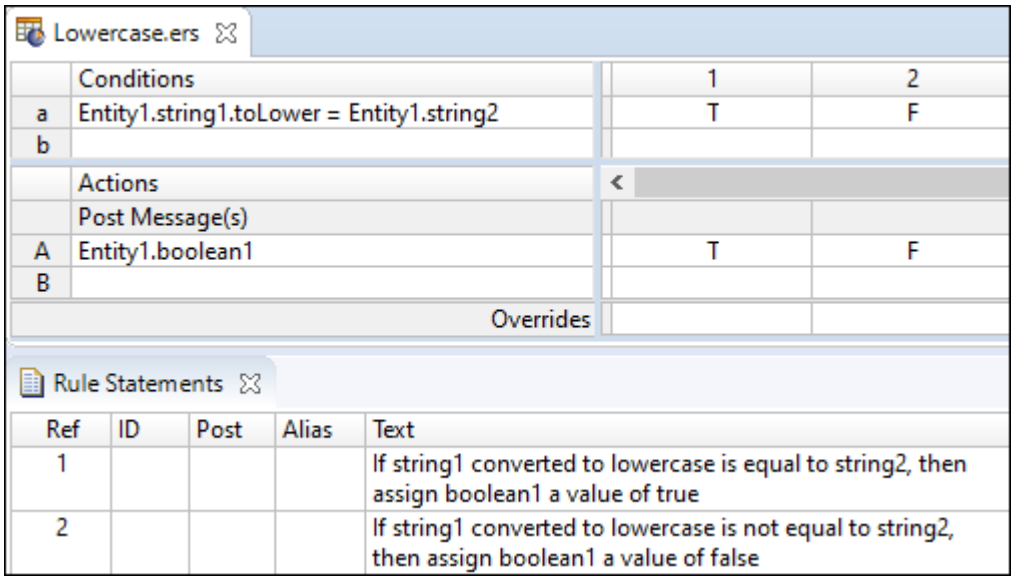
Converts all characters in <String> to lowercase characters.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.toLower** to convert string1 to lowercase, compare its value with string2, and assign a value to boolean1 based on the results of the comparison.



Lowercase.ers

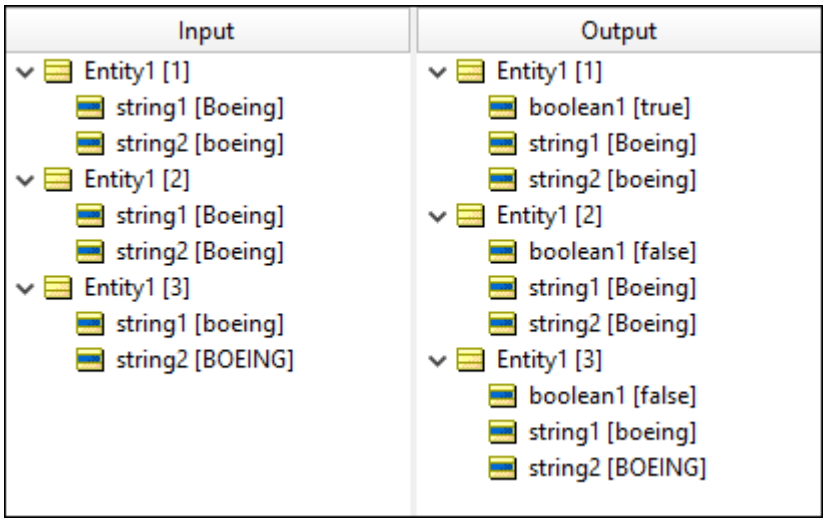
	Conditions	1	2
a	Entity1.string1.toLower = Entity1.string2	T	F
b			
	Actions		
	Post Message(s)		
A	Entity1.boolean1	T	F
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If string1 converted to lowercase is equal to string2, then assign boolean1 a value of true
2				If string1 converted to lowercase is not equal to string2, then assign boolean1 a value of false

SAMPLE RULETEST

A sample Ruletest provides three examples of string1 and string2. Input and Output panels are shown below:



Matches

SYNTAX

```
<String>.matches(regularExpression:String)
```

```
<String>.matches(regularExpression:String,flags:<String>)
```

DESCRIPTION

Returns true if the regular expression matches the String.

FLAGS

The characters `gis`, when one or more are added to the expression with no separator, represent:

- `g` global, replace more than first match (the default)
- `i` ignore case
- `s` match line terminator ("newline") characters in a string, which it would not match otherwise. See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/dotAll.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLES

This sample Rulesheet uses **matches** in non-conditional actions:

matches.ers

	Conditions	0
a		
b		
	Actions	
	Post Message(s)	
A	Entity1.boolean1 = Entity1.string1.matches ('[A-Z,a-z]{5}[0-9]{4}[A-Z,a-z]{1}')	☑
B	Entity1.boolean2 = Entity1.string2.matches ('[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\\.[a-zA-Z0-9-.]+$')	☑
	Overrides	

Rule Statements

Ref	ID	Post	Alias	Text
A0				Entity1.boolean1 is true if string1 is a valid identifier, and false otherwise
B0				Entity1.boolean2 is true if string2 is a valid email address, and false otherwise

Action A: Determine whether a String is a valid identifier - A String must contain an item identification with the following pattern:

1. Characters 1-5: alphabetic.
2. Characters 6-10: numeric.
3. Character 11: alphabetic.

Action B: Check whether an email address is valid - An email address must have alphanumeric characters and certain special characters before and after an @ and a dot.

SAMPLE RULETEST

A sample Ruletest provides various valid and invalid Strings that are evaluated by the two regular expression examples.

Input	Output
Entity1 [1]	**Entity1 [1]**
string1 [ABCDE1234x]	**boolean1 [true]**
string2 [ProgressSupport@progress.com]	**boolean2 [true]**
Entity1 [2]	string1 [ABCDE1234x]
string1 [ABCDEFGHIx]	string2 [ProgressSupport@progress.com]
string2 [???@progress.com]	**Entity1 [2]**
Entity1 [3]	**boolean1 [false]**
string1 [ABCDE-1234-x]	**boolean2 [false]**
string2 [ProgressSupport @ progress.com]	string1 [ABCDEFGHIx]
	string2 [???@progress.com]
	Entity1 [3]
	boolean1 [false]
	boolean2 [false]
	string1 [ABCDE-1234-x]
	string2 [ProgressSupport @ progress.com]

Maximum value

SYNTAX

`<Number1>.max(<Number2>)`

DESCRIPTION

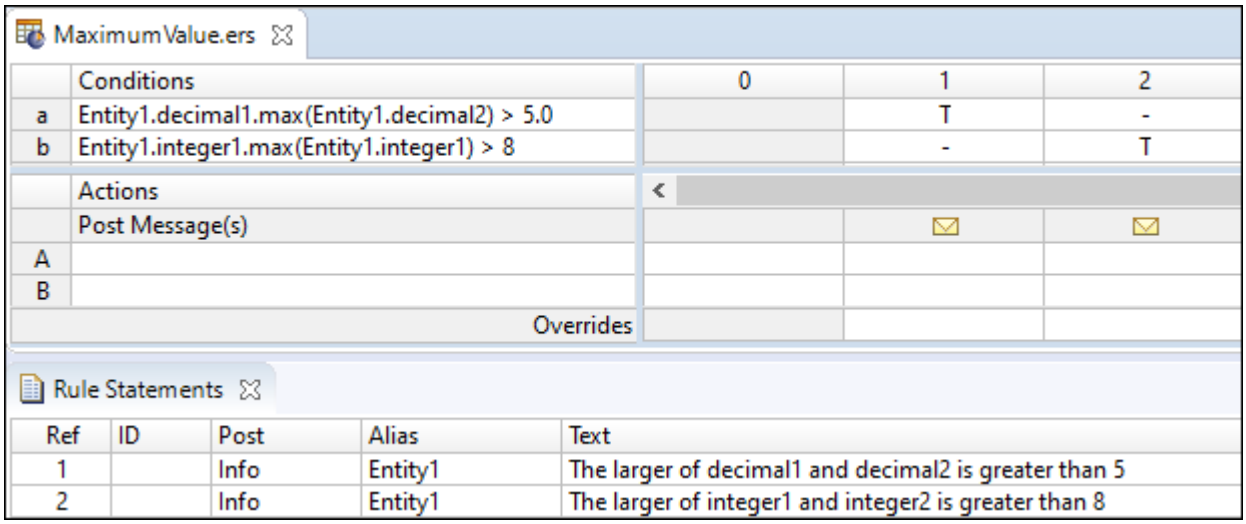
Returns either `<Number1>` or `<Number2>`, whichever is greater.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.max** to compare the values of `decimal1` and `decimal2`, and `integer1` and `integer2`, and posts a message based on their size relative to `5.0` and `8`, respectively.



MaximumValue.ers

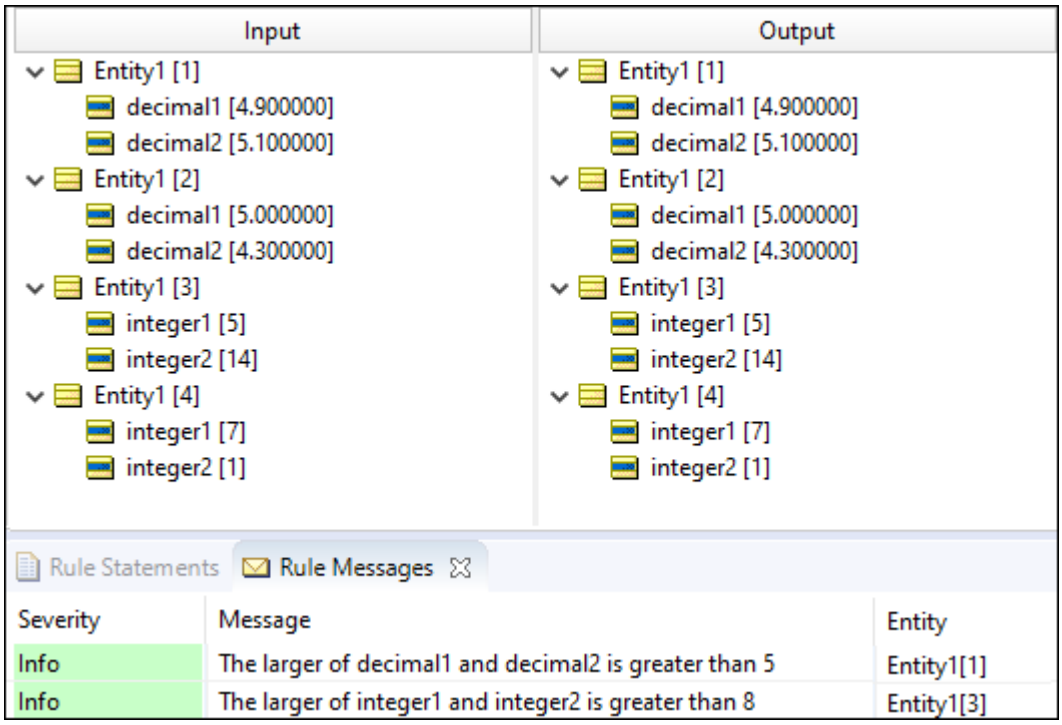
	Conditions	0	1	2
a	Entity1.decimal1.max(Entity1.decimal2) > 5.0		T	-
b	Entity1.integer1.max(Entity1.integer1) > 8		-	T
	Actions			
	Post Message(s)		✉	✉
A				
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1		Info	Entity1	The larger of decimal1 and decimal2 is greater than 5
2		Info	Entity1	The larger of integer1 and integer2 is greater than 8

SAMPLE RULETEST

A sample Ruletest provides four examples, two using `decimal1` and `decimal2`, and two using `integer1` and `integer2` as input data.



Maximum value COLLECTION

SYNTAX

`<Collection.attribute>` -> max

DESCRIPTION

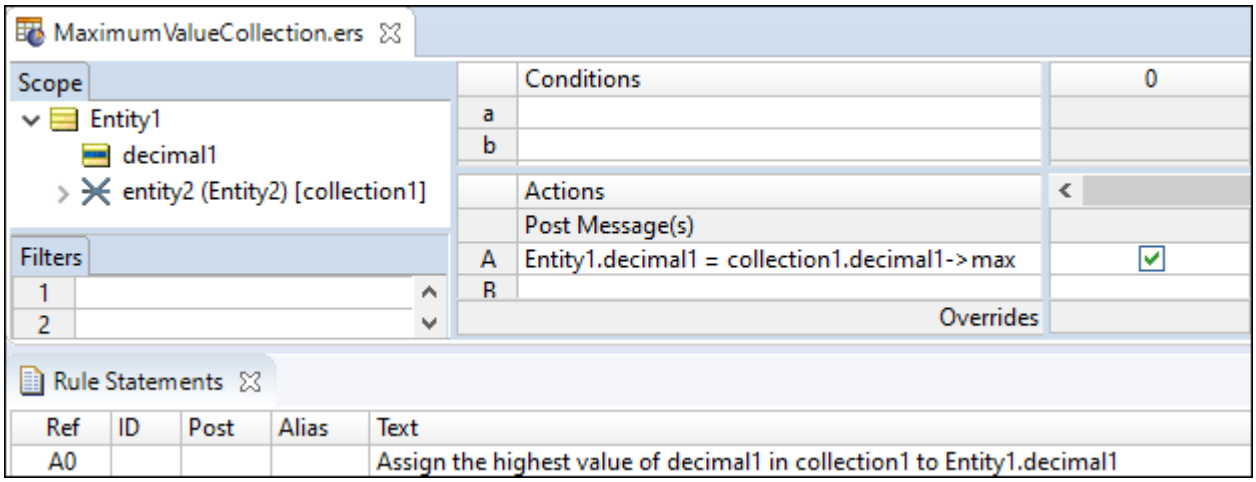
Returns the highest value of `<attribute>` for all elements in `<Collection>`. `<attribute>` must be a numeric data type. `<Collection>` must be expressed as a unique alias.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

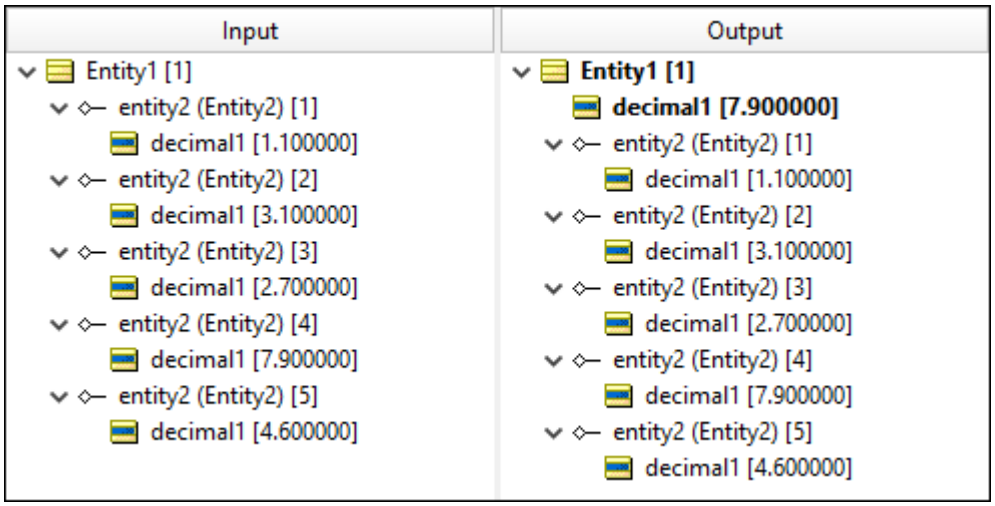
RULESHEET EXAMPLE

The following Rulesheet uses **->max** to identify the highest value of `decimal1` in all elements of `collection1`, then assign it to `Entity1.decimal1`.



SAMPLE RULETEST

A sample collection contains five elements, each with a value of `decimal1`.



Minimum value

SYNTAX

<Number1>.min(<Number2>)

DESCRIPTION

Returns either <Number1> or <Number2>, whichever is smaller.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.min** to compare the values of `decimal1` and `decimal2`, and `integer1` and `integer2`, and posts a message based on their size relative to `5.0` and `8`, respectively.

MinimumValue.ers

	Conditions	0	1	2
a	Entity1.decimal1.min(Entity1.decimal2) > 5.0		T	-
b	Entity1.integer1.min(Entity1.integer2) > 8		-	T
c				
	Actions			
	Post Message(s)		✉	✉
A				
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1		Info	Entity1	The smaller of decimal1 and decimal2 is greater than 5
2		Info	Entity1	The smaller of integer1 and integer2 is greater than 8

SAMPLE RULETEST

A sample Ruletest provides four examples, two using decimal inputs, and two using integers.

Input	Output
Entity1 [1]	Entity1 [1]
decimal1 [4.900000]	decimal1 [4.900000]
decimal2 [5.100000]	decimal2 [5.100000]
Entity1 [2]	Entity1 [2]
decimal1 [5.100000]	decimal1 [5.100000]
decimal2 [594.300000]	decimal2 [594.300000]
Entity1 [3]	Entity1 [3]
integer1 [1500]	integer1 [1500]
integer2 [245]	integer2 [245]
Entity1 [4]	Entity1 [4]
integer1 [350]	integer1 [350]
integer2 [1]	integer2 [1]

Rule Statements | Rule Messages

Severity	Message	Entity
Info	The smaller of decimal1 and decimal2 is greater than 5	Entity1[2]
Info	The smaller of integer1 and integer2 is greater than 8	Entity1[3]

Minimum value COLLECTION

SYNTAX

`<Collection.attribute>` -> min

DESCRIPTION

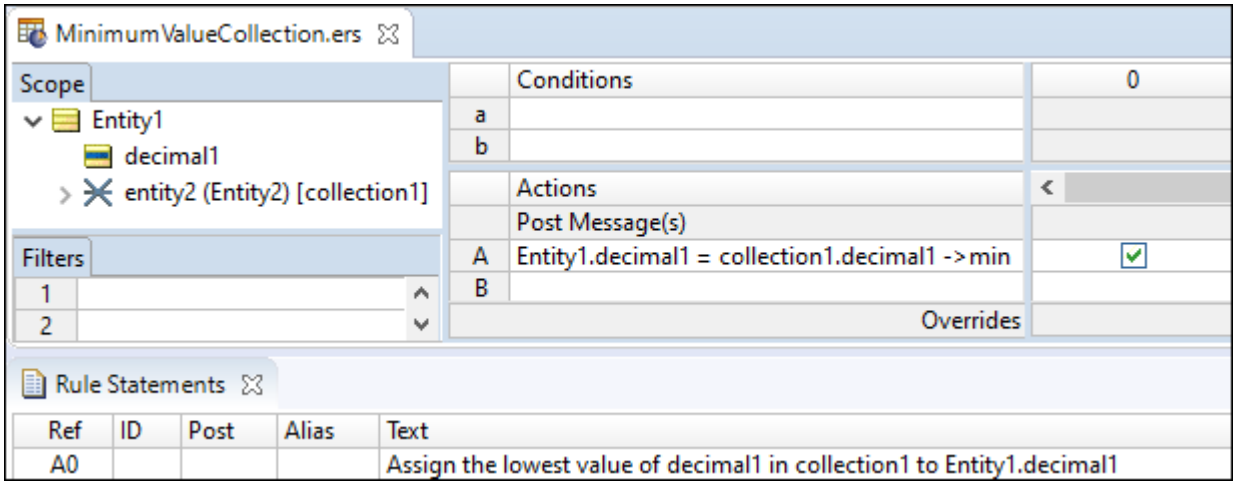
Returns the lowest value of `<attribute>` for all elements in `<Collection>`. `<attribute>` must be a numeric data type. `<Collection>` must be expressed as a unique alias.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **->min** to identify the lowest value of `decimal1` in all elements of `collection1`, then assign it to `Entity1.decimal1`.



SAMPLE RULETEST

A sample collection contains five elements, each with a value of `decimal1`.

Input	Output
Entity1 [1] entity2 (Entity2) [1] decimal1 [1.100000] entity2 (Entity2) [2] decimal1 [3.100000] entity2 (Entity2) [3] decimal1 [2.700000] entity2 (Entity2) [4] decimal1 [7.900000] entity2 (Entity2) [5] decimal1 [4.600000]	**Entity1 [1]** **decimal1 [1.100000]** entity2 (Entity2) [1] decimal1 [1.100000] entity2 (Entity2) [2] decimal1 [3.100000] entity2 (Entity2) [3] decimal1 [2.700000] entity2 (Entity2) [4] decimal1 [7.900000] entity2 (Entity2) [5] decimal1 [4.600000]

Minute

SYNTAX

`<DateTime>`.min

DESCRIPTION

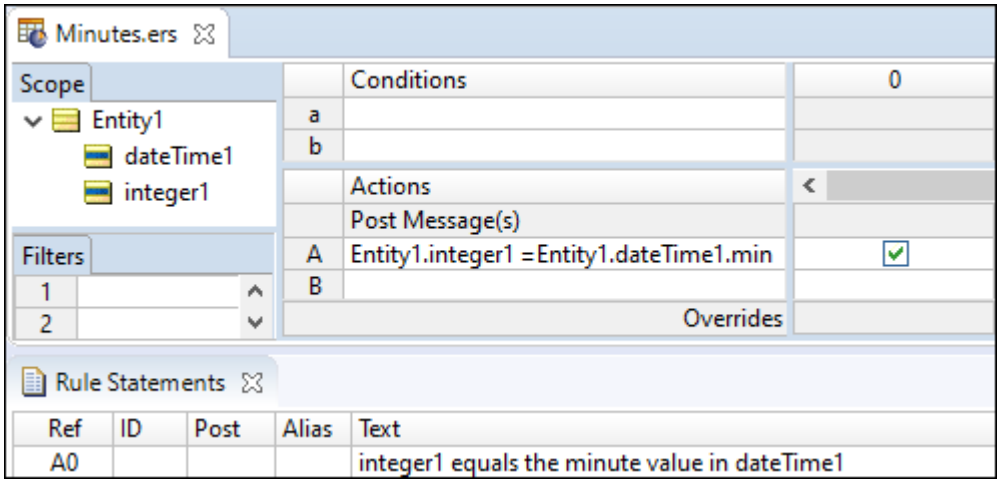
Returns the minute portion of `<DateTime>` as an Integer between 0 and 59.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

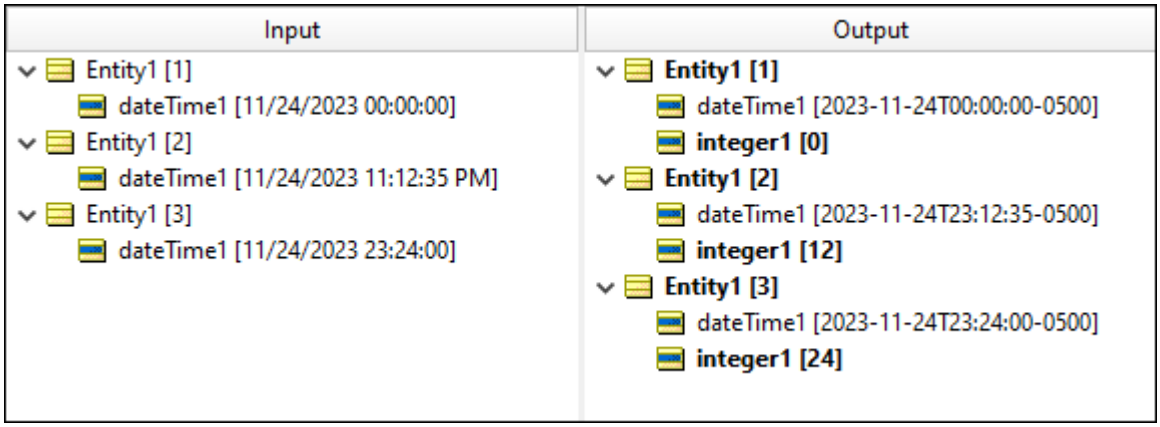
RULESHEET EXAMPLE

The following Rulesheet uses **.min** to evaluate `dateTime1` and assign the minute value to `integer1`.



SAMPLE RULETEST

A sample Ruletest provides three examples of `dateTime1`. Input and Output panels are shown below:



Minutes between

SYNTAX

`<DateTime1>`.minsBetween(`<DateTime2>`)

DESCRIPTION

Returns the Integer number of minutes between DateTimes. The function calculates the number of milliseconds between the two dates and divides that number by 60,000 (the number of milliseconds in a minute). The decimal portion is then truncated. If the two dates differ by less than a full minute, the returned value is zero. This function returns a positive number if `<DateTime2>` is later than `<DateTime1>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.minsBetween** to determine the number of minutes that have elapsed between `dateTime1` and `dateTime2`, compare it to the Values set, and assign a value to `string1`.

MinutesBetween.ers

	Conditions	1	2
a	Entity1.dateTime1.minsBetween(Entity1.dateTime2)	<= 30	> 30
b			
	Actions		
	Post Message(s)		
A	Entity1.string1	'Not Overdue'	'Overdue'
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If 30 or fewer minutes have elapsed between dateTime1 and dateTime2, then Entity1 is not overdue
2				If more than 30 minutes have elapsed between dateTime1 and dateTime2, then Entity2 is overdue

SAMPLE RULETEST

A sample Ruletest provides `dateTime1` and `dateTime2` for two examples. Input and Output panels are shown below.

Input	Output
Entity1 [1]	**Entity1 [1]**
dateTime1 [11/24/1960 00:00:00]	dateTime1 [1960-11-24T00:00:00-0500]
dateTime2 [12/15/1960 00:00:00]	dateTime2 [1960-12-15T00:00:00-0500]
Entity1 [2]	**string1 [Overdue]**
dateTime1 [11/24/1960 00:00:00]	**Entity1 [2]**
dateTime2 [11/24/1960 00:10:00]	dateTime1 [1960-11-24T00:00:00-0500]
Entity1 [3]	dateTime2 [1960-11-24T00:10:00-0500]
dateTime1 [08/04/2020 01:00:00]	**string1 [Not Overdue]**
dateTime2 [08/04/2020 01:15:00]	**Entity1 [3]**
	dateTime1 [2020-08-04T01:00:00-0400]
	dateTime2 [2020-08-04T01:15:00-0400]
	string1 [Not Overdue]

Mod

SYNTAX

`<Integer1>.mod(<Integer2>)`

DESCRIPTION

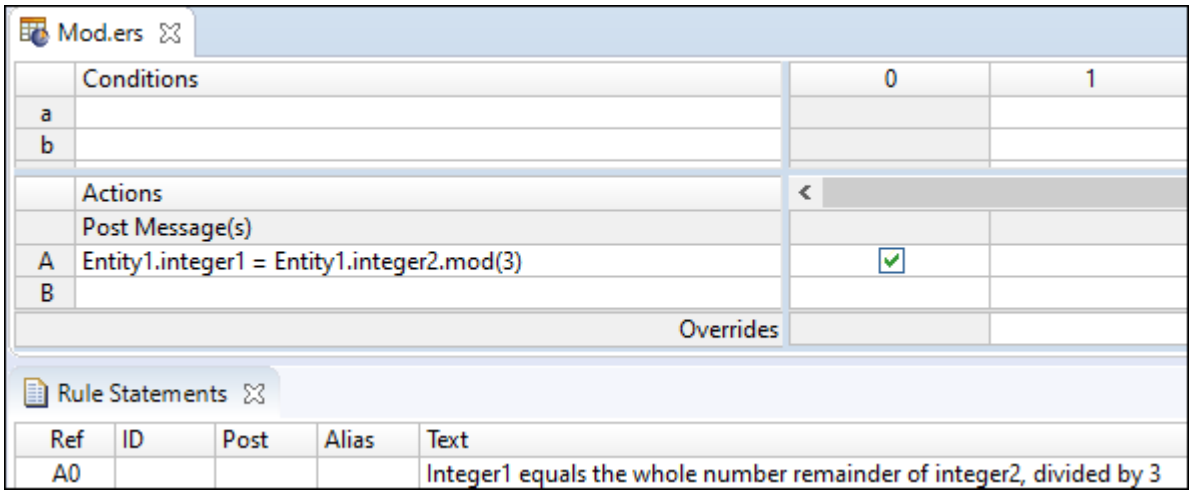
Returns the whole number remainder that results from dividing `<Integer1>` by `<Integer2>`. If the remainder is a fraction, then `0` (zero) is returned.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

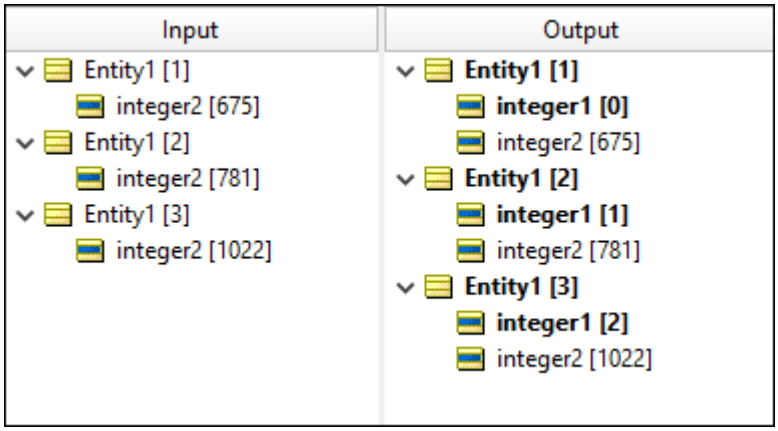
RULESHEET EXAMPLE

The following Rulesheet > uses **.mod** to calculate the whole number remainder resulting from the division of `integer2` by 3. The result is assigned to `integer1`.



SAMPLE RULETEST

A sample Ruletest provides three examples of `integer2`. Input and Output panels are shown below.



Month

SYNTAX

<DateTime>.month

<Date>.month

DESCRIPTION

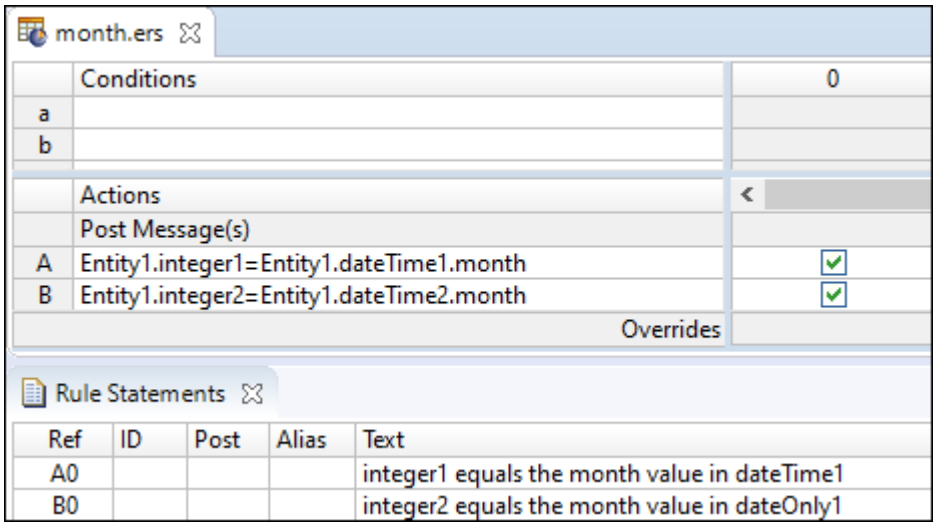
Returns the month in <DateTime> or Date as an Integer between 1 and 12.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

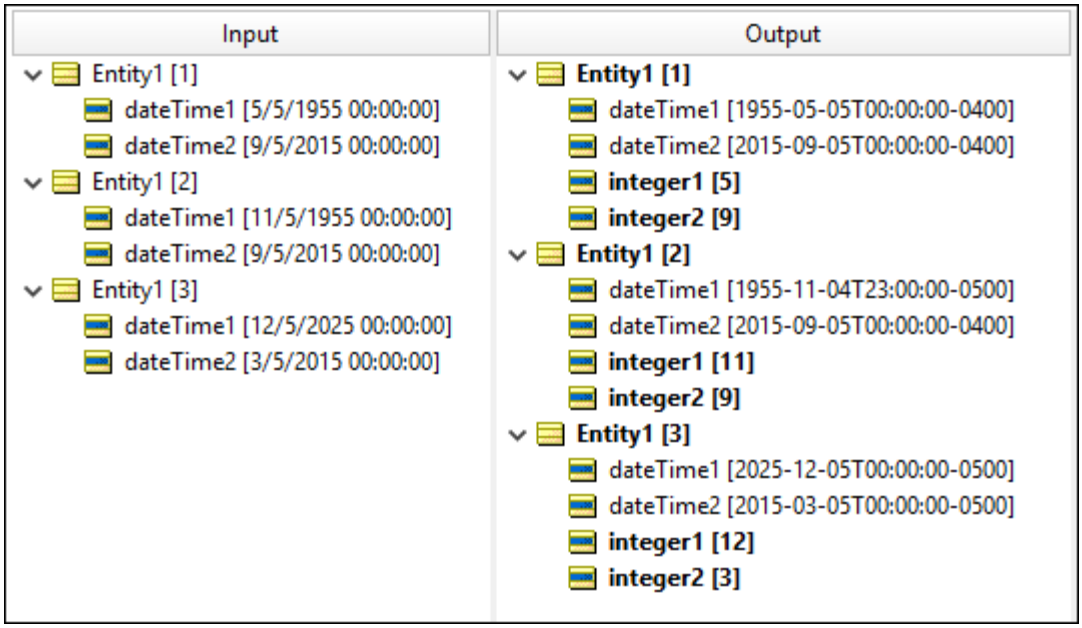
RULESHEET EXAMPLE

The following Rulesheet uses **.month** to evaluate `dateTime1` and `dateTime2` and assign the month value to `integer1` and `integer2`, respectively.



SAMPLE RULETEST

A sample Ruletest provides two examples with `dateTime1` and `dateTime2`. Input and Output panels are shown below.



Months between

SYNTAX

<DateTime1>.monthsBetween(<DateTime2>)

<Date1>.monthsBetween(<Date2>)

DESCRIPTION

Returns the Integer number of months between DateTimes or between Dates. The month and year portions of the date data are subtracted to calculate the number of elapsed months. The day portions are ignored. If the month and year portions are the same, the result is zero. This function returns a positive number if <DateTime2> is later than <DateTime1>.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.monthsBetween** to determine the number of months that have elapsed between dateTime1 and dateTime2, compare it to the values in the Condition Cells, and assign a value to string1.

monthsBetween.ers

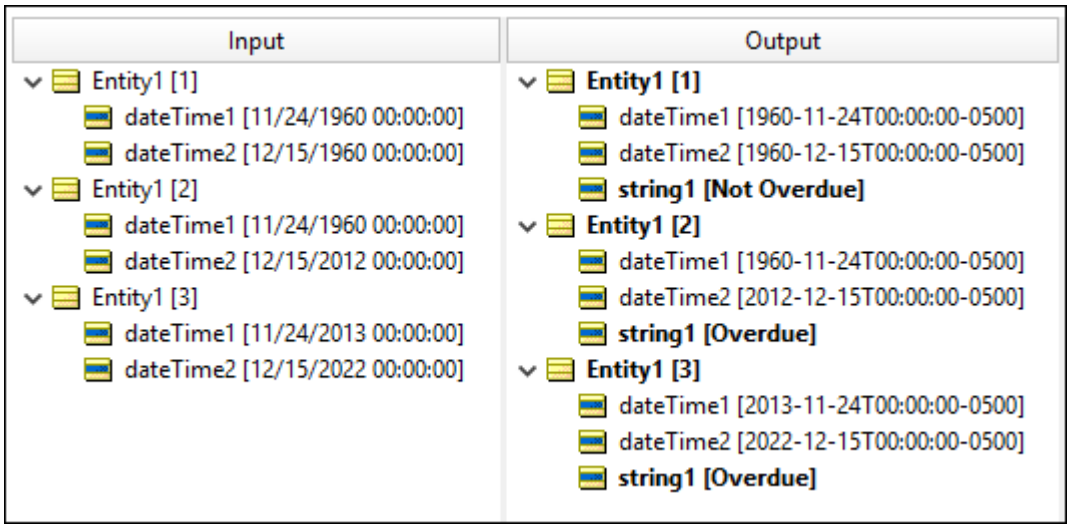
	Conditions	1	2
a	Entity1.dateTime1.monthsBetween(Entity1.dateTime2)	<= 6	> 6
b			
	Actions		
	Post Message(s)		
A	Entity1.string1	'Not Overdue'	'Overdue'
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If 6 or fewer months have elapsed between date1 and date2, then Entity1 is not overdue
2				If more than 6 months have elapsed between date1 and date2, then Entity1 is overdue

SAMPLE RULETEST

A sample Ruletest provides dateTime1 and dateTime2 for two examples. Input and Output panels are shown below.



Multiply

SYNTAX

`<Number1>` * `<Number2>`

DESCRIPTION

Multiplies `<Number1>` by `<Number2>`. The resulting data type is the more expansive of those of `<Number1>` and `<Number2>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

This sample Rulesheet uses **multiply** to multiply `integer1` and `integer2` and compare the result to `100`

Multiply.ers

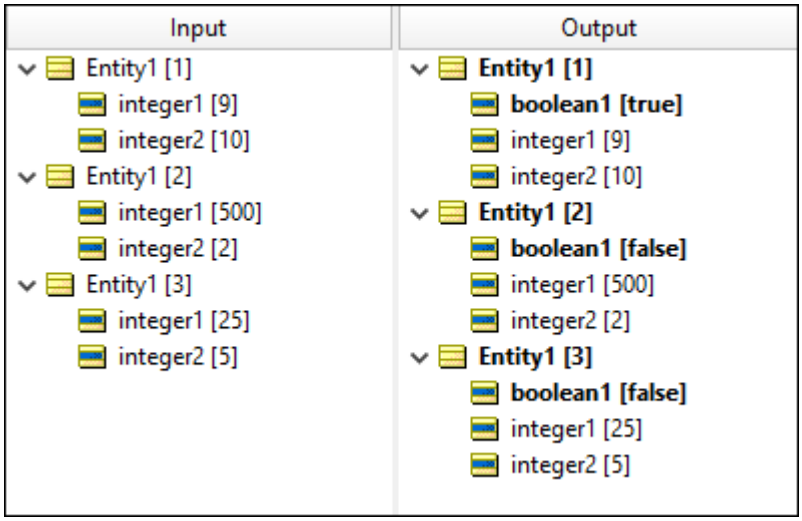
	Conditions	1	2
a	Entity1.integer1 * Entity1.integer2	< 100	>= 100
b			
	Actions		
	Post Message(s)		
A	Entity1.boolean1	T	F
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If integer1 multiplied by integer2 is less than 100, then boolean1 is true
2				If integer1 multiplied by integer2 is greater than or equal to 100, then boolean1 is false

SAMPLE RULETEST

A sample Ruletest provides three examples of `integer1` and `integer2`. Input and Output panels are shown below.



Natural logarithm

SYNTAX

`<Number>.ln`

DESCRIPTION

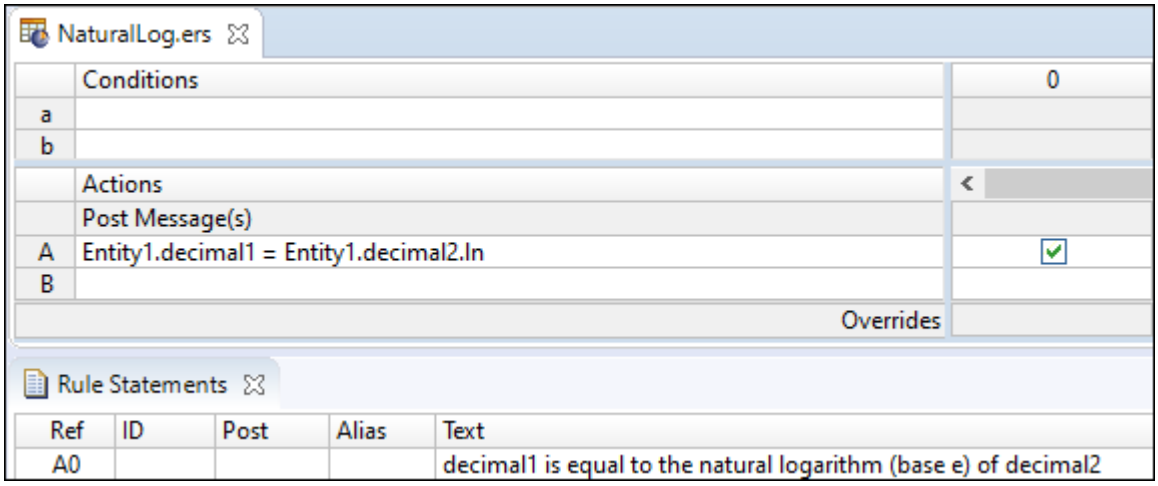
Returns a Decimal value equal to the natural logarithm (base e) of `<Number>`. If `<Number>` is equal to `0` (zero), an error is returned when the rule is executed. This error will halt execution for all data present.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

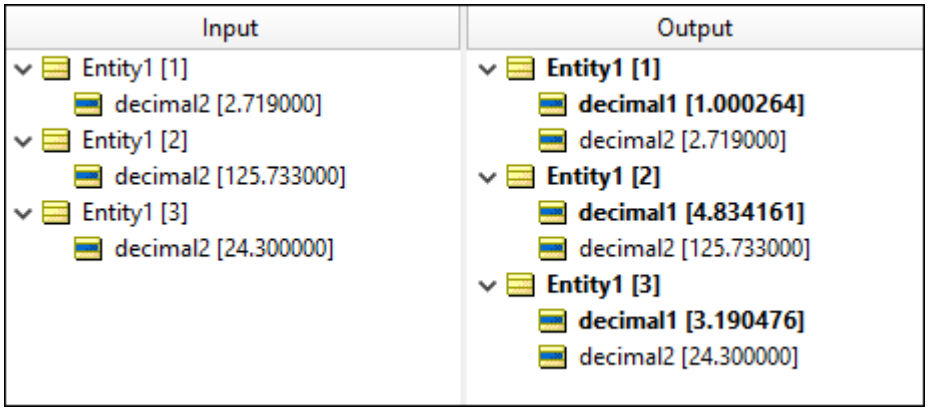
RULESHEET EXAMPLE

The following Rulesheet uses **.ln** to calculate the natural logarithm of `decimal2` and assign it to `decimal1`.



SAMPLE RULETEST

A sample Ruletest provides results for three examples of `decimal2`. Input and Output panels are shown below:



Note: In a case where the rule encounters `0.ln`, it throws an exception that halts execution. That's because the value of `0.ln` is undefined. If the rule is executing against multiple entities, the arbitrary order of execution might be different on subsequent runs before execution is halted.

New

SYNTAX

```
<Entity>.new[<Expression1>,<Expression2>...]
```

DESCRIPTION

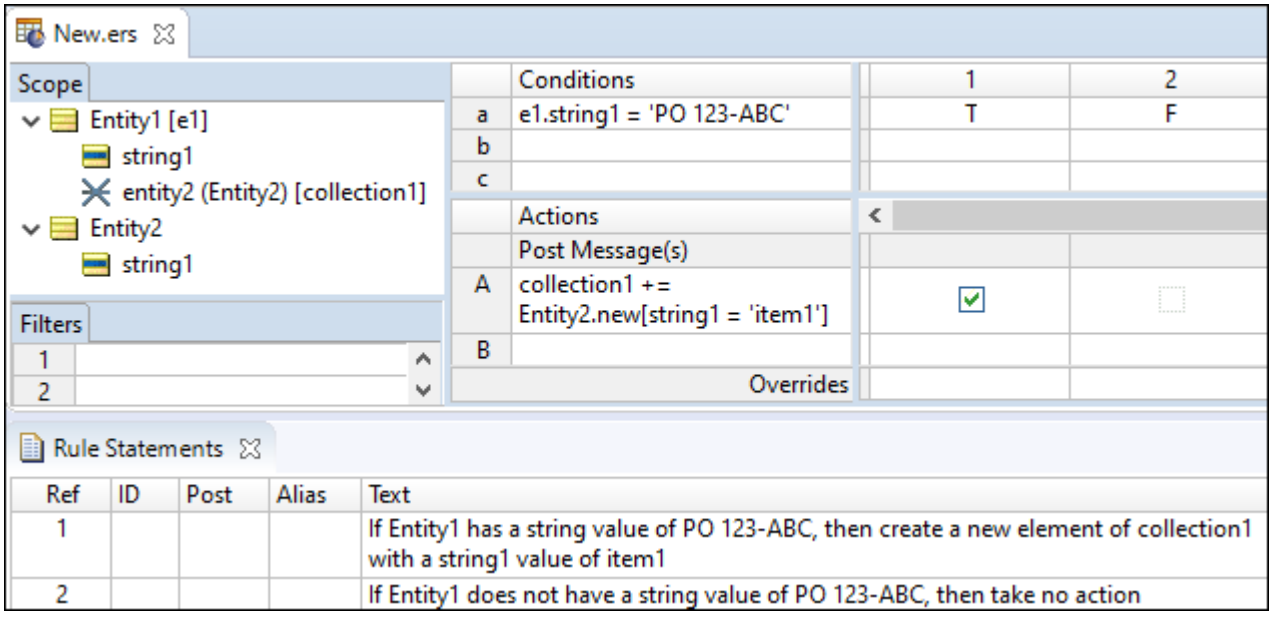
creates a new <Entity> with attribute values defined by optional <Expression>. Expressions (when present) should be written as assignments in the form: *attribute* = *value*. The attribute used in <Expression> (when present) must be an attribute of <Entity>.

USAGE RESTRICTIONS

The Operators row in the table of Summary Table of Vocabulary Usage Restriction does not apply. Special exceptions: **new** may only be used in Action Rows (section 5 in Sections of Rulesheet that correlate with usage restrictions).

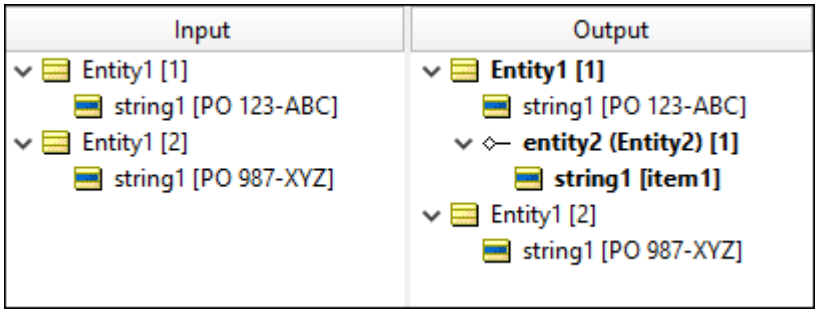
RULESHEET EXAMPLE

The following Rulesheet uses **.new** to create a new Entity2 element in collection1 when Entity1 has a string1 value equal to "PO 123-ABC". An alias is not required by the **.new** operator, because it is possible to create a new entity at the root level, without inserting it into a collection. The collection1 alias used here is required by the += (Associate Element to collection) operator.



SAMPLE RULETEST

A sample Ruletest provides 2 collections of Entity1. Input and Output panels are illustrated below:



Behavior of the .new operator

The .new operator does not consider implied conditions of non-mandatory attributes (from the initialize expressions) during execution (in other words, a .new operator always fires when explicit conditions are met).

Each initialize expression within a `.new...` expression will be executed (or not) depending upon implied conditions; that is, if any input to the expression is null, the target attribute remains null. Another case where an implied condition would prevent a `.new` operator for executing is where the new entity is a target to an association assignment and the parent of that association does not exist.

The following examples assume that all attributes are not mandatory.

- Rule 1:

  ```
  IF entity1.attr1 > 10 THEN Entity2.new[attr1 = entity1.attr2]
  ```

  Executes only if `entity1` exists, `entity1.attr1` is not null, and `entity1.attr1 > 10`. The new`Entity2.attr1` will be left as null if `entity1.attr2` is null.

- Rule 2:

  ```
  Entity2.new[attr1 = entity1.attr1 + entity1.attr2]
  ```

  Will always execute. `Entity2.attr1` will remain null if `entity1` does not exist, or `entity1.attr1` is null, or `entity1.attr2` is null.

- Rule 3:

  ```
  entity1.assoc2 += Entity2.new[attr1 = entity1.attr1]
  ```

  Will execute only if `entity1` exists. `Entity2.attr1` will remain null if `entity1.attr1` is null.

- Rule 4:

  ```
  Entity2.new[attr1 = entity1.assoc1.attr1]
  ```

  This action will always fire. `entity2.attr1` will remain null if `entity1` does not exist, or `entity1.assoc1` does not exist, or `entity1.assoc1.attr1` is null. Note that this action will fire multiple times if `entity1.assoc1` contains multiple entities (once for each entity contained in the `entity1.assoc1` collection).

New unique

SYNTAX

`<Entity>`.newUnique[`<Expression1>`,`<Expression2>`...]

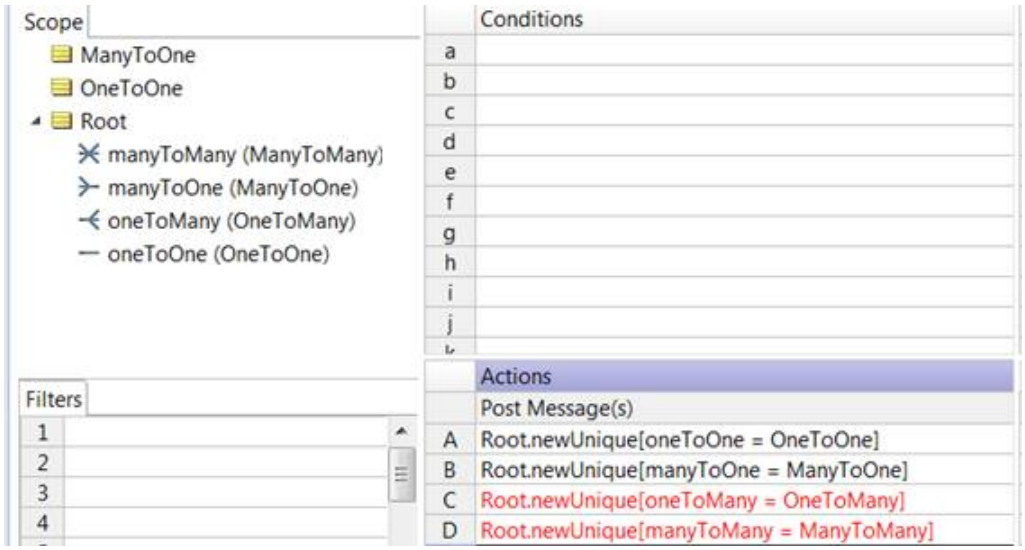
DESCRIPTION

newUnique is an unusual operator in that it contains both action *and* condition logic. When an Action containing this operator is executed, a new `<Entity>` will be created only if no other entity exists with the characteristics defined by `<Expression1>` **and** `<Expression2>`, etc. `<Expression1>` and `<Expression2>` are optional. If no expression is present within the square brackets `[..]`, the **newUnique** operator will create a new entity only if none currently exists in memory.

USAGE RESTRICTIONS

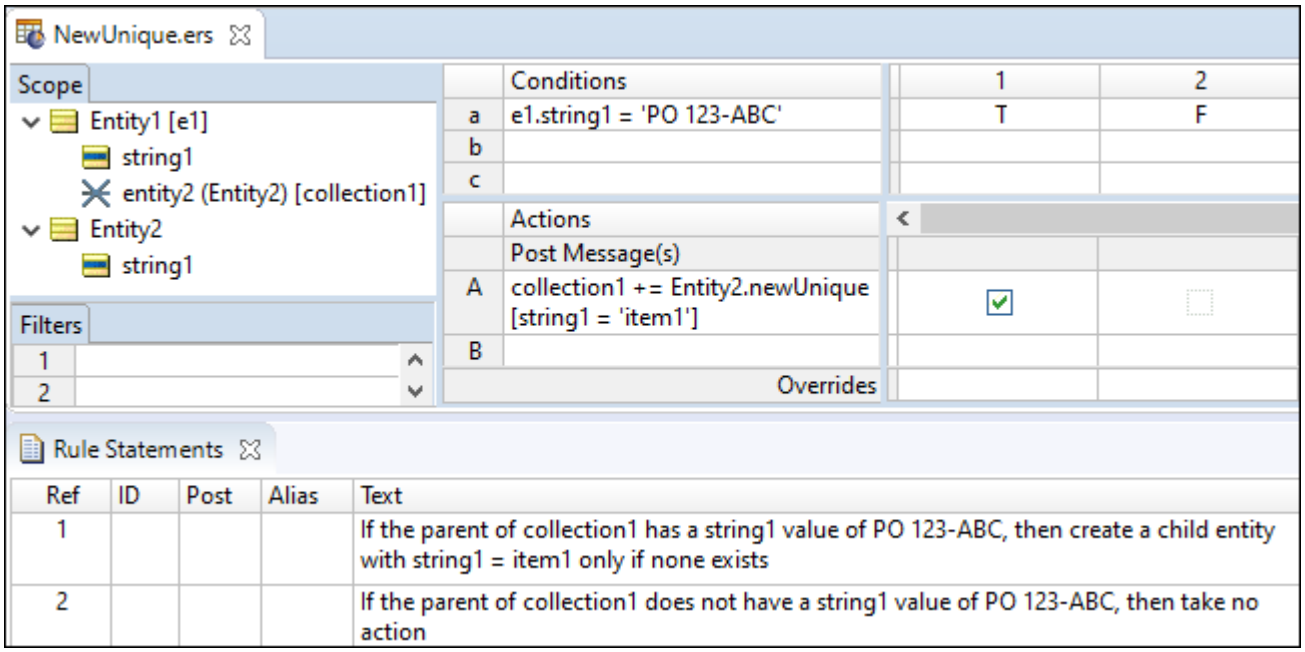
The Operators row in the table of Summary Table of Vocabulary Usage Restriction does not apply. Special exceptions: **newUnique** may only be used in Action Rows (section 5 in Sections of Rulesheet that correlate with usage restrictions).

There is some restriction to using **newUnique** with associations. **newUnique** is valid for associations of multiplicity One to One or Many to One, but is invalid for associations One to Many or Many to Many, as illustrated:



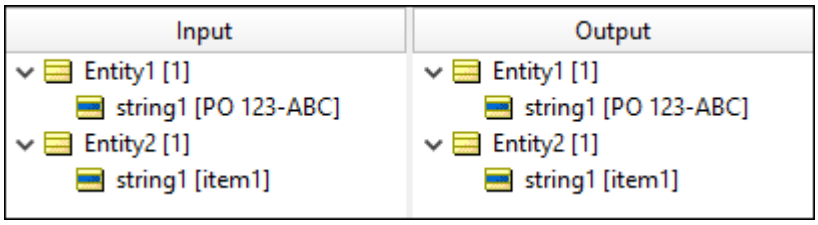
RULESHEET EXAMPLE

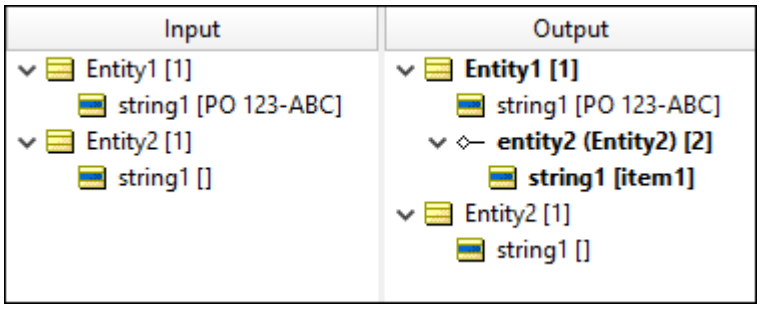
The following Rulesheet uses **.newUnique** to create a new `Entity2` element with `string1="item1"`, and add it to `collection1` only if no existing `Entity2` already has `string1="item1"`. A collection alias is not required by the **.newUnique** operator because it is possible to create a new entity at the root level, without inserting it into a collection. The collection alias used here is required by the += (Associate Element to collection) operator.

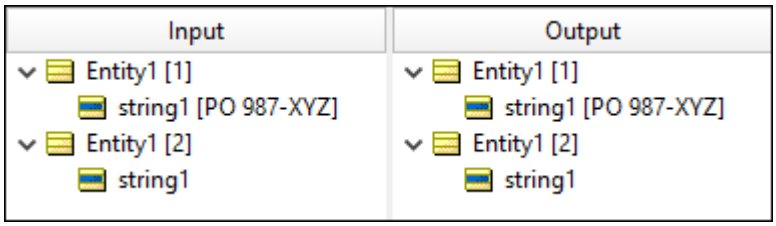


SAMPLE RULETEST 1

Each of three sample tests provides different combinations of `Entity1` and `Entity2`. Input and Output panels are illustrated below:







Not

SYNTAX

not `<Expression>`

DESCRIPTION

Returns the negation of the truth value of `<Expression>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies, with the following special exception: **not** may also be used in Conditional Cells.

RULESHEET EXAMPLE

The following Rulesheet uses **not** to negate the value of A in the Condition Cell of rule 2. **Not** may only be used in this manner if there is at least one other value (including other or null) present in the Condition Cells values drop-down list (in other words, there must be at least one alternative to the value negated by **not**).

Not.ers

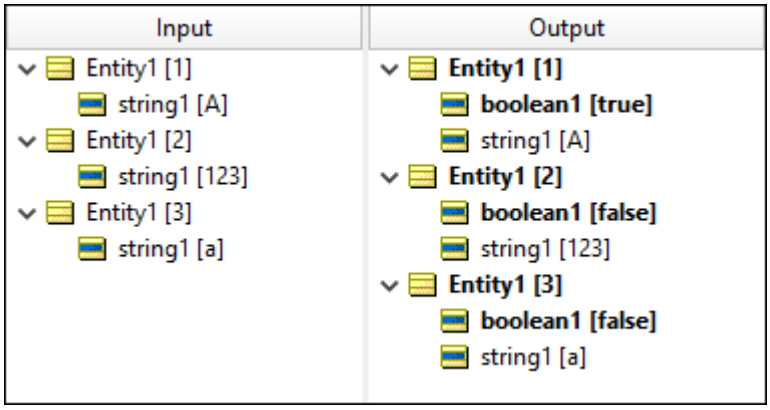
	Conditions	0	1	2
a	Entity1.string1		'A'	not 'A'
b				
	Actions			
	Post Message(s)			
A	Entity1.boolean1		T	F
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1				If string1 is equal to A, then boolean1 is assigned the value of true
2				If string1 is not equal to A, then boolean1 is assigned the value of false

SAMPLE RULETEST

A sample Ruletest provides three examples of `string1`. Input and Output panels are shown below:



Limitations to using NOT in a Conditional cell

When you use **not** in a Conditional cell with an attribute name, the form is `not` *`valueSet`* which evaluates as `true` when the condition is not a member of an entry in the *valueSet*. Such entries in the *valueSet* must be literals (or partial expressions containing only literals); no variables or attributes may be included. Inclusion of an attribute reference in the *valueSet* is not valid.

Although `not` *`attribute`* is unsupported, it is not determined that it is invalid until it does not process. Then, it indicates that it is invalid.

Consider the following examples:

Table 1: Valid usage

Condition	Cell value
`foo.color`	`not 'red'`
`foo.color`	`<> 'red'`
`foo.color`	`<> bar.color`

Table 2: Invalid usage

Condition	Cell value
`foo.color`	`not bar.color`

Not empty

SYNTAX

`<Collection>` ->notEmpty

DESCRIPTION

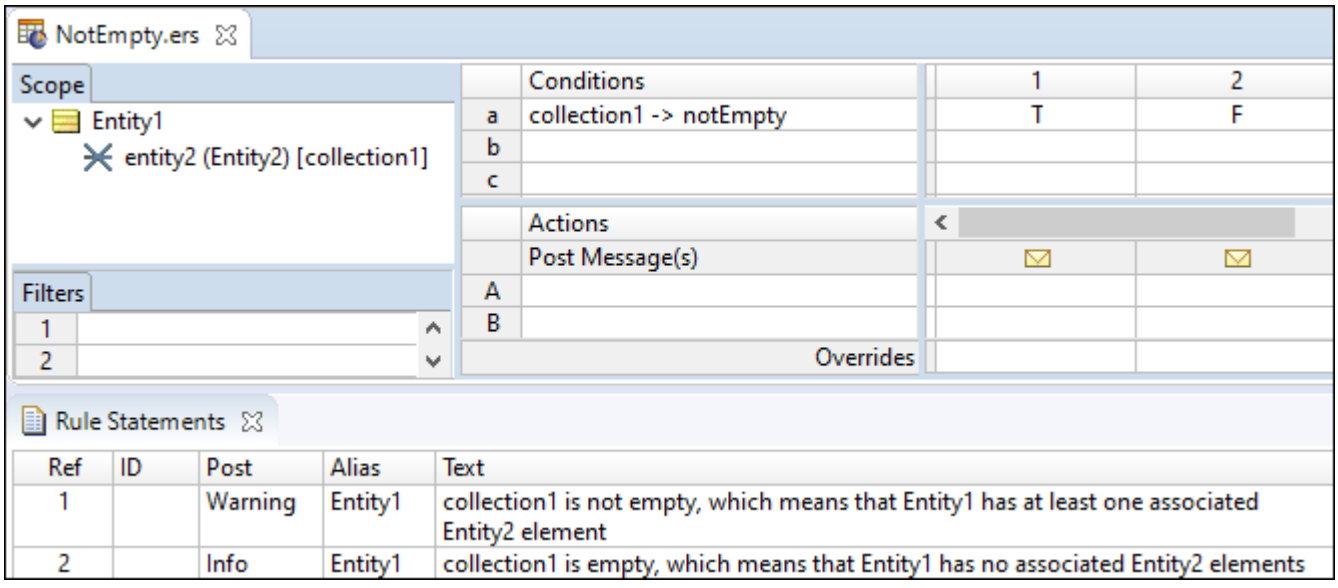
Returns a value of true if `<Collection>` contains *at least one* element. **->notEmpty** does not check for attribute values, but instead checks for the *existence of elements within a collection*. As such, it requires the use of a unique alias to represent the collection being tested.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

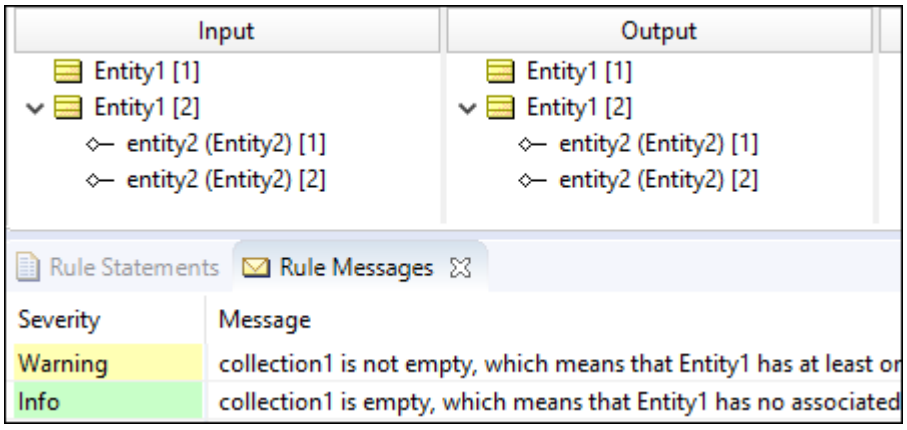
RULESHEET EXAMPLE

This sample Rulesheet uses the **->notEmpty** function to determine if `collection1` has elements. Note the use of unique alias `collection1` to represent the collection of `Entity2` associated with `Entity1`.



SAMPLE RULETEST

A sample Ruletest provides two collections. The following illustration shows Input and Output panels



Not equal to

SYNTAX

Boolean	`<Expression1> <> <Expression2>`
DateTime*	`<DateTime1> <> <DateTime2>`
Number	`<Number1> <> <Number2>`
String	`<String1> <> <String2>`

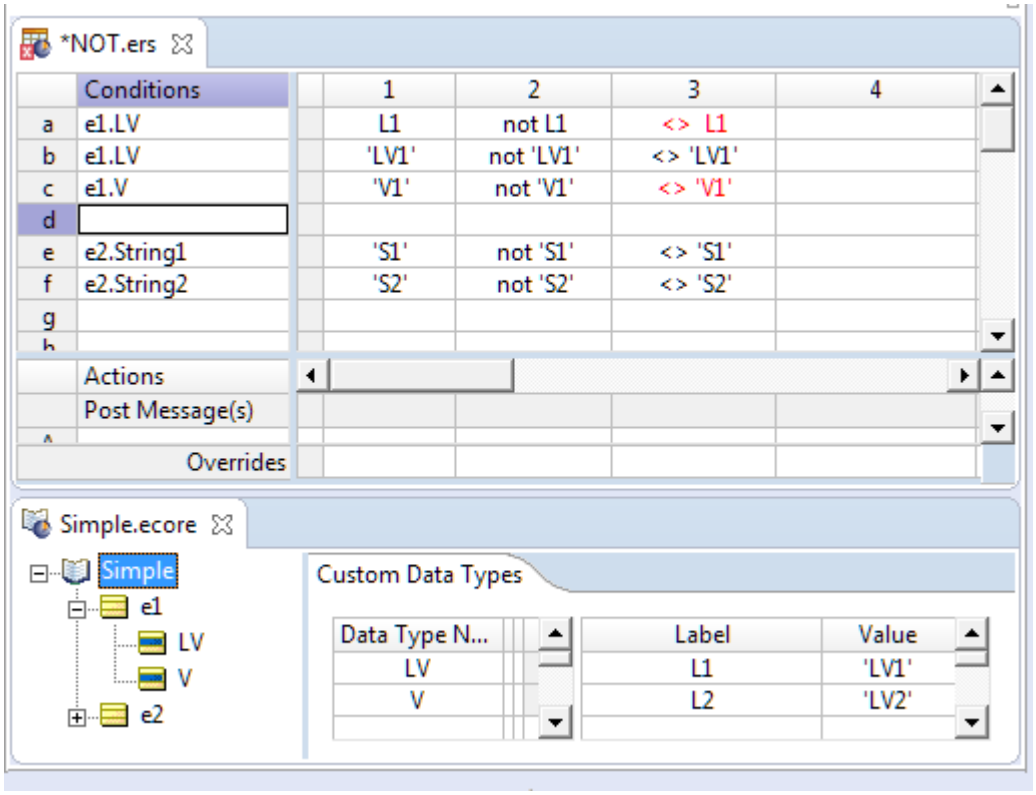
DESCRIPTION

Boolean	Returns a value of true if `<Expression1>` does not have the same truth value as `<Expression2>`.
DateTime	Returns a value of true if `<DateTime1>` does not equal `<DateTime2>`. This is equivalent to `<DateTime1>` not occurring “on” `<DateTime2>`
Number	Returns a value of true if `<Number1>` is not equal to `<Number2>`. Different numeric data types may be compared in the same expression.
String	Returns a value of true if `<String1>` is not equal to `<String2>`.

USAGE RESTRICTIONS

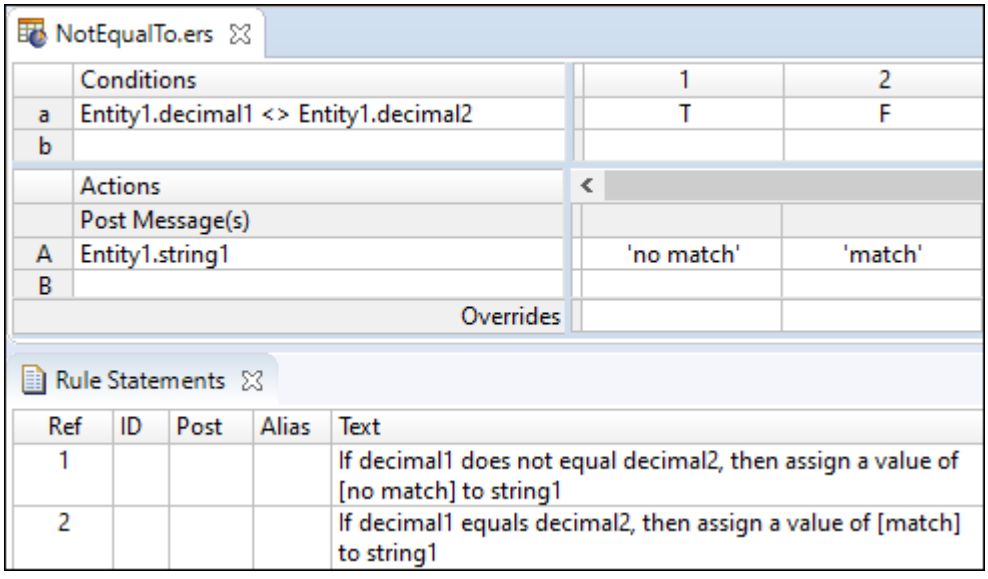
The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

Note: Use of < > when using custom data types - If your Vocabulary uses custom data types, there are limits to the validity of < > in cells. In the following illustration, the `not` operator will validly work against a custom data type label, a value where a label is in use, and the value of a value-only definition. However, only the value where a label is in use is valid when < > is used.



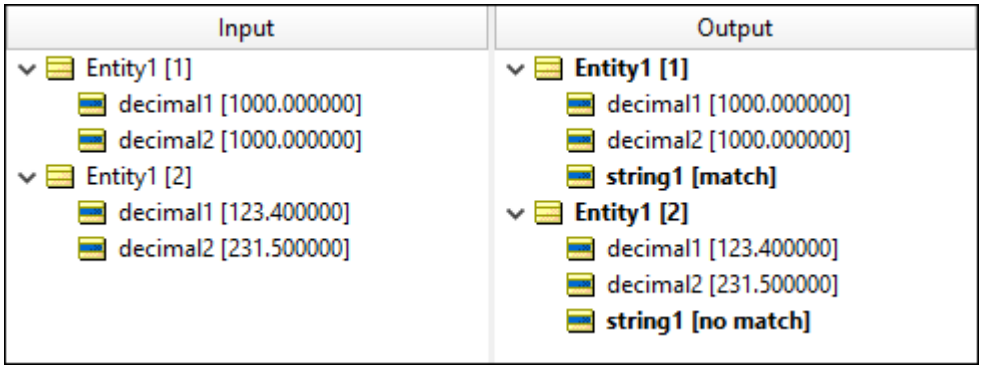
RULESHEET EXAMPLE

The following Rulesheet uses **not equal to** to test whether `decimal1` equals `decimal2`, and assign a value to `string1` based on the result of the comparison.



SAMPLE RULETEST

A sample Ruletest provides two examples. Input and Output panels are shown below:



Now

SYNTAX

now

DESCRIPTION

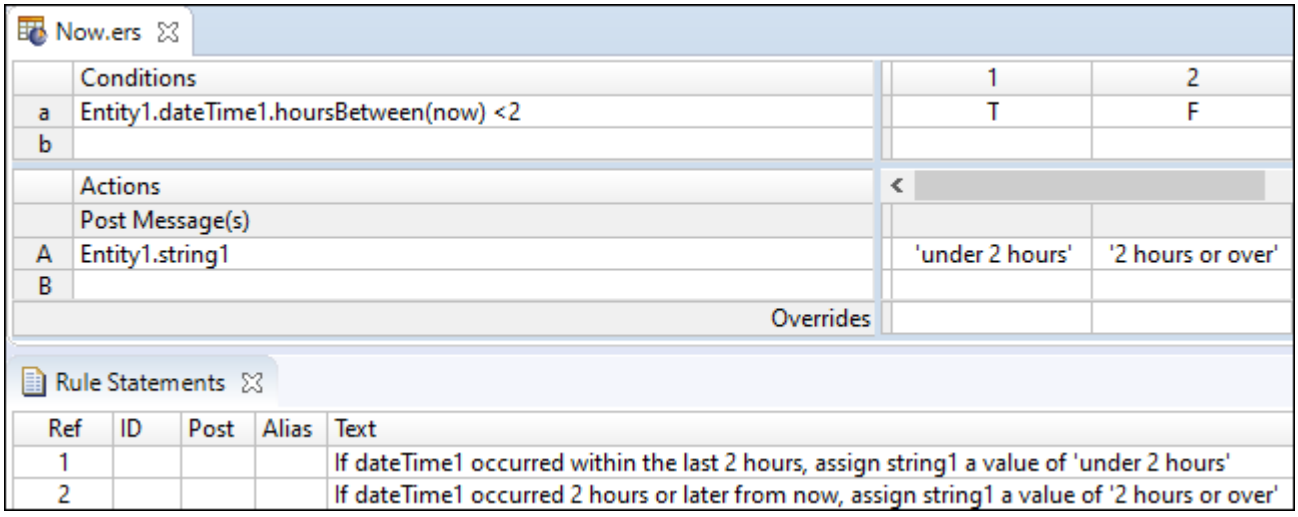
Returns the current system date and time when the rule is executed. This DateTime value is assigned the first time **now** is used in a Decision Service, then remains constant until the Decision Service finishes execution, regardless of how many additional times it is used. This means that every rule in a Ruleflow containing **now** will use the same DateTime value.

USAGE RESTRICTIONS

The Literals row in the table of Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

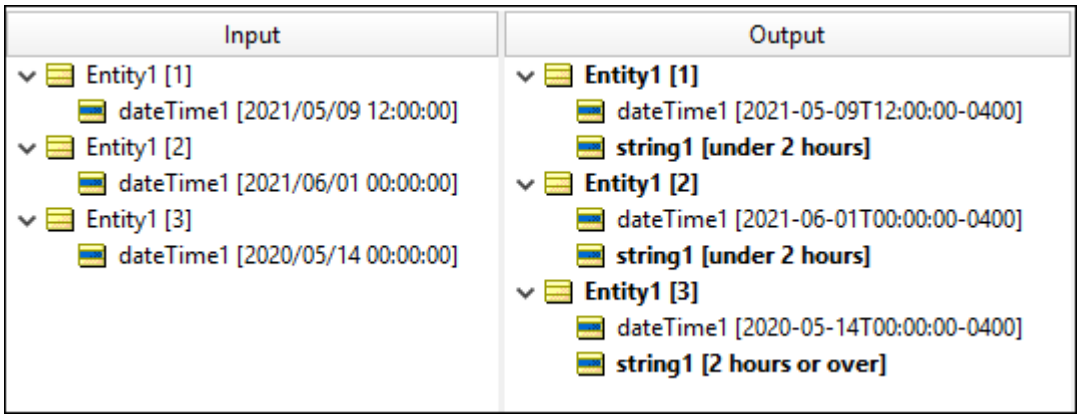
RULESHEET EXAMPLE

The following Rulesheet uses **now** to determine how many hours have elapsed between now and `dateTime1` (See .hoursBetween for more details on this operator), and assign a value to `string1` based on the result.



SAMPLE RULETEST

A sample Ruletest provides two examples of `dateTime1`. Assume **now** is equal to `May 9, 2021 14:20:00 EST`. Note that a future date in example 2 results in a negative value and therefore is under 2 hours. Input and Output panels are shown below.



Null

SYNTAX

null

DESCRIPTION

The null value corresponds to one of three different scenarios:

1. the absence of an attribute in a Ruletest Input pane or request message
2. the absence of data for an attribute in a Ruletest (the value zero counts as data)
3. a business object (supplied by an external application) that has an instance variable of null

A **null** value is different from an empty String (for String data types) or zero for numeric data types. An empty String is represented in a Ruletest as [] -- open then close square brackets. Any attribute value, including any empty strings, may be reset to **null** in a Ruletest by right-clicking the attribute and choosing **Set to null**. Mandatory attributes (property set in the Vocabulary) may not have a null value.

USAGE RESTRICTIONS

The Literals row in the table of Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **null** to test for the existence of a real value in `decimal1`, and assign a value to `boolean1` as a result.

Null.ers

	Conditions	0	1	2
a	Entity1.decimal1		null	other
b				
	Actions			
	Post Message(s)		✉	✉
A	Entity1.boolean1		T	F
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1		Warning	Entity1	If decimal1 has the value of null, then assign boolean1 the value of true
2		Info	Entity1	If decimal1 has any value other than null (any real number), then assign boolean1 the value of false

SAMPLE TEST

A sample Ruletest provides four examples of `decimal1`. Input and Output panels are illustrated below. Posted messages are not shown.

Input	Output
Entity1 [1]	**Entity1 [1]**
decimal1 [4.000000]	**boolean1 [false]**
Entity1 [2]	decimal1 [4.000000]
decimal1	**Entity1 [2]**
Entity1 [3]	**boolean1 [true]**
decimal1 [0.000000]	decimal1
Entity1 [4]	**Entity1 [3]**
decimal1 [-13.000000]	**boolean1 [false]**
	decimal1 [0.000000]
	Entity1 [4]
	boolean1 [false]
	decimal1 [-13.000000]

Rule Statements | Rule Messages

Severity	Message
Warning	If decimal1 has the value of null, then assign boolean1 the value of true
Info	If decimal1 has any value other than null (any real number), then assign
Info	If decimal1 has any value other than null (any real number), then assign
Info	If decimal1 has any value other than null (any real number), then assign

Other

SYNTAX

other

DESCRIPTION

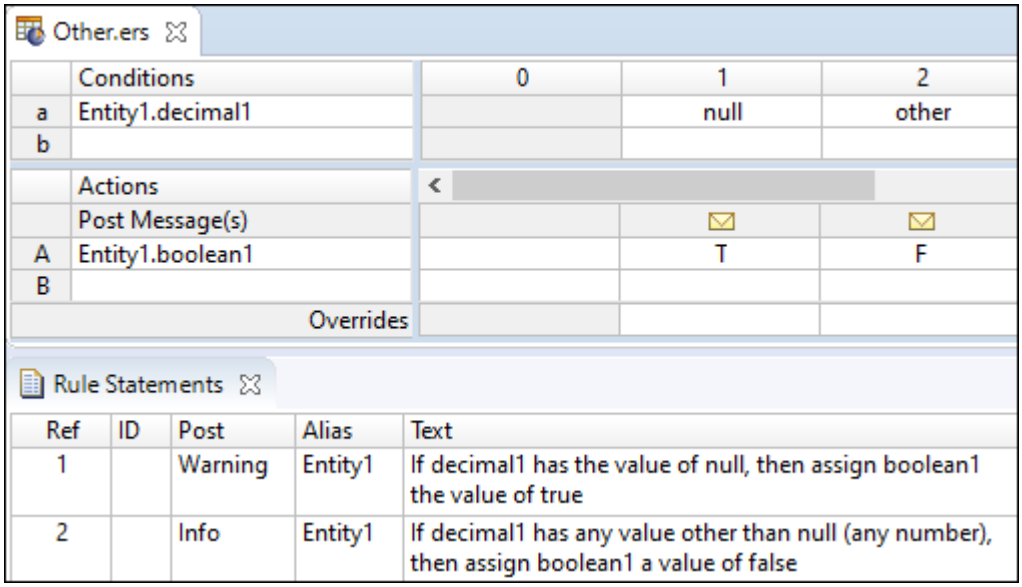
When included in a condition's Values set (the drop-down list of values available in a Conditions Cell), **other** represents any value not explicitly included in the set, including null. If null is explicitly included in the Values set, then **other** does not include null.

USAGE RESTRICTIONS

The Literals row in the table of Summary Table of Vocabulary Usage Restriction does not apply. Special exception: **other** may only be used in Condition Cells (section 4 of the Sections of Rulesheet that correlate with usage restrictions) because it is a non-specific value used in comparisons.

RULESHEET EXAMPLE

The following Rulesheet uses **other** to test the value of `decimal1`. If `decimal1` has any value *other* than null, `boolean1` is assigned the value of `false`.



Other.ers

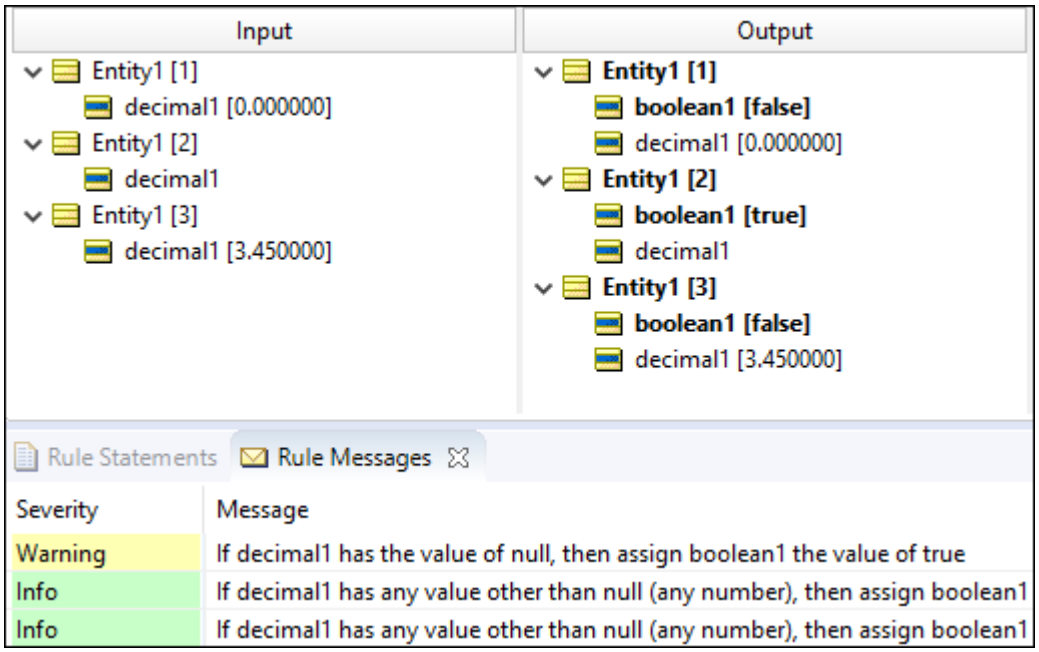
	Conditions	0	1	2
a	Entity1.decimal1		null	other
b				
	Actions			
	Post Message(s)		✉	✉
A	Entity1.boolean1		T	F
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1		Warning	Entity1	If decimal1 has the value of null, then assign boolean1 the value of true
2		Info	Entity1	If decimal1 has any value other than null (any number), then assign boolean1 a value of false

SAMPLE TEST

A sample Ruletest provides three examples of `decimal1`. Ruletest Input and Output panels are shown below:



Or

SYNTAX

`<Expression1>` or `<Expression2>` or ….

DESCRIPTION

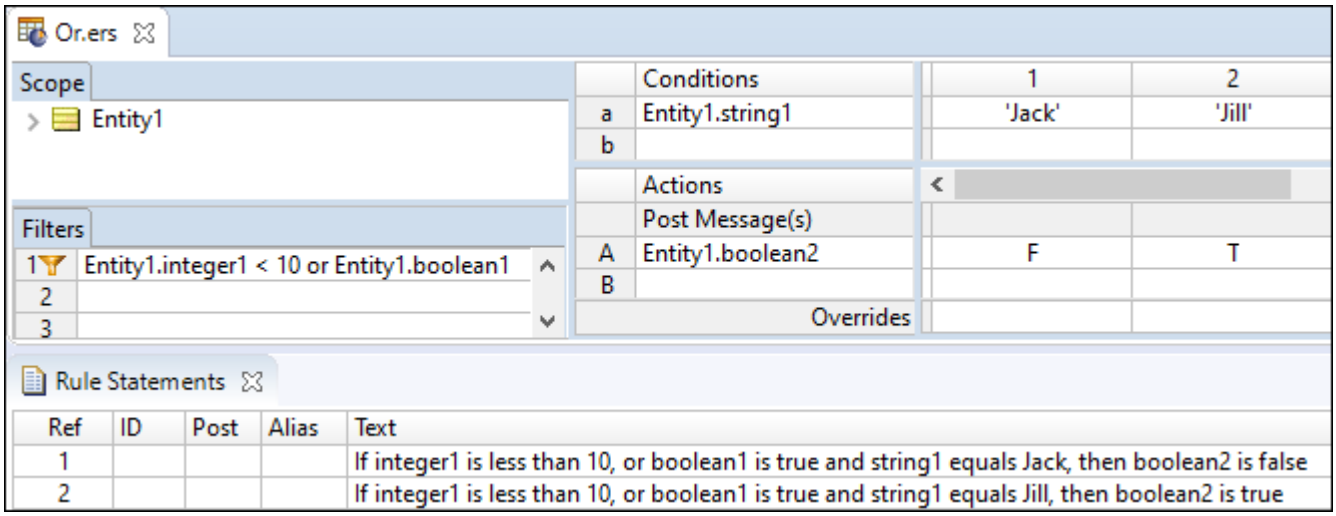
Returns a value of true if either `<Expression1>` or `<Expression2>` evaluates to true. When used between two or more expressions in the Preconditions section, creates a compound filter for the Rulesheet that follows. See *Rule Modeling Guide* for details on using Preconditions as filters. **OR** is not available in the Conditions section because the logical **OR** construction is implemented using multiple Columns in the decision table, or by value sets in Conditions Cells.

USAGE RESTRICTIONS

The Literals row in the table of Sections of Rulesheet that correlate with usage restrictions does not apply. Special exception: **or** may only be used in the Filters section of the Rulesheet to join 2 or more expressions, as shown above.

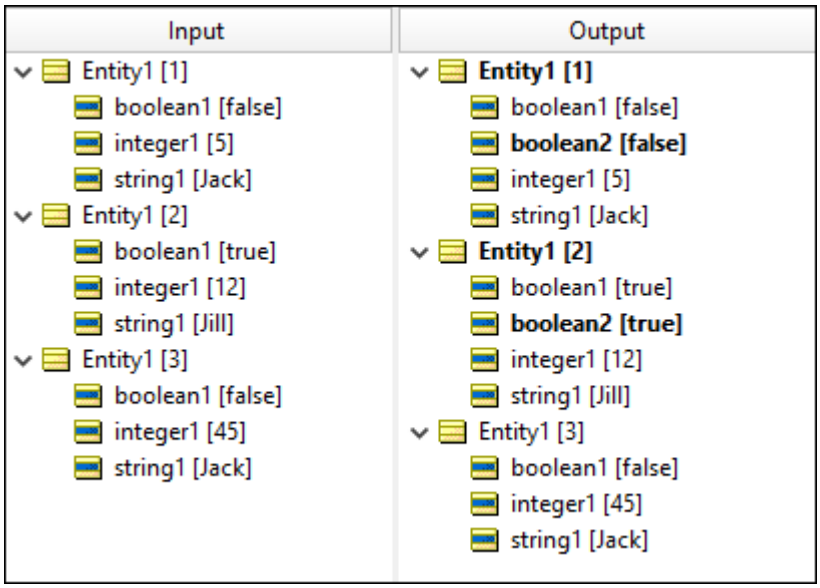
RULESHEET EXAMPLE

The following Rulesheet uses **or** to test the value of `integer1`, `boolean1`, and `string1` to set the value of `boolean2`



SAMPLE TEST

A sample Ruletest provides three examples. Input and Output panels are shown below:



Random

SYNTAX

```
<IntegerAttribute>.random(minRange,maxRange)
```

```
<DecimalAttribute>.random(minRange,maxRange)
```

DESCRIPTION

Returns a random value between `minRange` and `maxRange`. Either range can be a numeric value of the same datatype, or numeric attributes of the same type; in which case, the attributes can have arithmetic operators, absoluteValue, and unary negative applied.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLES

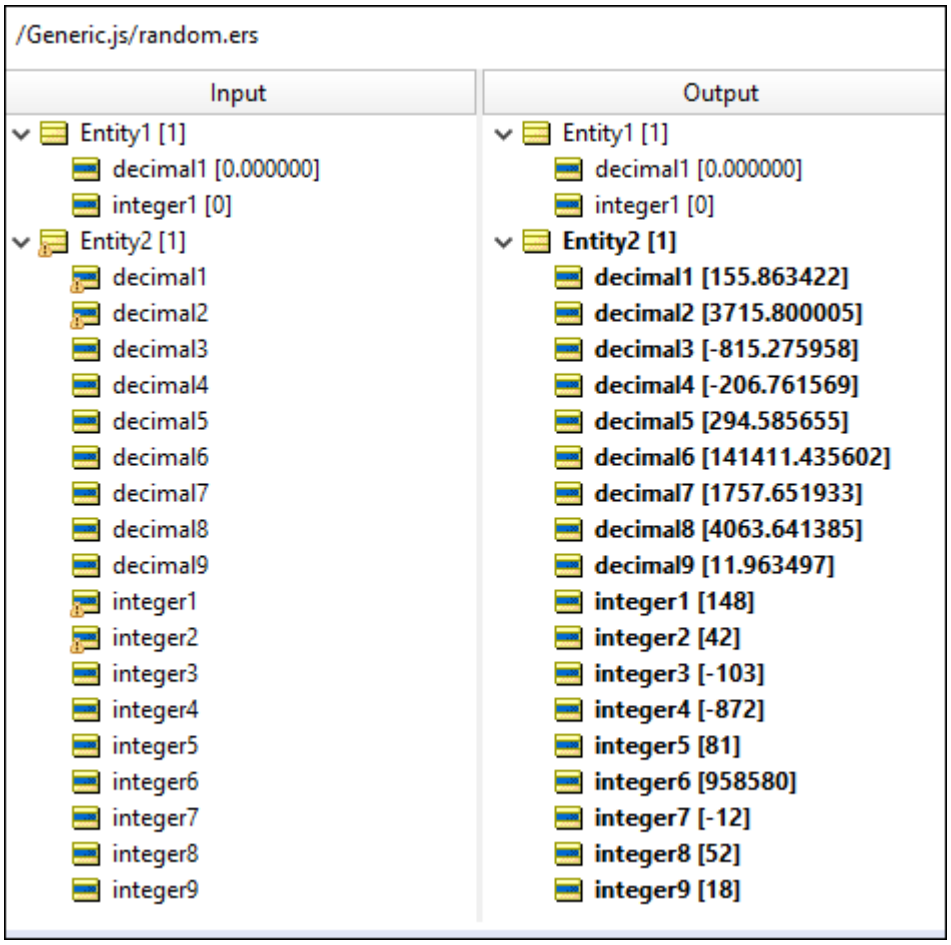
This sample Rulesheet uses **random** in non-conditional actions:

random.ers

	Conditions	0
a		
	Actions	
	Post Message(s)	
A	Entity2.decimal1=Entity1.decimal1.random(100.0,200.0)	✓
B	Entity2.decimal2=Entity1.decimal1.random(0.0,10000.0)	✓
C	Entity2.decimal3=Entity1.decimal1.random(-1000.0,-100.0)	✓
D	Entity2.decimal4=Entity1.decimal1.random(-1000.0,Entity2.decimal2)	✓
E	Entity2.decimal5=Entity1.decimal1.random(0.0,Entity2.decimal3.absVal)	✓
F	Entity2.decimal6=Entity1.decimal1.random(Entity2.decimal1, 1000000.0)	✓
G	Entity2.decimal7=Entity1.decimal1.random(Entity2.decimal3,Entity2.decimal2)	✓
H	Entity2.decimal8=Entity1.decimal1.random(Entity2.decimal3*10.0,Entity2.decimal2*10.0)	✓
I	Entity2.decimal9=Entity1.decimal1.random(20.0,10.0)	✓
J		
K	Entity2.integer1=Entity1.integer1.random(100,200)	✓
L	Entity2.integer2=Entity1.integer1.random(0,100)	✓
M	Entity2.integer3=Entity1.integer1.random(-105,-100)	✓
N	Entity2.integer4=Entity1.integer1.random(-1000,Entity2.integer2)	✓
O	Entity2.integer5=Entity1.integer1.random(0,Entity2.integer3.absVal)	✓
P	Entity2.integer6=Entity1.integer1.random(Entity2.integer1, 1000000)	✓
Q	Entity2.integer7=Entity1.integer1.random(Entity2.integer3,Entity2.integer2)	✓
R	Entity2.integer8=Entity1.integer1.random(Entity2.integer3*10,Entity2.integer2*10)	✓
S	Entity2.integer9=Entity1.integer1.random(20,10)	✓
T		
	Overrides	

SAMPLE RULETEST

A sample Ruletest requires values for Entity1 although they have no impact on the output. As the result is random, there cannot be an expected value.



Regular expression to replace String

SYNTAX

`<String>`.regexReplaceString(`regularExpression`,`replacementString`)

`<String>`.regexReplaceString(`regularExpression`,`replacementString`,regexFlag:`<String>`)

DESCRIPTION

Returns a new String where the strings matching the regular expression are replaced by the replacement string.

Note: Regular expressions are a well-established technique that uses a sequence of characters to define a search pattern. For more information, see Wikipedia, as well one of the many sites that provide examples, such as `regular-expresssions.info`, and others that analyze the expressions you create.

FLAGS

The characters `gis`, when one or more are added to the expression with no separator, represent:

- g global, replace more than first match (the default)
- `i` ignore case
- s match line terminator ("newline") characters in a string, which it would not match otherwise. See https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp/dotAll.

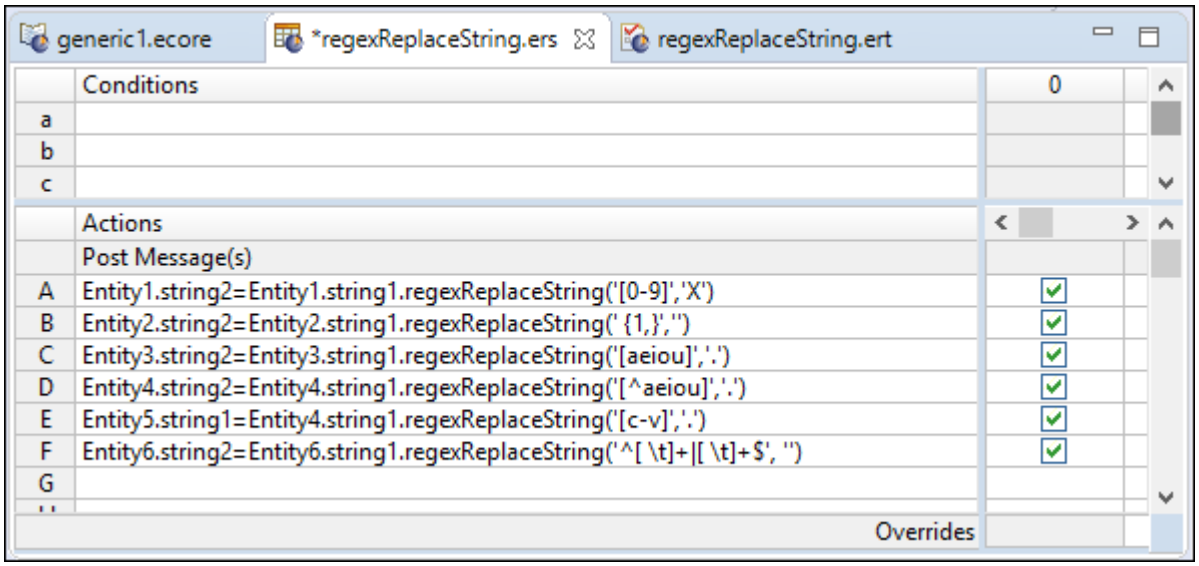
USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

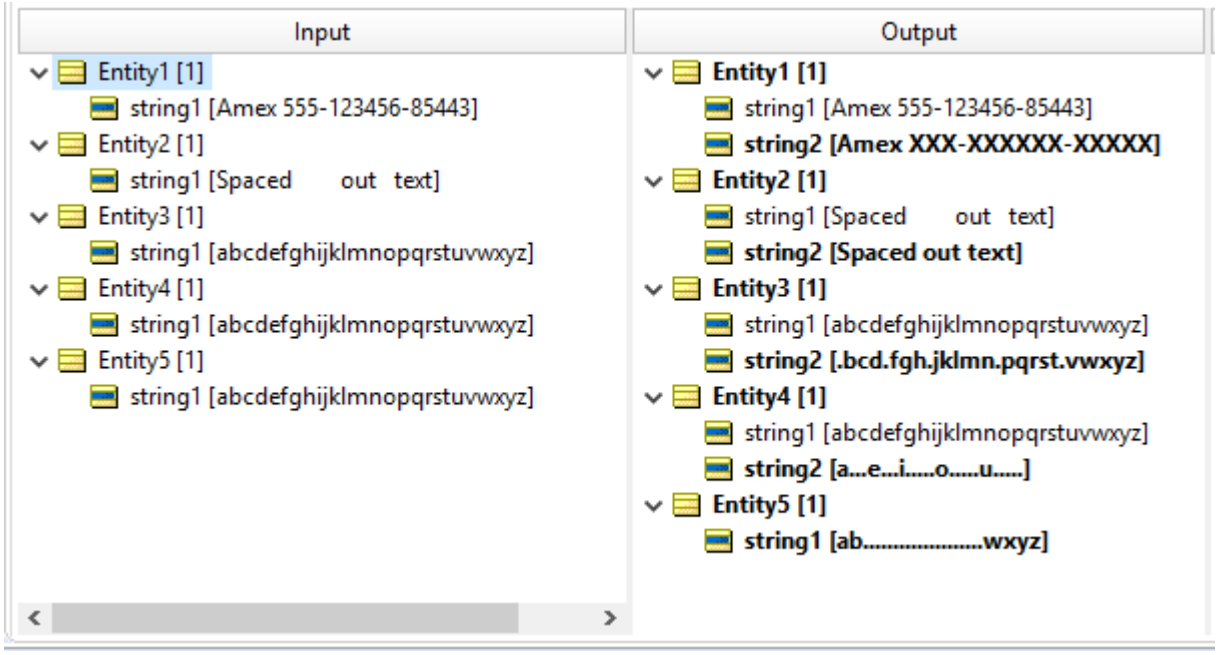
This sample Rulesheet uses **regexReplaceString** in non-conditional actions as follows:

- `regexReplaceString("[0-9]", "X")`
- `regexReplaceString(" {2,}", " "): Replace all instances of digits with the character X` - Replace all instances of multiple spaces with a single space
- `regexReplaceString("[aeiou]", ".")` - Replace all vowels with a dot .
- `regexReplaceString("[^aeiou]", ".")` - Replace all non-vowel characters with a dot.
- `regexReplaceString("[c-v]", ".")` - Replace each character in the range from c to v with a dot.
- `regexReplaceString('^[ \t]+|[ \t]+$', '')` - Strip off leading and trailing spaces.



SAMPLE RULETEST

A sample Ruletest shows the `regexReplaceString` effect in output.



Remove element

SYNTAX

`<Entity>`.remove

`<Collection>`.remove

DESCRIPTION

Removes `<Entity>` or removes elements from `<Collection>` and deletes it/them. If removing from a collection, then using a unique alias to represent the collection is optional since **.remove** is not a collection operator. If any elements in `<Collection>` have one-to-many associations with other entities, then those entities will also be deleted.

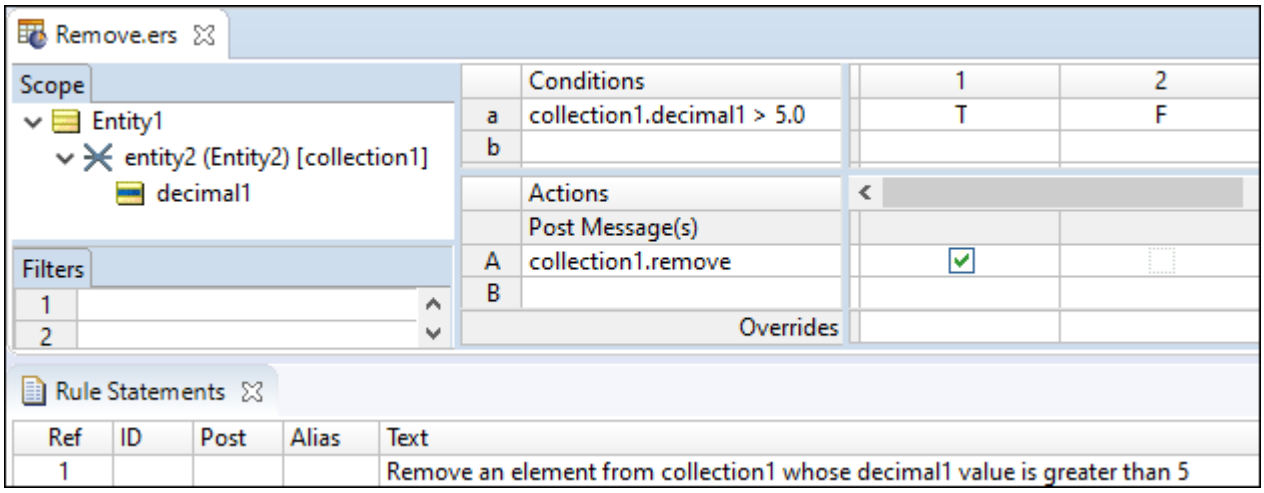
USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction does not apply. Special exceptions: **.remove** may only be used in Action Rows (section 5 in Sections of Rulesheet that correlate with usage restrictions).

EXAMPLE 1: Remove an element from a collection

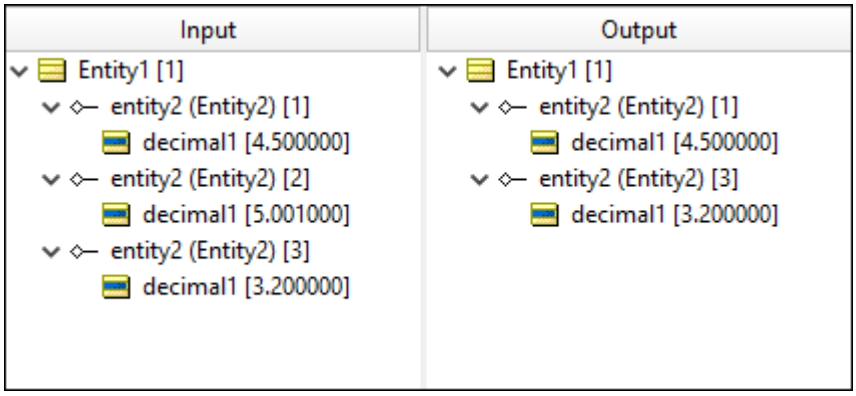
RULESHEET 1

This Rulesheet uses the operator to remove elements from `collection1` whose `decimal1` value is greater than `5`. Note the *optional* use of unique alias `collection1` to represent the collection of `Entity2` elements associated with `Entity1`.



RULETEST 1

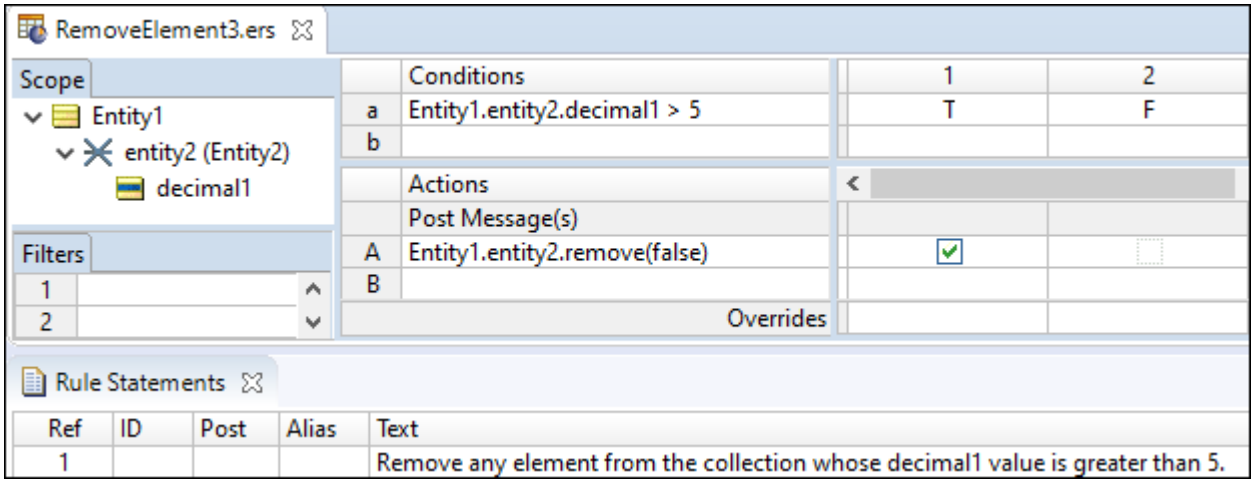
A sample Ruletest provides a collection with two elements. The illustration shows Ruletest Input and Output panels



EXAMPLE 2: Remove an entity then promote its children

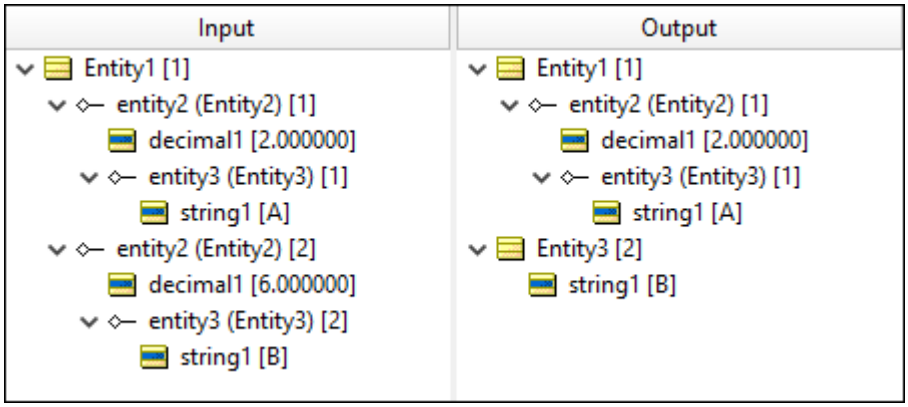
RULESHEET 2

This Rulesheet uses the operator with its `(false)` parameter to remove only the specified elements from `Entity1.entity2` whose `decimal1` value is greater than `5`. Note no unique alias has been used to represent the collection of `Entity2` elements associated with `Entity1`.



RULETEST 2

A sample Ruletest provides an `Entity1` with two `entity2`, each of which has an `entity3` child of its own. The illustration shows Ruletest Input and Output panels. Note that when an `entity2` is removed, its associated `entity3` is promoted to root level.



Replace elements

SYNTAX

`<Collection1>` = `<Collection2>`

`<Collection>` = `<Entity>`

DESCRIPTION

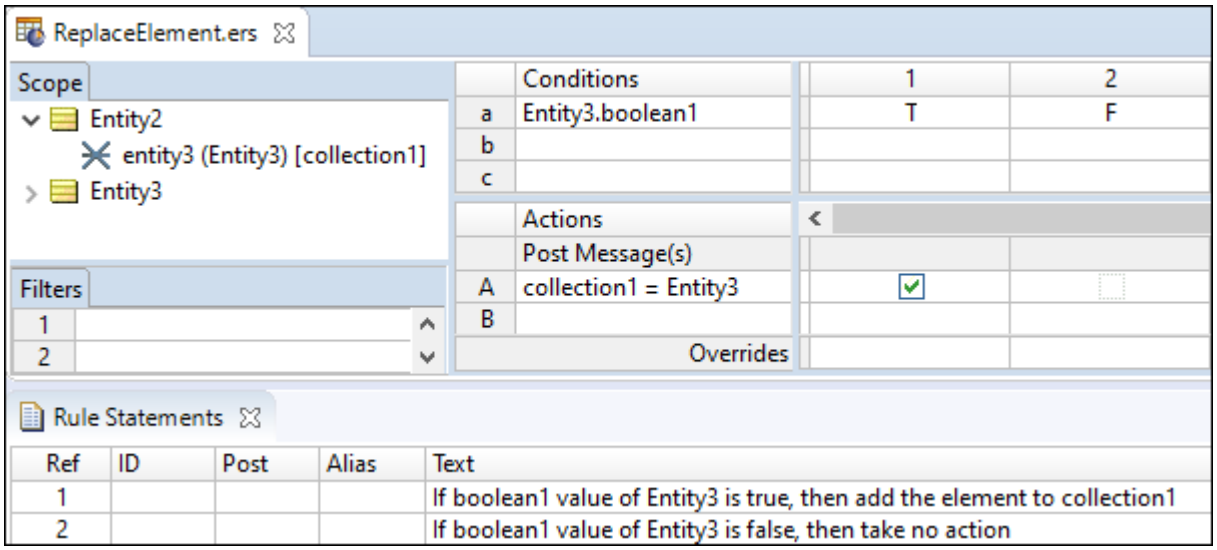
Replaces all elements in `<Collection1>` with the elements in `<Collection2>`, provided the association between the two is permitted by the Business Vocabulary. In the second syntax, `<Entity>` is associated with `<Collection>`, replacing the `<Entity>` already associated, when the association between the two is "one-to-one" in the Business Vocabulary. All collections must be expressed as unique aliases.

USAGE RESTRICTIONS

The Operators row in the table of Summary Table of Vocabulary Usage Restriction does not apply. Special exceptions: **replace elements** may only be used in Action Rows (section 5 in Sections of Rulesheet that correlate with usage restrictions).

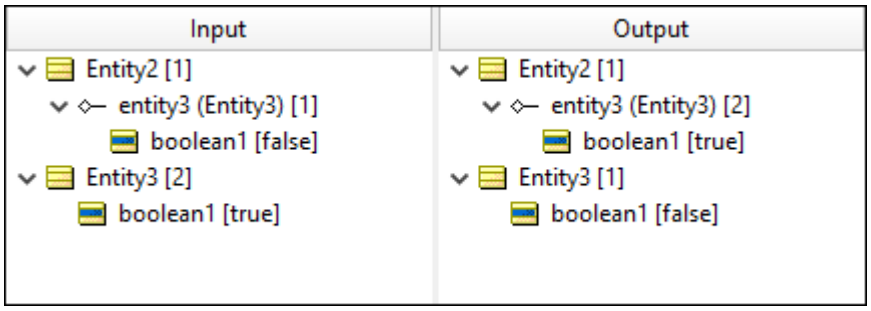
RULESHEET EXAMPLE

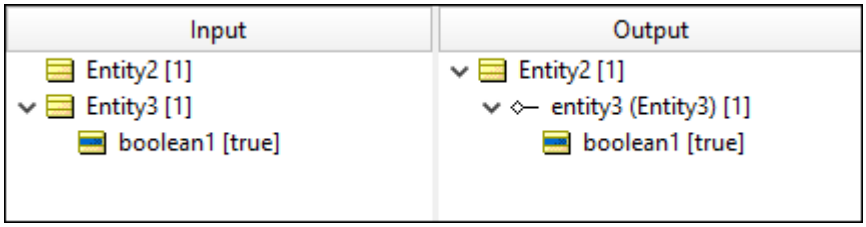
This sample Rulesheet uses the **replace element** operator to add `Entity3` to `collection1` if its `boolean1` value is `true`. Note the use of unique alias `collection1` to represent the collection of `Entity3` elements associated with `Entity2`. The association between `Entity2` and `Entity3` has a cardinality of "one-to-one". If multiple `Entity3` are present, only one will be added to `collection1`.

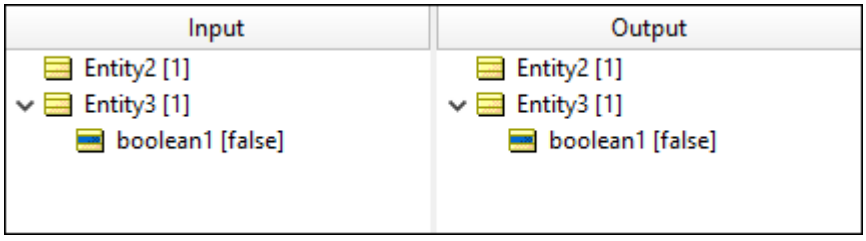


SAMPLE TEST

Three sample tests provide scenarios of two elements which share a one-to-one association. Input and Output panels are illustrated below:







Replace String

SYNTAX

<String>.replaceString(stringToBeReplaced,replacementString)

DESCRIPTION

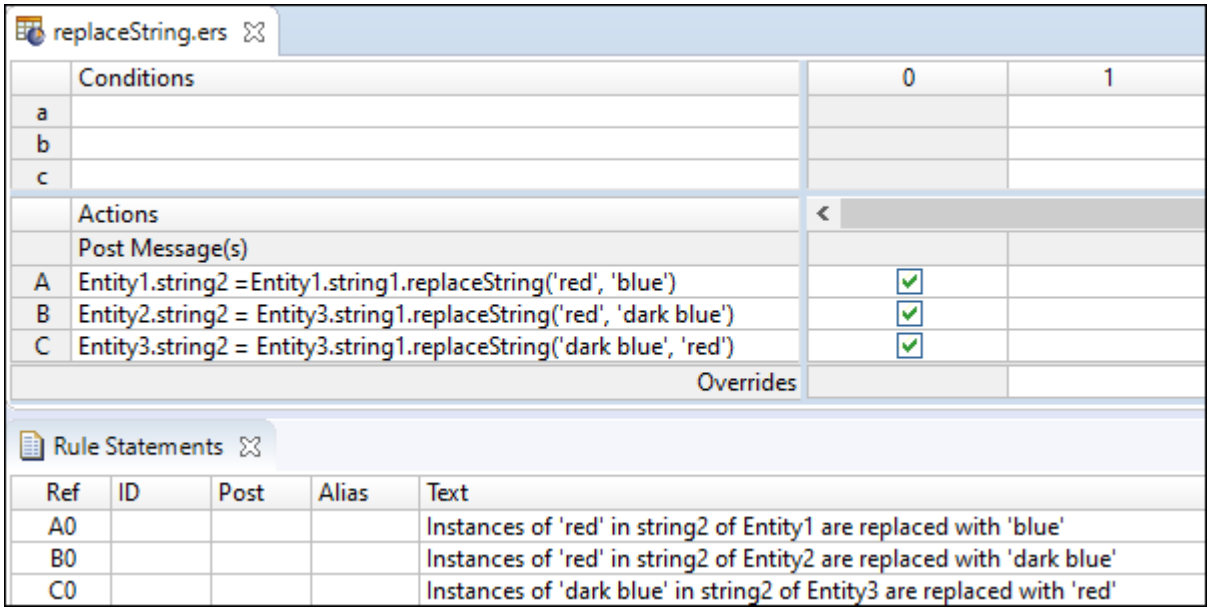
Returns a new string where the instances of the String to be replaced are replaced by the value of the replacement String.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

This sample Rulesheet uses **replaceString** in non-conditional actions.



replaceString.ers

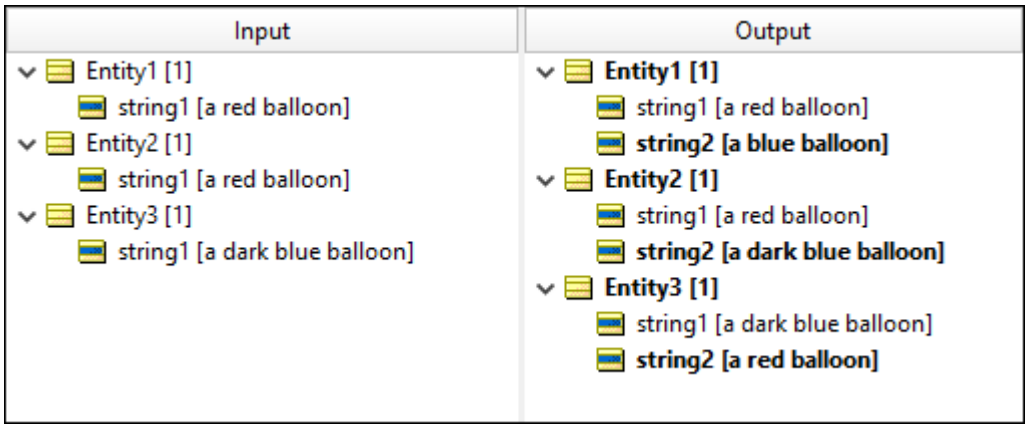
	Conditions	0	1
a			
b			
c			
	Actions		
	Post Message(s)		
A	Entity1.string2 =Entity1.string1.replaceString('red', 'blue')	☑	
B	Entity2.string2 = Entity3.string1.replaceString('red', 'dark blue')	☑	
C	Entity3.string2 = Entity3.string1.replaceString('dark blue', 'red')	☑	
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
A0				Instances of 'red' in string2 of Entity1 are replaced with 'blue'
B0				Instances of 'red' in string2 of Entity2 are replaced with 'dark blue'
C0				Instances of 'dark blue' in string2 of Entity3 are replaced with 'red'

SAMPLE RULETEST

A sample Ruletest shows the `replaceString` effect in output.



Round

SYNTAX

`<Decimal>`.round

`<Decimal>`.round(`<Integer>`)

`<Decimal>`.round(`<Integer>`, roundingMode: `<Integer>`)

DESCRIPTION

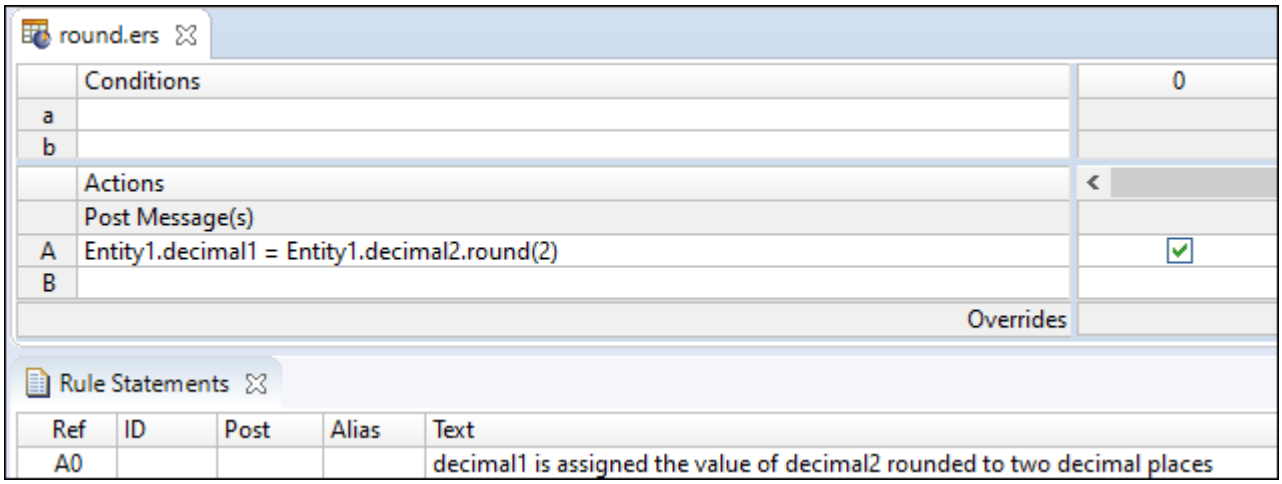
- When `<Integer>` is not provided, `<Decimal>` is rounded to the nearest whole number of type Decimal.
- When `<Integer>` is provided , `<Decimal>` is rounded to the number of decimal places specified by `<Integer>`. Standard rounding conventions apply, meaning numbers ending with significant digits of 5 or more round up and numbers ending with significant digits less than 5 round down.
- When `<Integer>` and roundingMode: `<Integer>` are provided , then `<Decimal>` is rounded to the number of decimal places specified by the mode:
  - 0 - Rounds away from zero
  - 1 - Rounds towards zero
  - 2 - Rounds towards Infinity
  - 3 - Rounds towards -Infinity
  - 4 - Rounds towards nearest neighbor; if equidistant, rounds away from zero
  - 5 - Rounds towards nearest neighbor; if equidistant, rounds towards zero
  - 6 - Rounds towards nearest neighbor; if equidistant, rounds towards even neighbor
  - 7 - Rounds towards nearest neighbor; if equidistant, rounds towards Infinity
  - 8 - Rounds towards nearest neighbor; if equidistant, rounds towards -Infinity

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

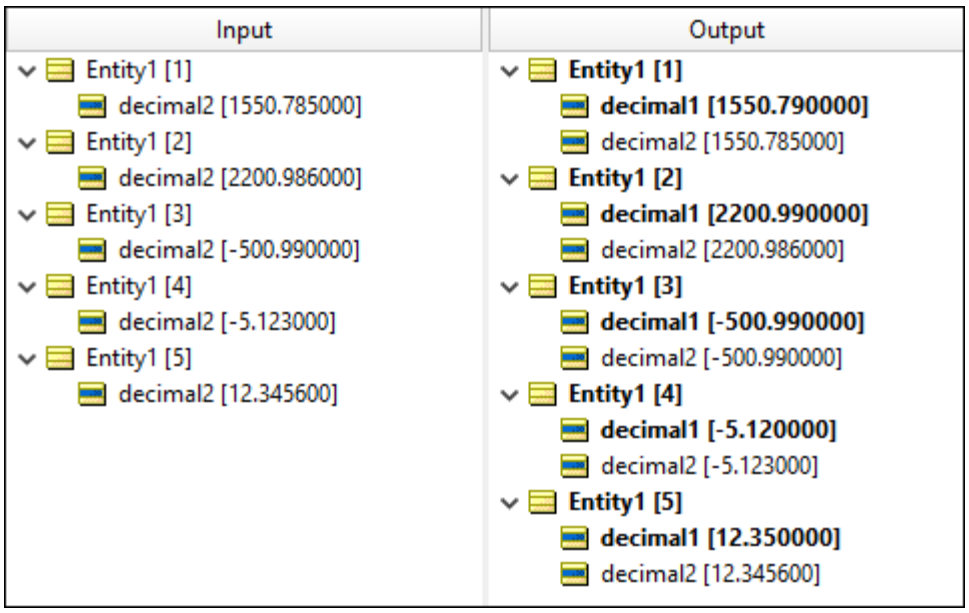
RULESHEET EXAMPLE

The following Rulesheet uses **.round** to round the value of `decimal2` to the second decimal place, and assigns it to `decimal1`.



SAMPLE TEST

A sample Ruletest provides results for five examples of `decimal2`.



SAMPLES OF ROUNDING USING DIFFERENT MODES

0 (ROUND UP) Rounds away from zero

```
10.123 -> 10.13
```

1 (ROUND DOWN) Rounds towards zero

```
10.123 -> 10.12
```

2 (ROUND CEIL) 2 Rounds towards Infinity

```
10.123 -> 10.13
```

3 (ROUND_FLOOR) Rounds towards -Infinity

```
10.123 -> 10.12
```

4 (ROUND_HALF_UP) Rounds towards nearest neighbor. If equidistant, rounds away from zero

```
10.123 -> 10.12
10.126 -> 10.13
10.125 -> 10.13
```

5 (ROUND_HALF_DOWN) Rounds towards nearest neighbor. If equidistant, rounds towards zero

```
10.123 -> 10.12
10.126 -> 10.13
10.125 -> 10.12
```

6 (ROUND_HALF_EVEN) Rounds towards nearest neighbor. If equidistant, rounds towards even neighbor

See *Round to Even (Banker's Rounding)* https://www.mathsisfun.com/numbers/rounding-methods.html and https://en.wikipedia.org/wiki/Rounding

```
8.123 -> 8.12
9.126 -> 9.13
// variation from other modes: go to nearest even number
10.125 -> 10.12
11.135 -> 11.14
```

7 (ROUND_HALF_CEIL) Rounds towards nearest neighbor. If equidistant, rounds towards Infinity

```
9.123 -> 9.12
10.126 -> 10.1
11.125 -> 11.13
```

8 (ROUND_HALF_FLOOR) Rounds towards nearest neighbor. If equidistant, rounds towards -Infinity

```
9.123 -> 9.12
10.126 -> 10.13
11.125 -> 11.12
```

Second

SYNTAX

`<DateTime>`.sec

DESCRIPTION

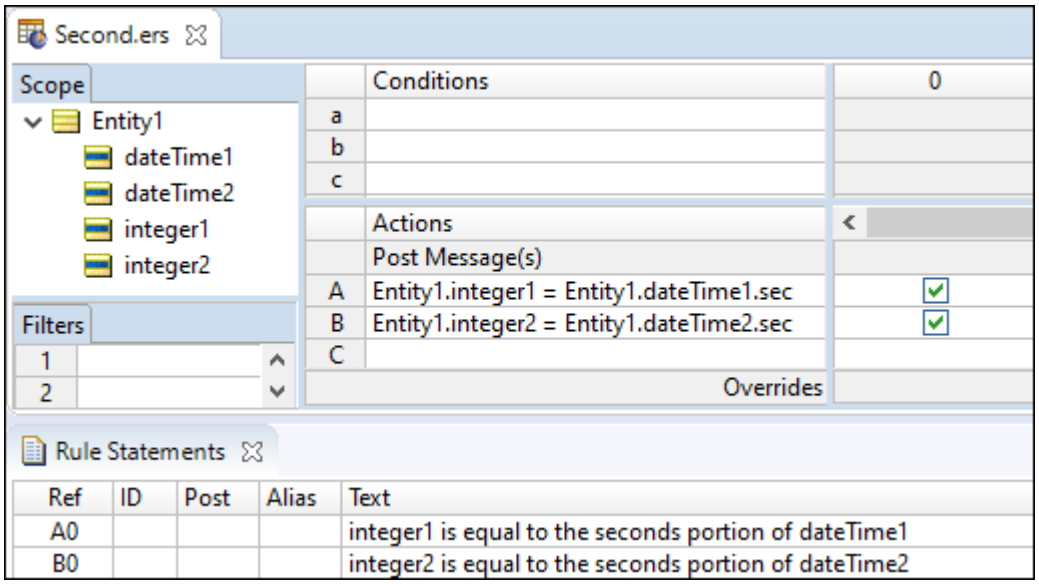
Returns the seconds portion of `<DateTime>`. The returned value is an Integer between 0 and 59.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

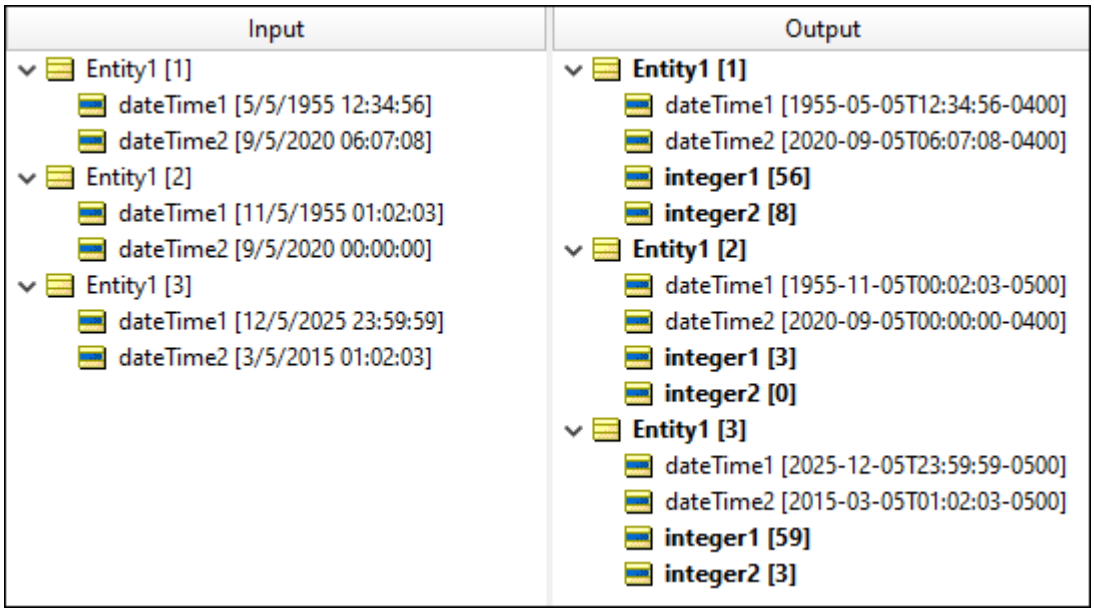
RULESHEET EXAMPLE

The following Rulesheet uses the **.sec** function to evaluate `dateTime1`, return the seconds value, and assign it to `integer1`.



SAMPLE TEST

A sample Ruletest provides results for two examples of `dateTime1`.



Seconds between

SYNTAX

`<DateTime1>.secsBetween(<DateTime2>)`

DESCRIPTION

Returns the Integer number of seconds between DateTimes. The number of milliseconds in `<DateTime1>` is subtracted from that in `<DateTime2>`, and the result divided by 1000 (the number of milliseconds in a second). The result is truncated. This function returns a positive number if `<DateTime2>` is later than `<DateTime1>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.secsBetween** to determine the number of seconds that have elapsed between `dateTime1` and `dateTime2`, compare it to the Values set, and assign a value to `string1`.

SecondsBetween.ers

	Conditions	1	2
a	Entity1.dateTime1.secsBetween(Entity1.dateTime2)	<= 60	> 60
b			
	Actions		
	Post Message(s)		
A	Entity1.string1	'Not Overdue'	'Overdue'
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If 60 or fewer seconds have elapsed between dateTime1 and dateTime2, then Entity1 is Not Overdue
2				If more than 60 seconds have elapsed between dateTime1 and dateTime2, then Entity1 is Overdue

SAMPLE TEST

A sample Ruletest provides `dateTime1` and `dateTime2` for three examples. Input and Output panels are shown below.

Input	Output
Entity1 [1]	**Entity1 [1]**
dateTime1 [5/5/1955 12:34:56]	dateTime1 [1955-05-05T12:34:56-0400]
dateTime2 [9/5/2015 06:07:08]	dateTime2 [2015-09-05T06:07:08-0400]
Entity1 [2]	**string1 [Overdue]**
dateTime1 [11/5/1955 01:02:03]	**Entity1 [2]**
dateTime2 [9/5/2015 00:00:00]	dateTime1 [1955-11-05T00:02:03-0500]
Entity1 [3]	dateTime2 [2015-09-05T00:00:00-0400]
dateTime1 [12/5/2025 23:59:59]	**string1 [Overdue]**
dateTime2 [3/5/2015 01:02:03]	**Entity1 [3]**
	dateTime1 [2025-12-05T23:59:59-0500]
	dateTime2 [2015-03-05T01:02:03-0500]
	string1 [Not Overdue]

Size of collection

SYNTAX

`<Collection>` ->size

DESCRIPTION

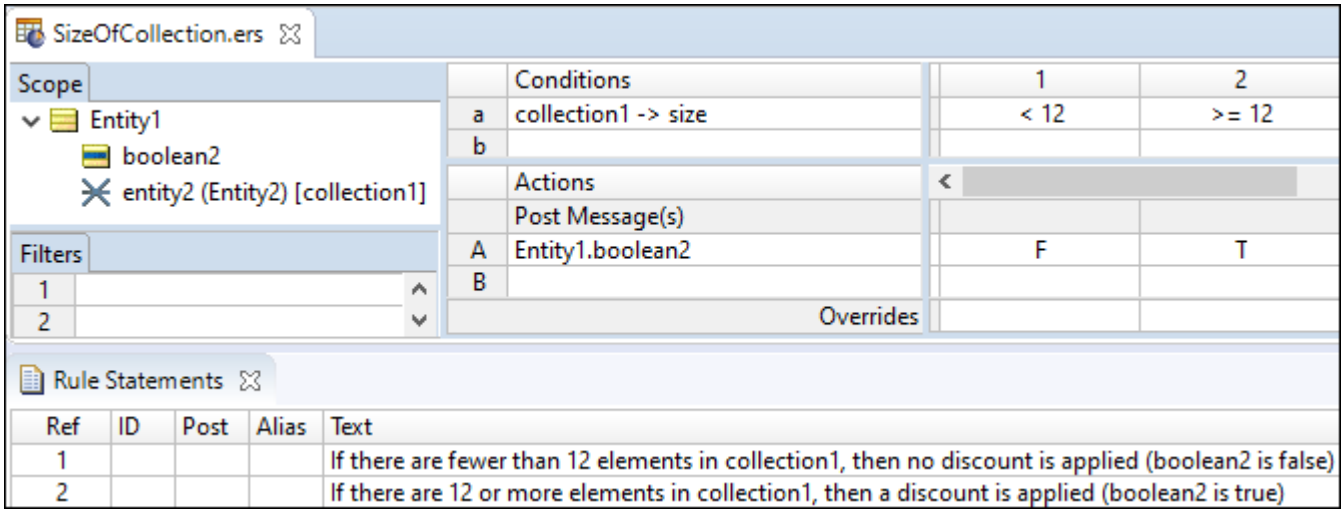
Returns the Integer number of elements in `<Collection>`. `<Collection>` must be expressed as a unique alias.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

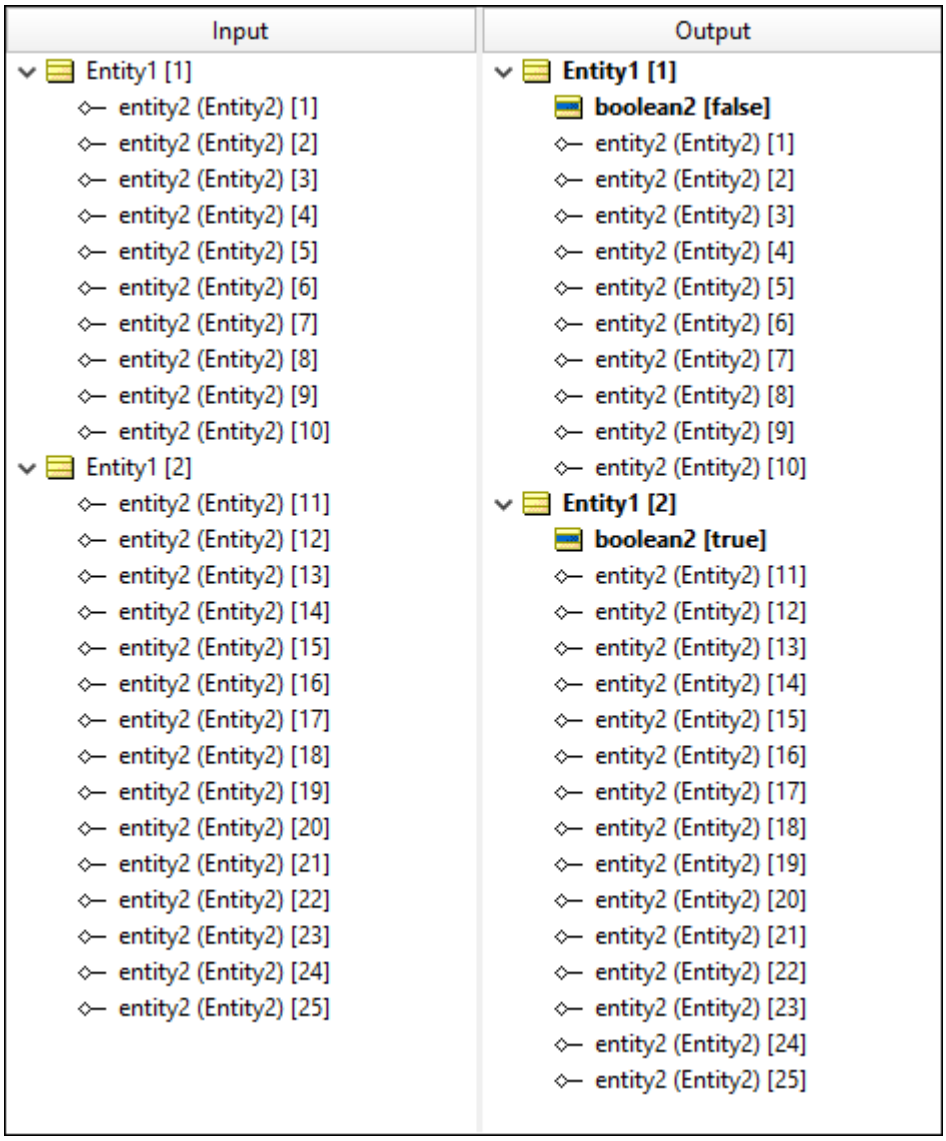
RULESHEET EXAMPLE

This sample Rulesheet uses **->size** to count the number of elements in `collection1`, and assign a value to `boolean2`. Note the use of unique alias `collection1` to represent the collection of `Entity2` associated with `Entity1`.



SAMPLE TEST

A sample Ruletest provides three examples of `collection1`. Input and Output panels are shown below.



Size of string

SYNTAX

`<String>`.size

DESCRIPTION

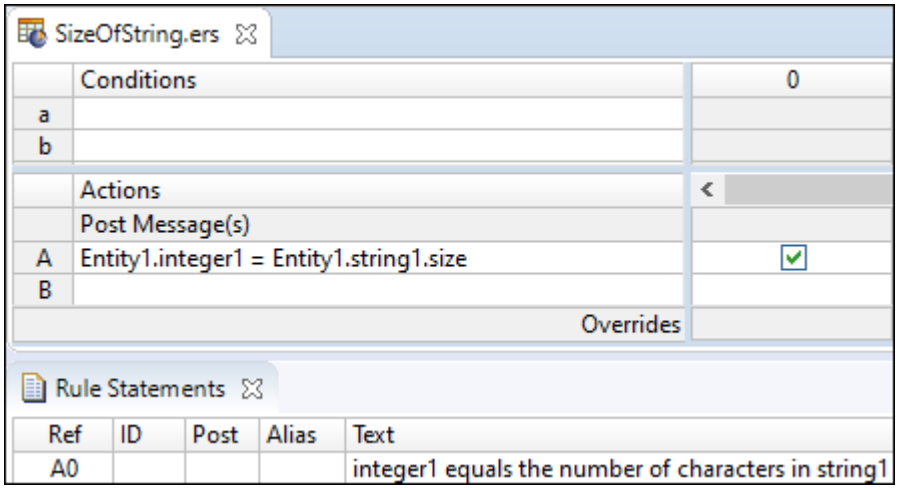
Returns the Integer number of characters in `<String>`. All characters, numbers, symbols, and punctuation marks are counted, including spaces before, within, and after words.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

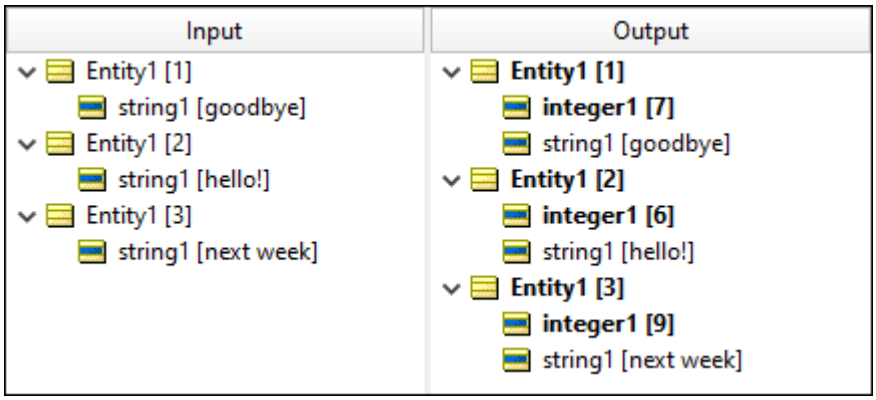
RULESHEET EXAMPLE

The following Rulesheet uses the **.size** function to determine the length of `string1` and assign it to `integer1`



SAMPLE TEST

A sample Ruletest provides three examples. Input and Output panels are shown below:



Sorted by

SYNTAX

`<Collection>` ->sortedBy(`<Attribute2>`) -> *sequence operator*.`<Attribute1>`

DESCRIPTION

Sequences the elements of `<Collection>` in ascending order, using the value of `<Attribute2>` as the index, and returns the `<Attribute1>` value of the element in the sequence position determined by the sequence operator. A sequence must be created before any sequence operator (->first, ->last, or ->at) is used to identify a particular element. `<Attribute1>` and `<Attribute2>` must be attributes of `<Collection>`.

`<Attribute2>` may be any data type except Boolean. Strings are sorted according to character precedence. `<Collection>` must be expressed as a unique alias.

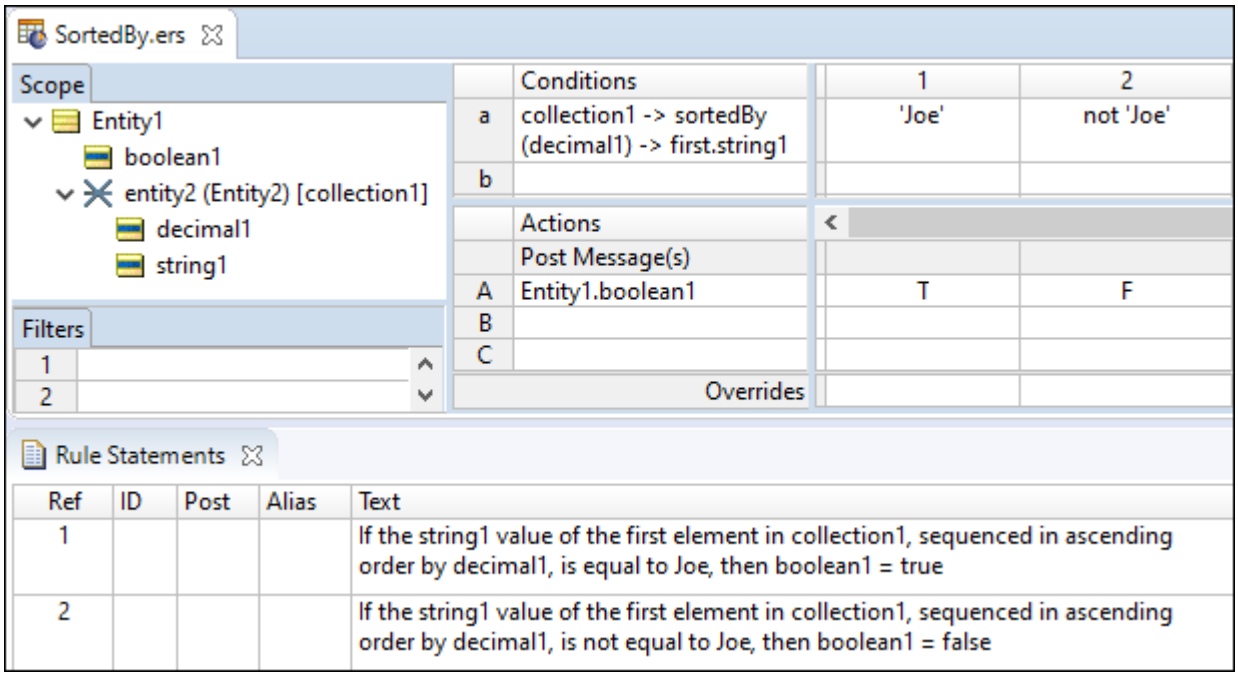
See *"Advanced collection sorting syntax"* in the Rule Modeling Guide for more examples of usage.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

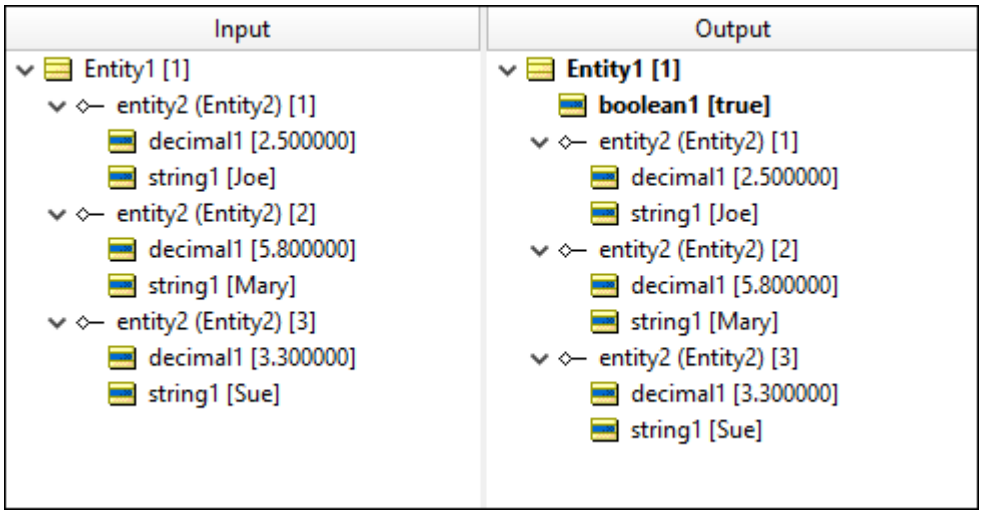
RULESHEET EXAMPLE 1 - USED IN A CONDITION

This sample Rulesheet uses **->sortedBy** in a conditional expression to create an ascending sequence from `collection` with `decimal1` as the index. `->first.string1` is used to return the value of the `string1` attribute of the first element of the sequence. If the value of `string1` is `Joe`, then `boolean1` attribute of `Entity1` is assigned the value of `true`.



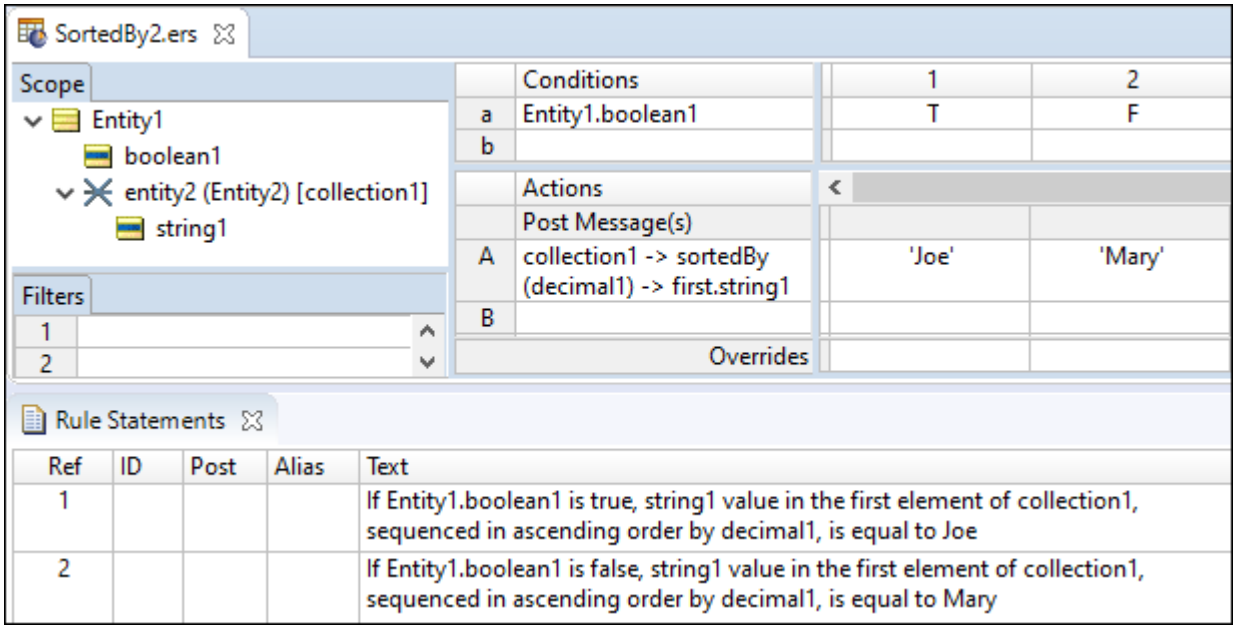
SAMPLE RULETEST 1

A sample Ruletest provides a collection of three elements, each with a `decimal1` and `string1` value. Input and Output panels are shown below.



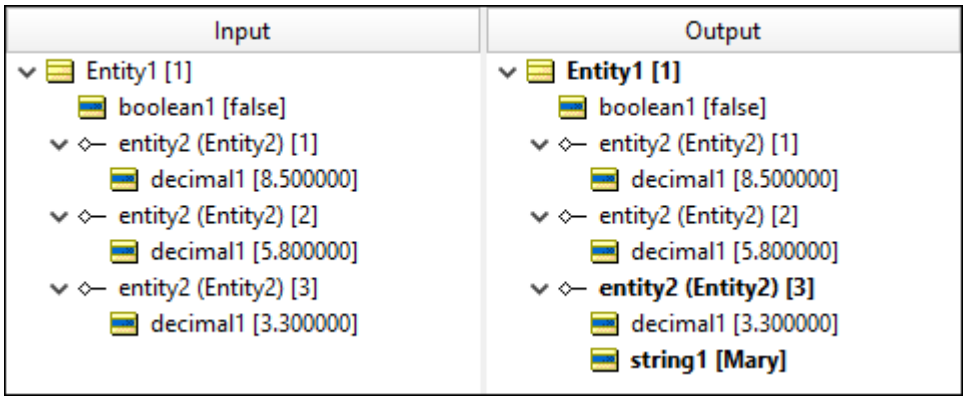
RULESHEET EXAMPLE 2 – USED IN AN ACTION

This sample Rulesheet uses **->sortedBy** in an action expression to create an ascending sequence from `collection` with `decimal1` as the index. `->first.string1` is used to return the value of the `string1` attribute of the first element of the sequence. The value of `string1` is assigned the value of `Joe` if `boolean1` attribute of `Entity1` is `true`, if `false` it is assigned the value of `Mary`.



SAMPLE RULETEST 2

A sample Ruletest provides a collection of three elements, each with a `decimal1` and `string1` value. Input and Output panels are shown below.



Sorted by descending

SYNTAX

<Collection> ->sortedByDesc(<Attribute2>) -> *sequence operator*.<Attribute1>

DESCRIPTION

Sequences the elements of <Collection> in descending order, using the value of <Attribute2> as the index, and returns the <Attribute1> value of the element in the sequence position determined by the sequence operator. A sequence must be created before any sequence operator (->first, ->last, or ->at) is used to identify a particular element. <Attribute1> and <Attribute2> must be attributes of <Collection>.

<Attribute2> may be any data type except Boolean. Strings are sorted according to their ISO character precedence. <Collection> must be expressed as a unique alias.

See *"Advanced collection sorting syntax"* in the Rule Modeling Guide for more examples of usage.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE 1 - USED IN A CONDITION

This sample Rulesheet uses **-> sortedByDesc** in a conditional expression to create an descending sequence from `collection` with `decimal1` as the index. `->first.string1` is used to return the value of the `string1` attribute of the first element of the sequence. If the value of `string1` is `Joe`, then `boolean1` attribute of `Entity1` is assigned the value of `true`.

SortedByDescending.ers

Scope

- Entity1
  - boolean1
  - entity2 (Entity2) [collection1]
    - decimal1
    - string1

Filters

1	
2	

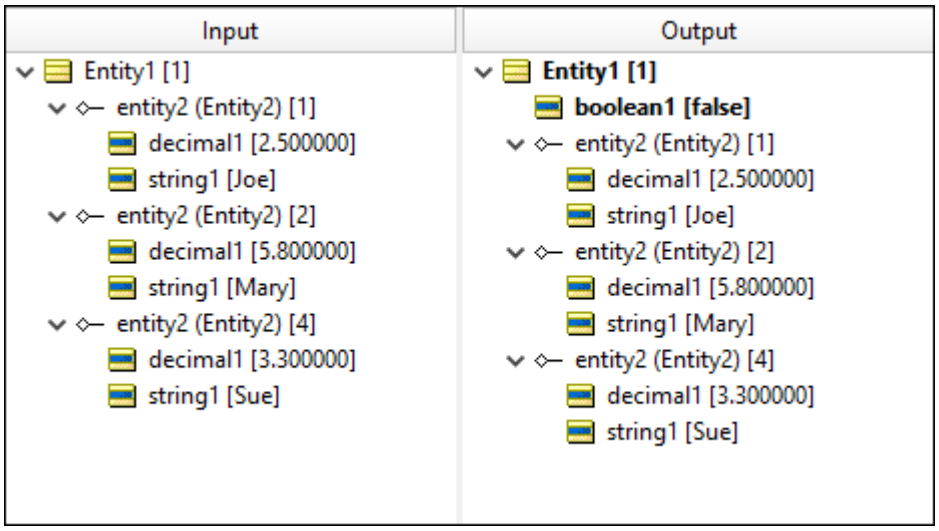
	Conditions	1	2
a	collection1 -> sortedByDesc (decimal1) -> first.string1	'Joe'	not 'Joe'
b			
	Actions		
	Post Message(s)		
A	Entity1.boolean1	T	F
B			
C			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If Entity1.boolean1 is true, string1 value in the first element of collection1, sequenced in descending order by decimal1, is equal to Joe
2				If Entity1.boolean1 is false, string1 value in the first element of collection1, sequenced in descending order by decimal1, is equal to Mary

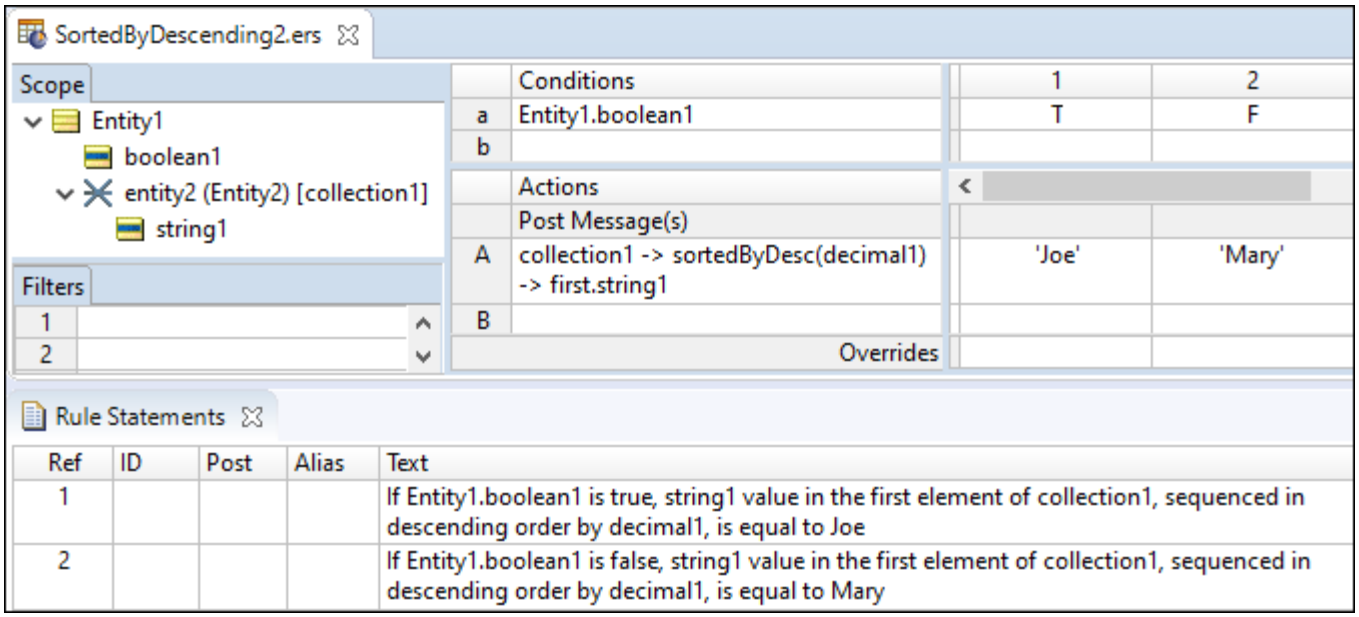
SAMPLE RULETEST 1

A sample Ruletest provides a collection of three elements, each with a `decimal1` value. Input and Output panels are shown below.



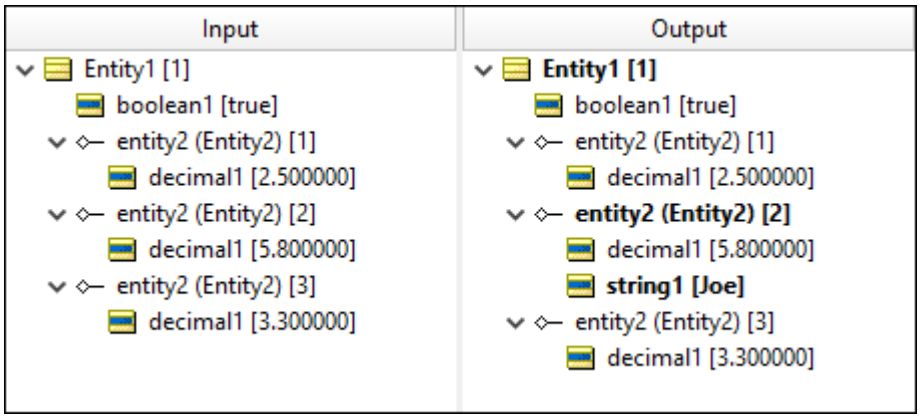
RULESHEET EXAMPLE 2 – USED IN AN ACTION

This sample Rulesheet uses **sortedByDesc** in an action expression to create an descending sequence from `collection` with `decimal1` as the index. `->first.string1` is used to return the value of the `string1` attribute of the first element of the sequence. The value of `string1` is assigned the value of `Joe` if `boolean1` attribute of `Entity1` is `true`, if `false` it is assigned the value of `Mary`.



SAMPLE RULETEST 2

A sample Ruletest provides a collection of three elements, each with a `decimal1` value. Input and Output panels are shown below.



Starts with

SYNTAX

<String1>.startsWith(<String2>)

DESCRIPTION

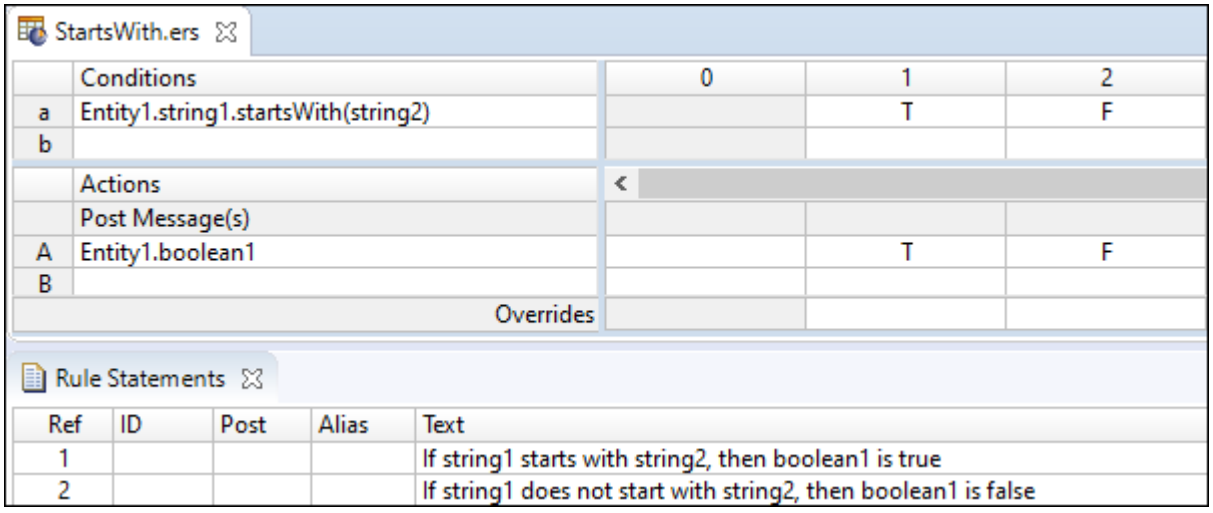
Returns a value of true if <String1> begins with the characters specified in <String2>. Comparisons are case-sensitive.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

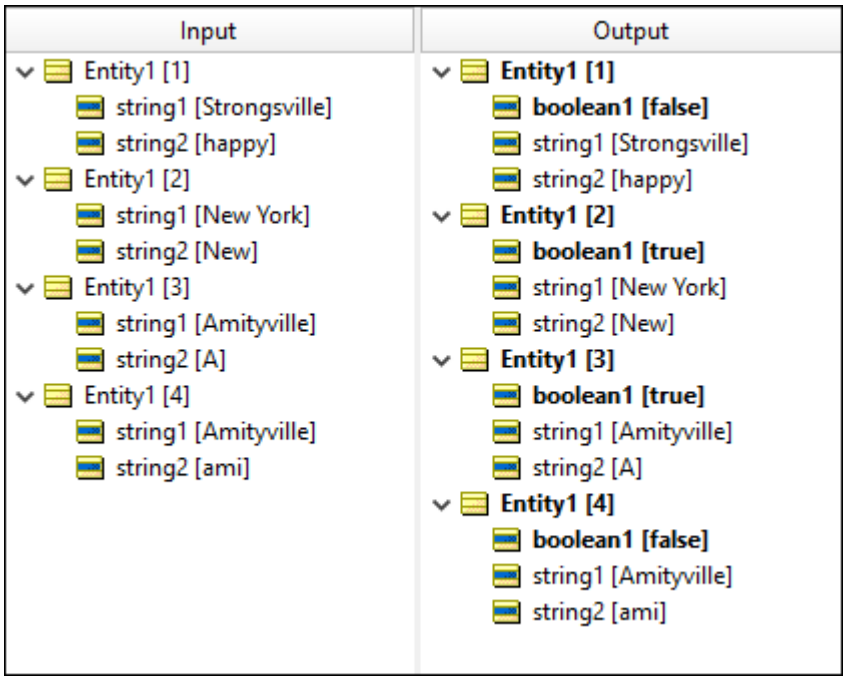
RULESHEET EXAMPLE

The following Rulesheet uses **.startsWith** to evaluate whether `string1` begins with the value of `string2` and assigns a different value to `boolean1` for each outcome.



SAMPLE TEST

A sample Ruletest provides `string1` and `string2` values for four examples. Input and Output panels are shown below.



Substring

SYNTAX

`<String>`.substring(`<Integer1>`, `<Integer2>`)

DESCRIPTION

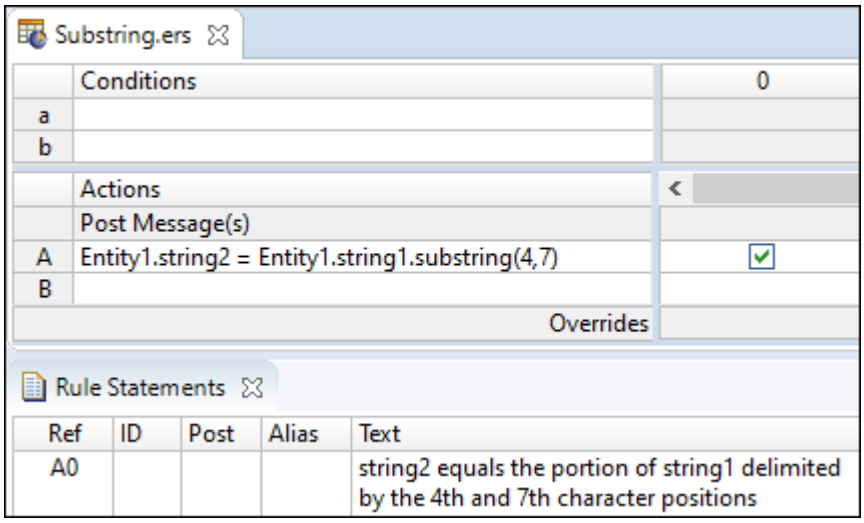
Returns the portion of `<String>` beginning with the character in position `<Integer1>` and ending with the character in position `<Integer2>`. The number of characters in `<String>` must be at least equal to `<Integer2>`, otherwise an error will be produced. Both `<Integer1>` and `<Integer2>` must be positive integers, and `<Integer2>` must be greater than `<Integer1>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

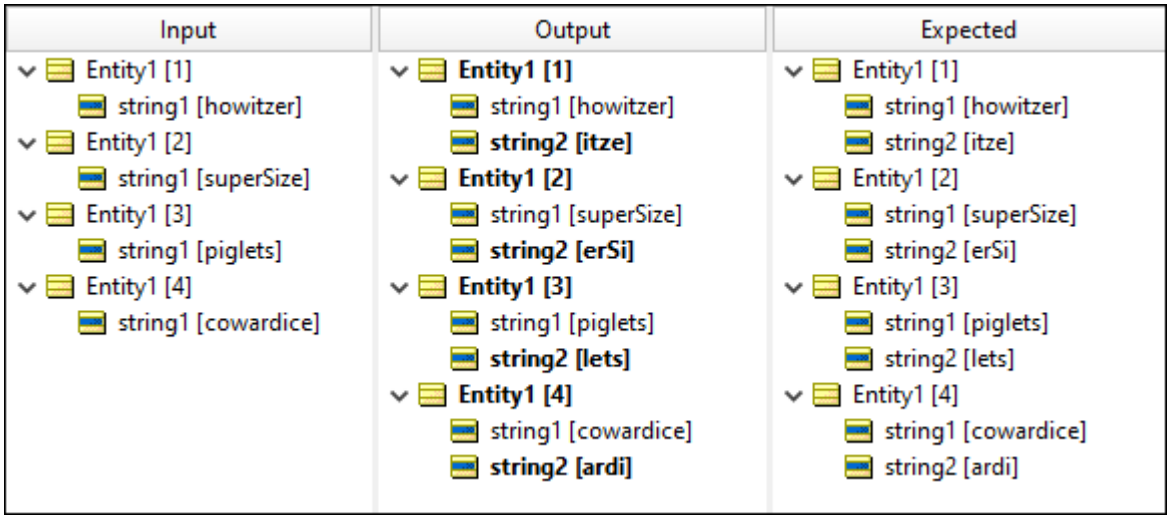
RULESHEET EXAMPLE

This sample Rulesheet uses **.substring** to return those characters of `string1` between positions `4` and `7` (inclusive), and assign the resulting value to `string2`.



SAMPLE RULETEST

A sample Ruletest provides `string1` values for four examples. Input and Output panels are shown below.



Subtract

SYNTAX

`<Number1> - <Number2>`

DESCRIPTION

Subtracts the value of `<Number2>` from that of `<Number1>`. The resulting data type is the more expansive of those of `<Number1>` and `<Number2>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

This sample Rulesheet uses **subtract** to reduce the value of `decimal1` by `decimal2`, compare the resulting value to zero, and assign a value to `boolean1`

Subtract.ers

	Conditions	0	1	2
a	Entity1.decimal1 - Entity1.decimal2 > 0		T	F
b				
	Actions			
	Post Message(s)			
A	Entity1.boolean1		T	F
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1				decimal1 is greater than decimal2
2				decimal2 is greater than decimal1

SAMPLE TEST

A Ruletest provides three examples of `decimal1` and `decimal2`. Input and Output panels are shown below.

Input	Output
Entity1 [1]	**Entity1 [1]**
decimal1 [23.000000]	**boolean1 [true]**
decimal2 [7.200000]	decimal1 [23.000000]
Entity1 [2]	decimal2 [7.200000]
decimal1 [10.300000]	**Entity1 [2]**
decimal2 [41.670000]	**boolean1 [false]**
Entity1 [3]	decimal1 [10.300000]
decimal1 [-4.560000]	decimal2 [41.670000]
decimal2 [-8.120000]	**Entity1 [3]**
	boolean1 [true]
	decimal1 [-4.560000]
	decimal2 [-8.120000]

Sum

SYNTAX

`<Collection.attribute>` ->sum

DESCRIPTION

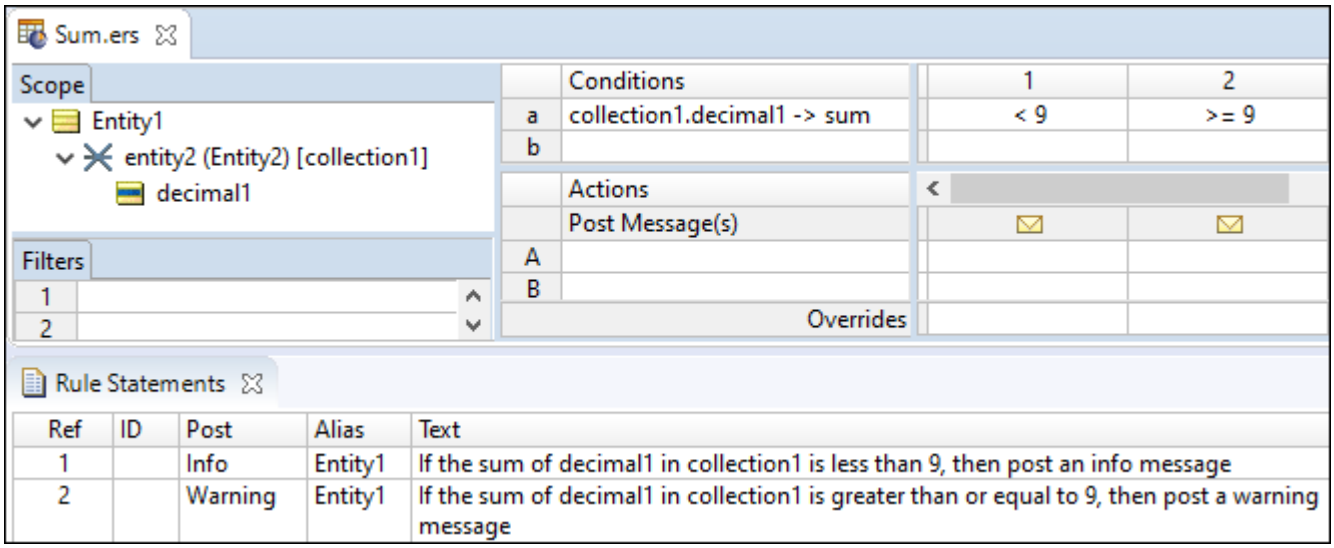
Sums the values of the specified `<attribute>` for all elements in `<Collection>`. `<attribute>` must be a numeric data type. `<Collection>` must be expressed as a unique alias.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

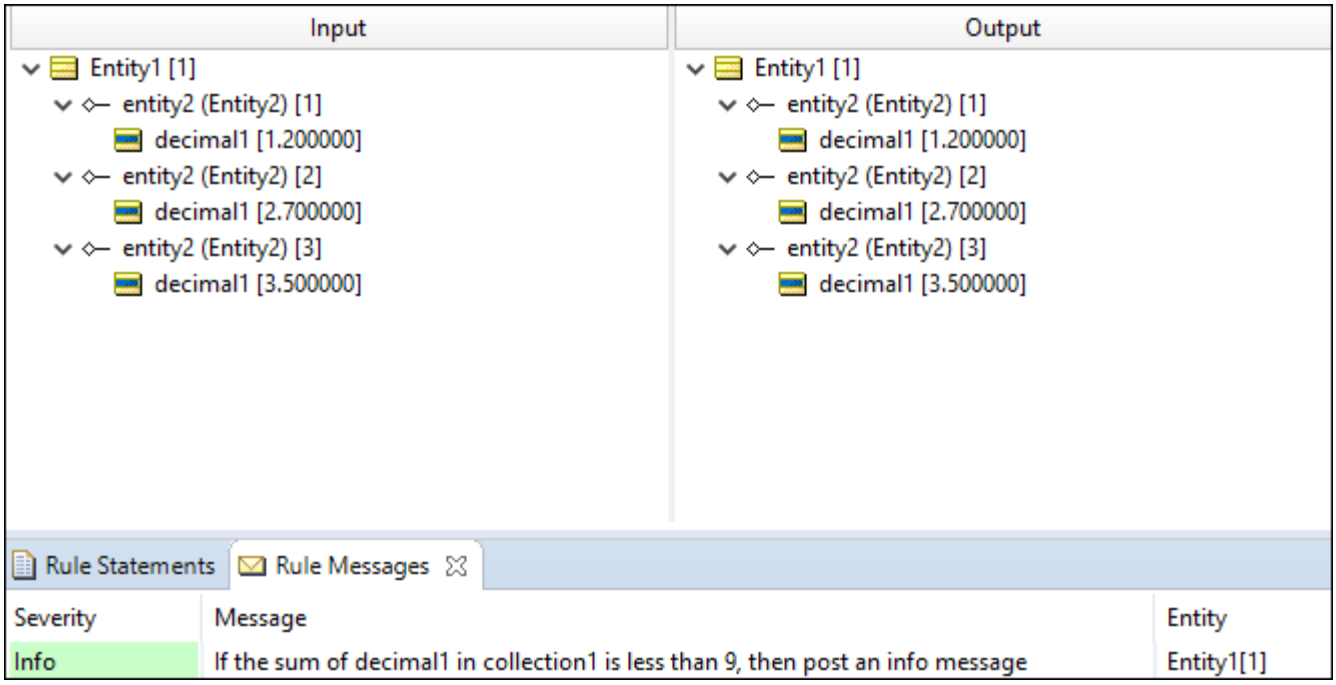
RULESHEET EXAMPLE

This Rulesheet uses the **->sum** function to add all `decimal1` attributes within `collection1`. Note the use of unique alias `collection1` to represent the collection of `Entity2` associated with `Entity1`



SAMPLE TEST

A sample Ruletest provides 3 elements in `collection1`. Input and Output panels are shown below.



Today

SYNTAX

today

DESCRIPTION

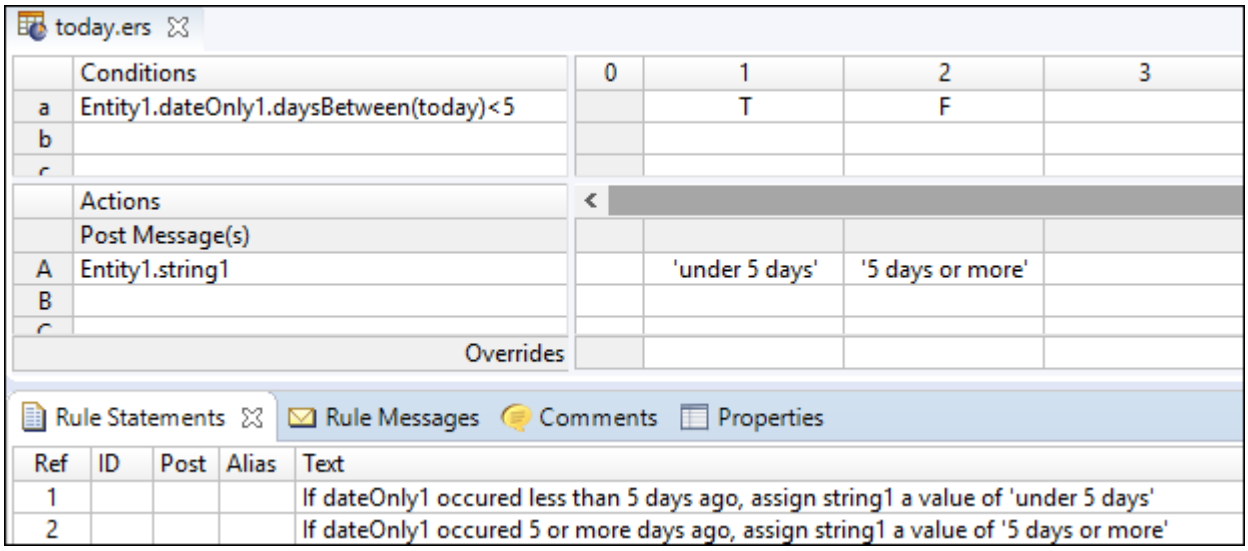
Returns the current system date when the rule is executed. This Date Only value is assigned the first time **today** is used in a Decision Service, then remains constant until the Decision Service finishes execution, regardless of how many additional times it is used. This means that every rule in a Rule Set using **today** will use the same Date Only value. No time portion is assigned

USAGE RESTRICTIONS

The Literals row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

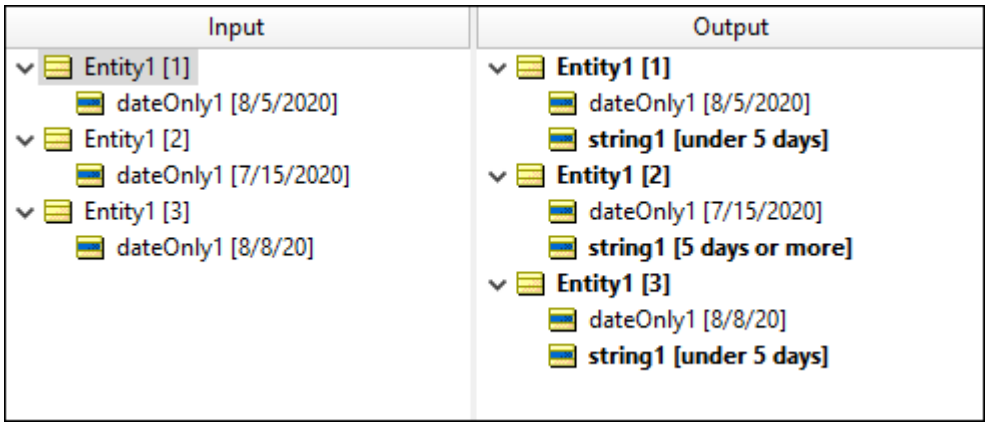
RULESHEET EXAMPLE

The following Rulesheet uses **today** to determine how many days have elapsed between today and `dateTime1`, and assign a value to `string1` based on the result.



SAMPLE TEST

A sample Ruletest provides three examples of `dateOnly1`. Assume **today** is equal to `August 9, 2020`. Input and Output panels are shown below:



To dateTime

SYNTAX

<String>.toDateTime

DESCRIPTION

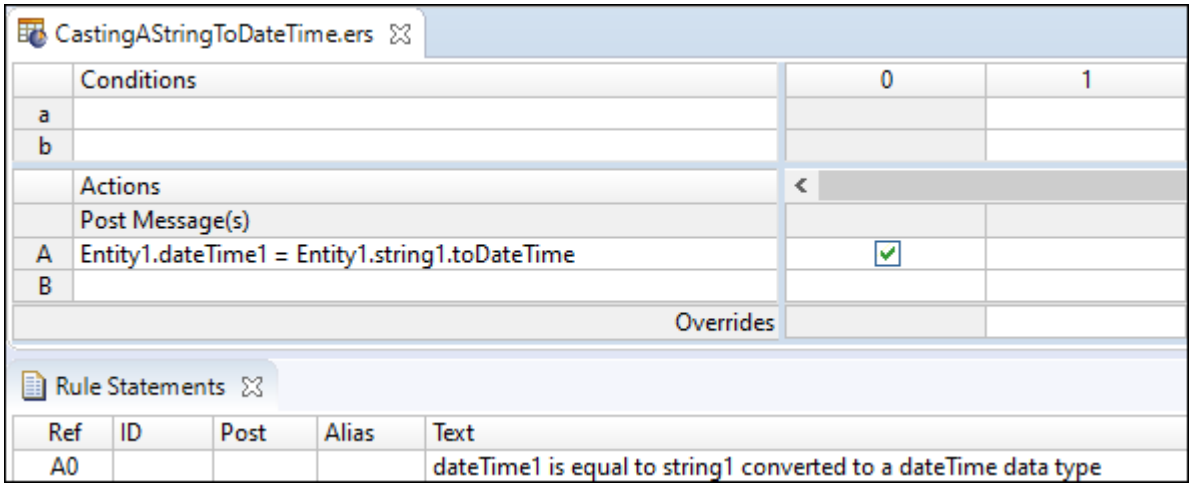
Converts the value in <String> to data type DateTime ONLY if date is an ISO8601 string or millisecs since epoch as a string. An invalid casting results in a null.

USAGE RESTRICTIONS

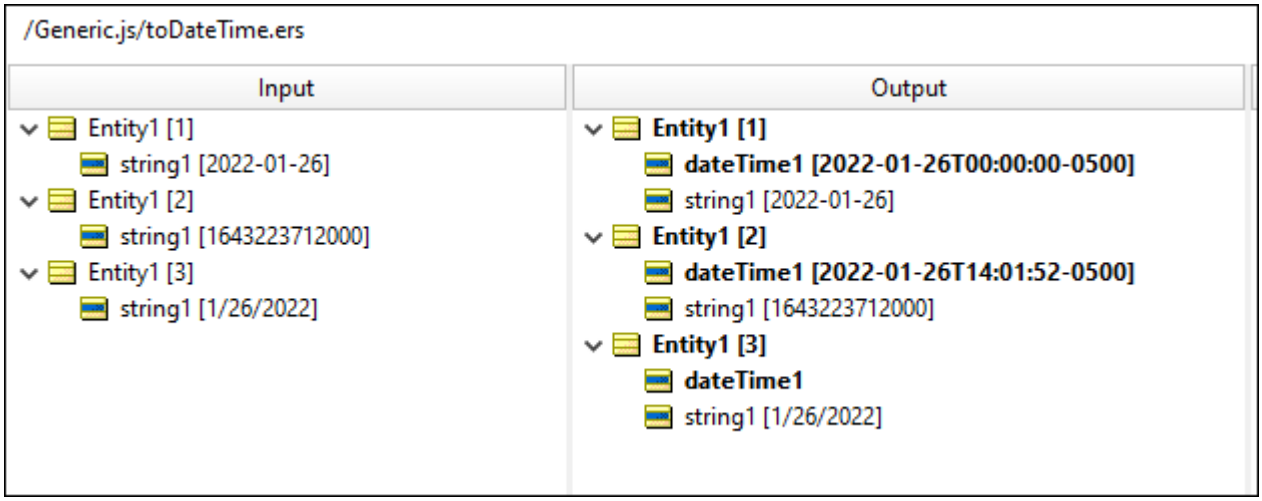
The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.toDateTime** to convert `string1` to type DateTime and assign the value to `dateTime1`.



SAMPLE TEST



To date Casting a dateTime to a date

SYNTAX

`<DateTime>`.toDate

DESCRIPTION

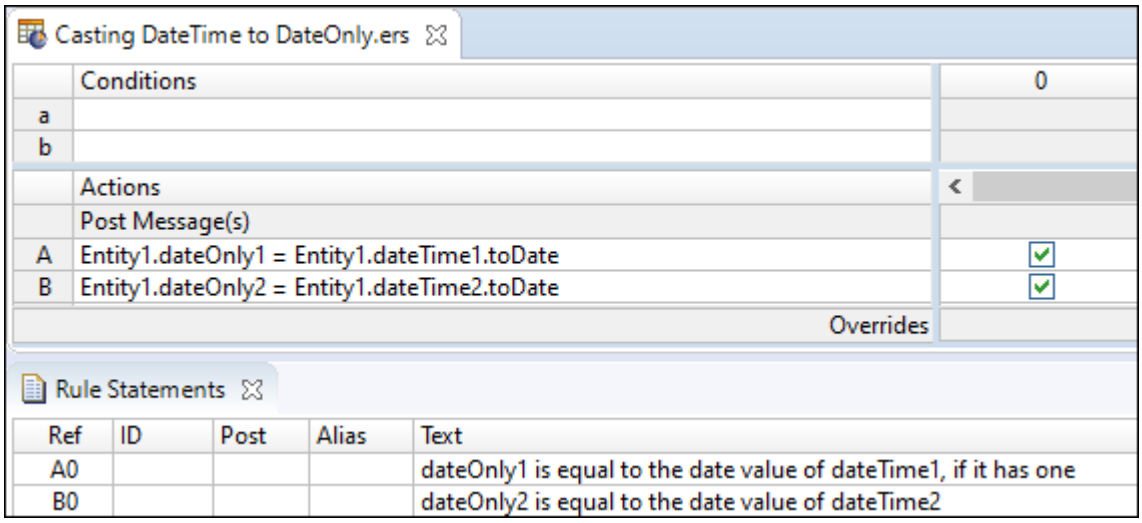
Converts the value in `<DateTime>` to a Date datatype, containing only the date portion of the DateTime. If `<DateTime>` contains no date information, then the system epoch is used.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.toDate** to convert `dateTime1` and `DateTime2` to Date datatypes and assign the values to `dateTime1` and `dateTime2`.



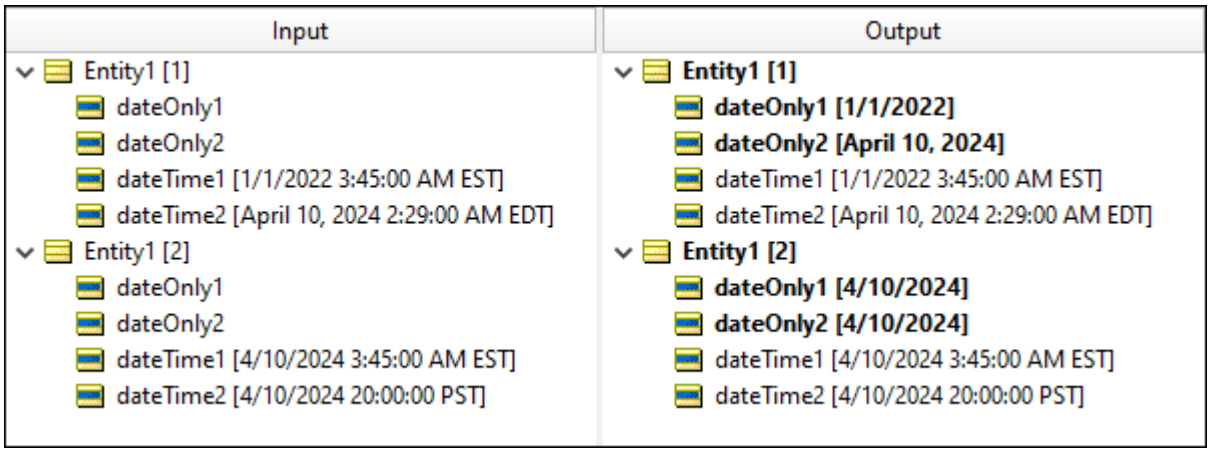
Casting DateTime to DateOnly.ers

	Conditions	0
a		
b		
	Actions	
	Post Message(s)	
A	Entity1.dateOnly1 = Entity1.dateTime1.toDate	☑
B	Entity1.dateOnly2 = Entity1.dateTime2.toDate	☑
	Overrides	

Rule Statements

Ref	ID	Post	Alias	Text
A0				dateOnly1 is equal to the date value of dateTime1, if it has one
B0				dateOnly2 is equal to the date value of dateTime2

SAMPLE TEST



Input	Output
Entity1 [1]	**Entity1 [1]**
dateOnly1	**dateOnly1 [1/1/2022]**
dateOnly2	**dateOnly2 [April 10, 2024]**
dateTime1 [1/1/2022 3:45:00 AM EST]	dateTime1 [1/1/2022 3:45:00 AM EST]
dateTime2 [April 10, 2024 2:29:00 AM EDT]	dateTime2 [April 10, 2024 2:29:00 AM EDT]
Entity1 [2]	**Entity1 [2]**
dateOnly1	**dateOnly1 [4/10/2024]**
dateOnly2	**dateOnly2 [4/10/2024]**
dateTime1 [4/10/2024 3:45:00 AM EST]	dateTime1 [4/10/2024 3:45:00 AM EST]
dateTime2 [4/10/2024 20:00:00 PST]	dateTime2 [4/10/2024 20:00:00 PST]

To dateTime Casting a string to a dateTime

SYNTAX

<String>.toDateTime

DESCRIPTION

Converts the value in <String> to data type DateTime ONLY if all characters in <String> correspond to a valid Date, Time, or DateTime mask (format). For complete details on DateTime masks, see *Rule Modeling Guide*.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.toDateTime** to convert `string1` to type DateTime and assign the value to `dateTime1`.

CastingAStringToDateTime.ers

	Conditions	0	1
a			
b			
	Actions		
	Post Message(s)		
A	Entity1.dateTime1 = Entity1.string1.toDateTime	☑	
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
A0				dateTime1 is equal to string1 converted to a dateTime data type

SAMPLE TEST

Input	Output
Entity1 [1] string1 [12/31/2021] Entity1 [2] string1 [January 29, 2022] Entity1 [3] string1 [Thursday, June 2, 2022 1:00:00 PM PDT]	**Entity1 [1]** **dateTime1 [12/31/2021 12:00:00 AM]** string1 [12/31/2021] **Entity1 [2]** **dateTime1 [January 29, 2022 12:00:00 AM]** string1 [January 29, 2022] **Entity1 [3]** **dateTime1 [Thursday, June 2, 2022 1:00:00 PM PDT]** string1 [Thursday, June 2, 2022 1:00:00 PM PDT]

To dateTime Casting a date to a dateTime

SYNTAX

<Date>.toDateTime

DESCRIPTION

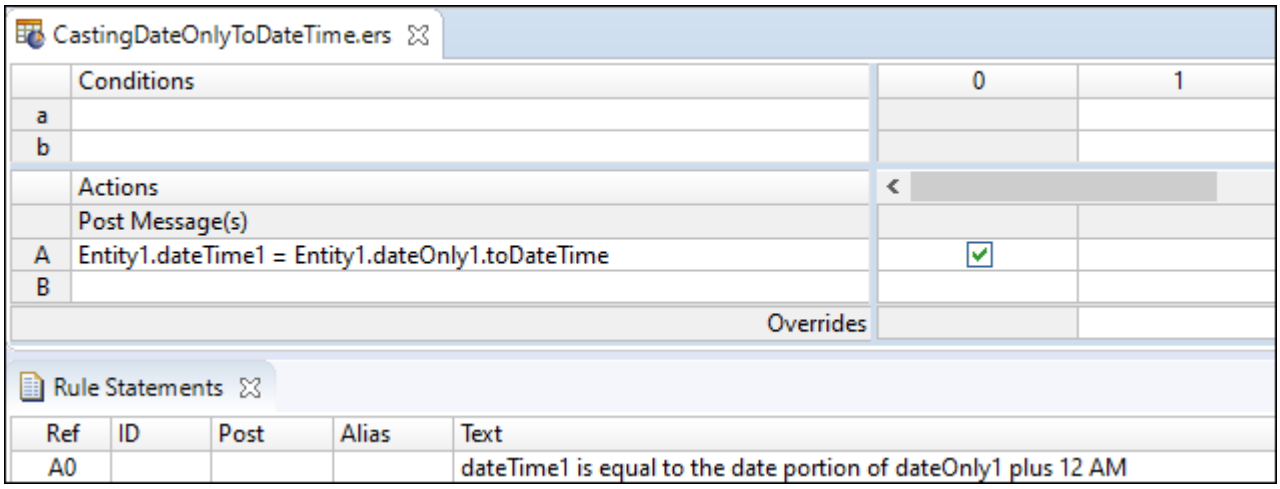
Converts the value in <Date> to data type DateTime. The date portion is the same as the <Date> value and the time portion is set to `00:00:00 AM` in the current timezone.

USAGE RESTRICTIONS

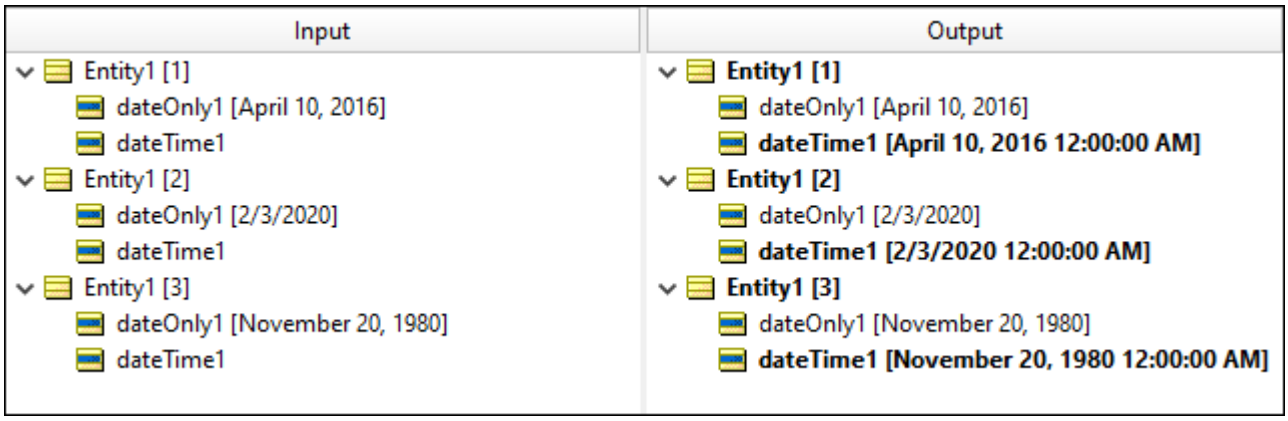
The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.toDateTime** to convert `dateOnly1` to type DateTime and assign the value to `dateTime1`.



SAMPLE TEST



To dateTime Timezone offset

SYNTAX

`<Date>.toDateTime(<String>)`

DESCRIPTION

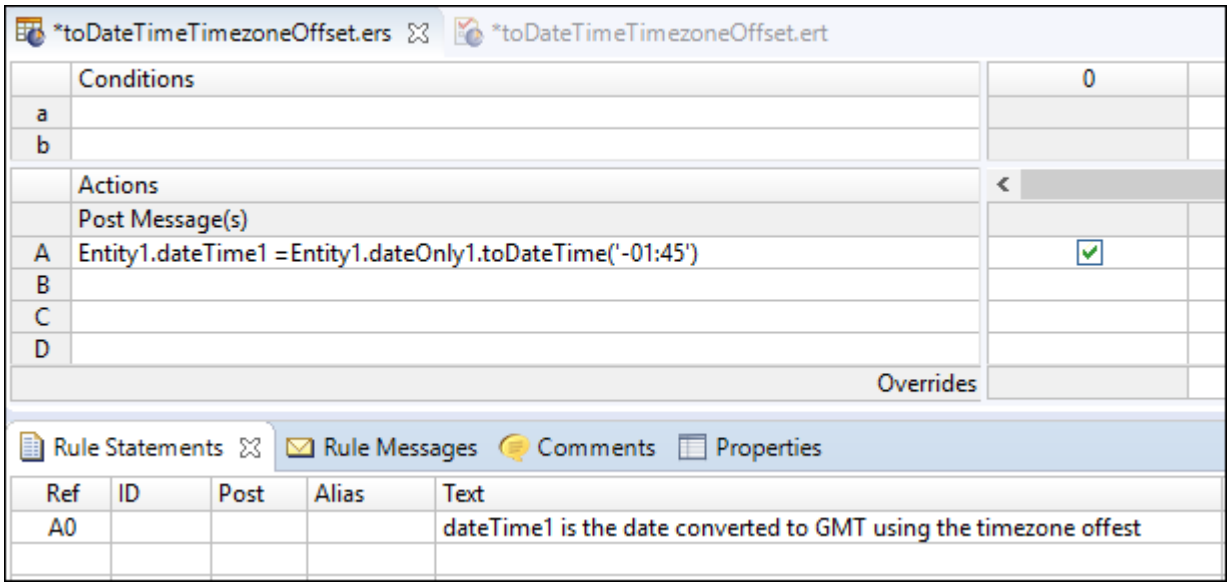
Converts the value in `<Date>` to data type DateTime ONLY if all characters in `<Date>` correspond to a valid DateTime mask (format). The date portion is the same as the `<Date>` value and the time portion is set to `00:00:00` in the timezone specified by `<String>`, which is the timeZoneOffset. The timeZoneOffset must take the form of a valid, signed timezone offset such as `'-08:00'`, `'+03:30'`, `'+01:45'`.

USAGE RESTRICTIONS

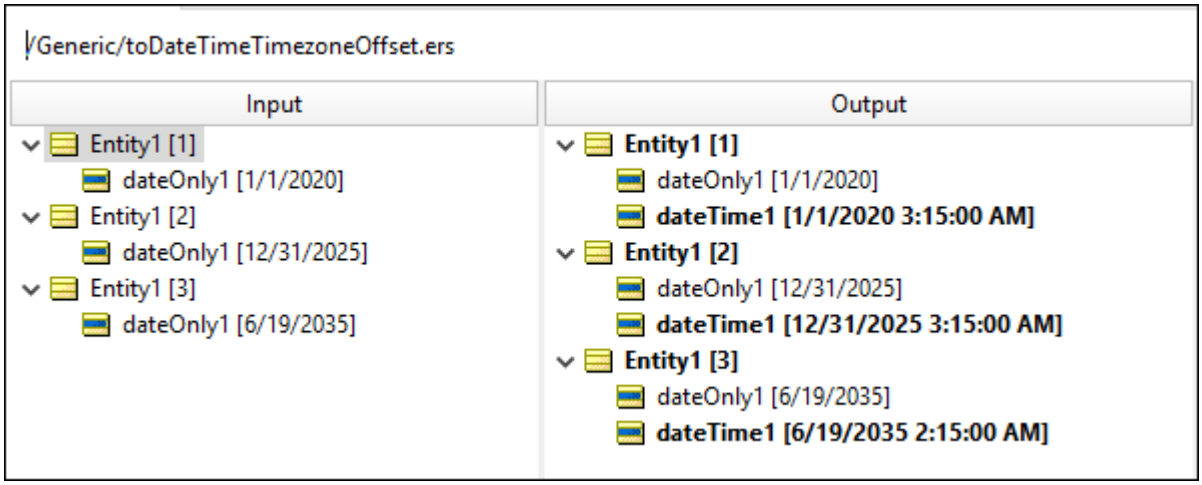
The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.toDateTime** to convert `dateOnly1` to type DateTime and assign the value to `dateTime1`, with a timezone offset of `-01:45`.



SAMPLE TEST



To decimal

SYNTAX

`<Integer>`.toDecimal

`<String>`.toDecimal

DESCRIPTION

Converts the value in `<Integer>` or all characters in `<String>` to data type Decimal. Converts a String to Decimal ONLY if all characters in `<String>` are numeric and contain not more than one decimal point. If any non-numeric characters are present in `<String>` (other than the single decimal point or a leading minus sign), no value is returned by the function.

Note: Integer values may be assigned directly to Decimal data types without using the **.toDecimal** operator because a Decimal data type is more expansive than an Integer.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.toDecimal** to convert `integer1` and `string1` to type Decimal and assign them to `decimal1` and `decimal2`, respectively.

ToDecimal.ers

	Conditions	0	1
a			
b			
	Actions		
	Post Message(s)		
A	Entity1.decimal1 = Entity1.integer1.toDecimal	☑	
B	Entity1.decimal2 = Entity1.string1.toDecimal	☑	
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
A0				decimal1 is equal to the value of integer1 converted into a decimal data type
B0				decimal2 is equal to the value of string1 converted into a decimal data type

SAMPLE TEST

Input	Output
Entity1 [1]	**Entity1 [1]**
integer1 [1]	**decimal1 [1.000000]**
Entity1 [2]	integer1 [1]
integer1 [25]	**Entity1 [2]**
Entity1 [3]	**decimal1 [25.000000]**
string1 [1]	integer1 [25]
Entity1 [4]	**Entity1 [3]**
string1 [5.345678]	**decimal2 [1.000000]**
	string1 [1]
	Entity1 [4]
	decimal2 [5.345678]
	string1 [5.345678]

To integer

SYNTAX

<Decimal>.toInteger

<String>.toInteger

DESCRIPTION

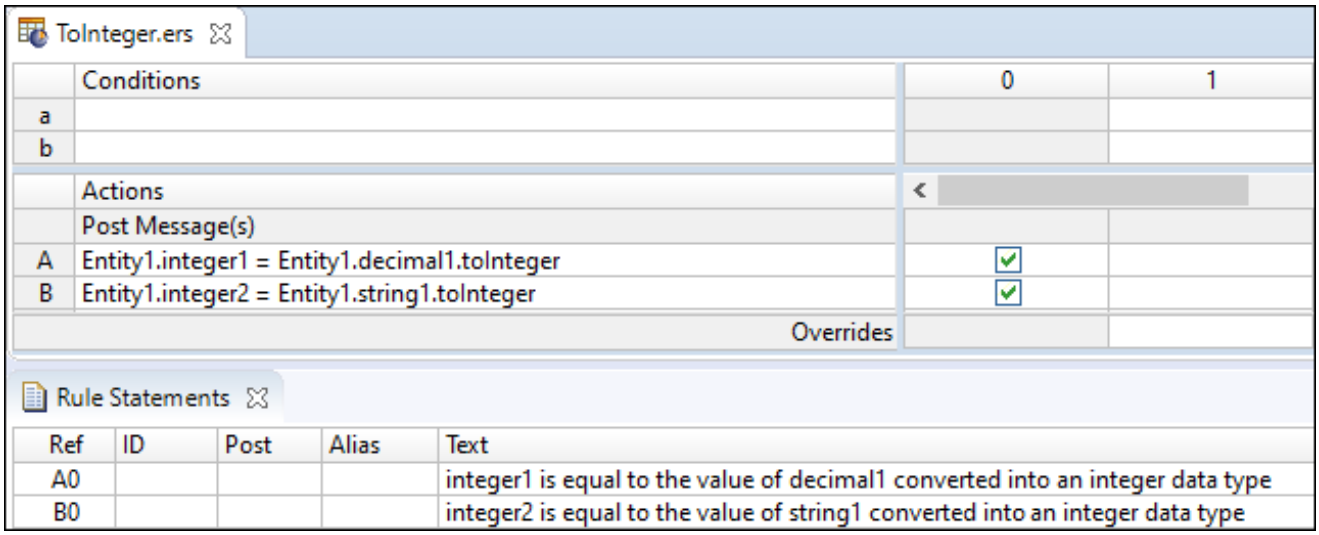
Converts the value in <Decimal> or all characters in <String> to data type Integer. All decimals have fractional portions truncated during the conversion. Strings are converted ONLY if all characters in <String> are numeric, without a decimal point. If any non-numeric characters (with the sole exception of a single leading minus sign for negative numbers) are present in <String>, no value is returned by the function. Do not use on String values of null or empty String ('') -- a pair of single quote marks -- as that will generate an error message.

USAGE RESTRICTIONS

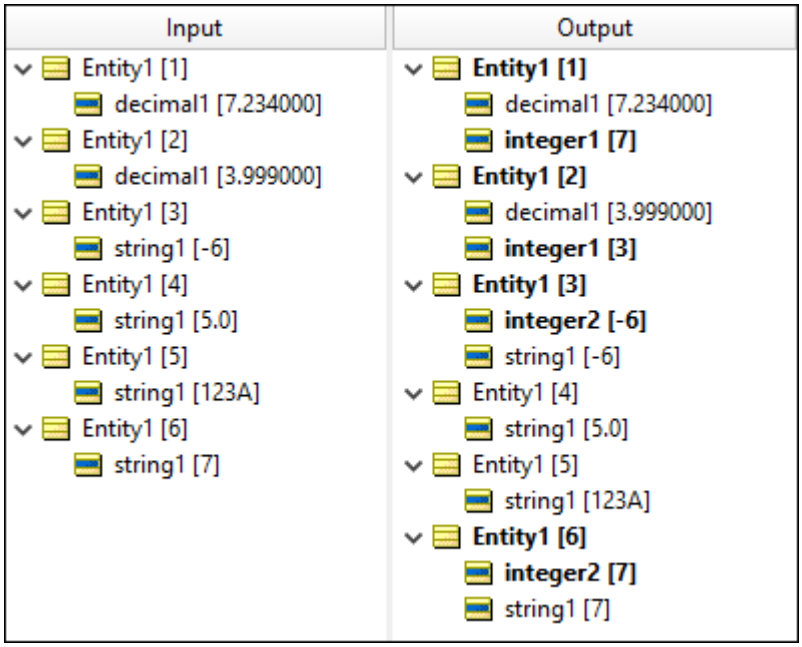
The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.toInteger** to convert decimal1 and string1 to type Integer and assign them to integer1 and integer2, respectively.



SAMPLE TEST



Cases when the toInteger operator accepts null and empty values for string attributes

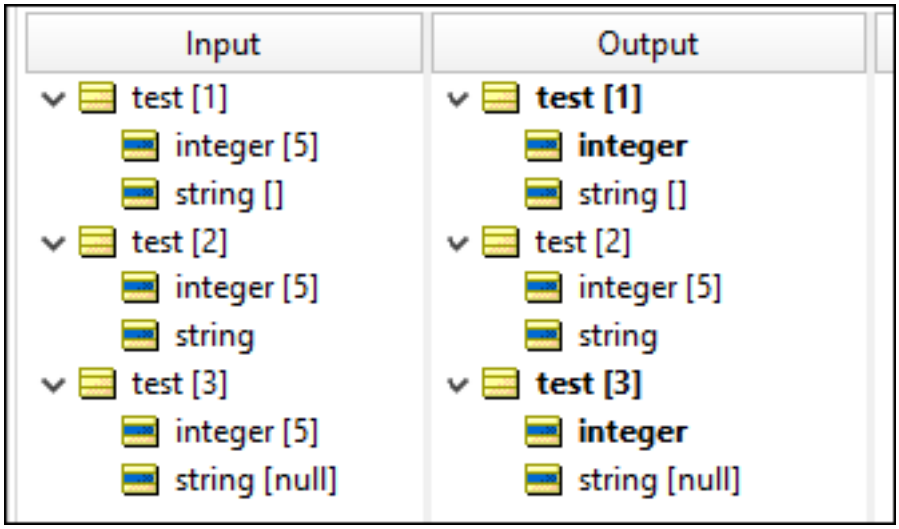
There are two factors:

1. Prior to evaluating a rule, Corticon checks if any attribute values used in the expressions in the rule are null and, if so, does not execute the rule.
2. During expression evaluation, Corticon protects against null pointer exceptions. The expression "test.string.toInteger" will return null if the string is not an integer. However, the expression "test.string.toInteger + 3" will return “3” if the string is not a number – the value 0 being used as the result of the toInteger.

Consider the action expression:

```
test.integer =test.string.toInteger
```

Here is the Ruletest output for three tests:



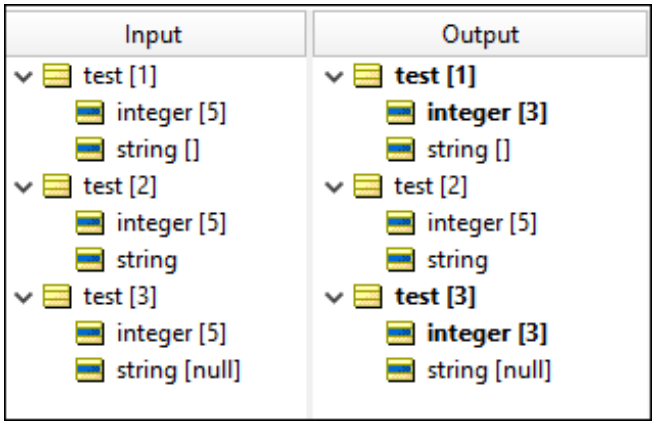
How this Ruletest was processed:

- In test 1, the string is empty but not a null value so the expression evaluates and assigns null to integer.
- In test 2, the string is null so the pre-check for null values does not pass and the expression is not evaluated and the value of integer is unchanged
- In test 3, the string is the string “null” but not a null value so the expression evaluates and assigns null to integer. (Note the value “null” here is a string, it could have just as well been “foo”).

Now change the action expression to:

```
test.integer =test.string.toInteger + 3
```

Here is the Ruletest output now:



How this Ruletest was processed now:

- In test 1, the string is empty but not a null value so the expression evaluates. To prevent a NPE during evaluation, the value 0 is used as the result of the toInteger resulting in the expression being “0 + 3” so integer is assigned a value of 3.
- In test 2, the string is null so the pre-check for null values does not pass and the expression is not evaluated and the value of integer is unchanged.

- In test 3, the string is the string “null” but not a null value so the expression evaluates in the same fashion as 1, that is, “0 + 3” and assigns a value of 3.

You might argue that you cannot assume a value of 0 when doing toString on a non-number string. However, to protect a business user against runtime exceptions, Corticon makes logical substitutions during rule evaluation to protect against null values.

To string

SYNTAX

`<Decimal>`.toString

`<Integer>`.toString

DESCRIPTION

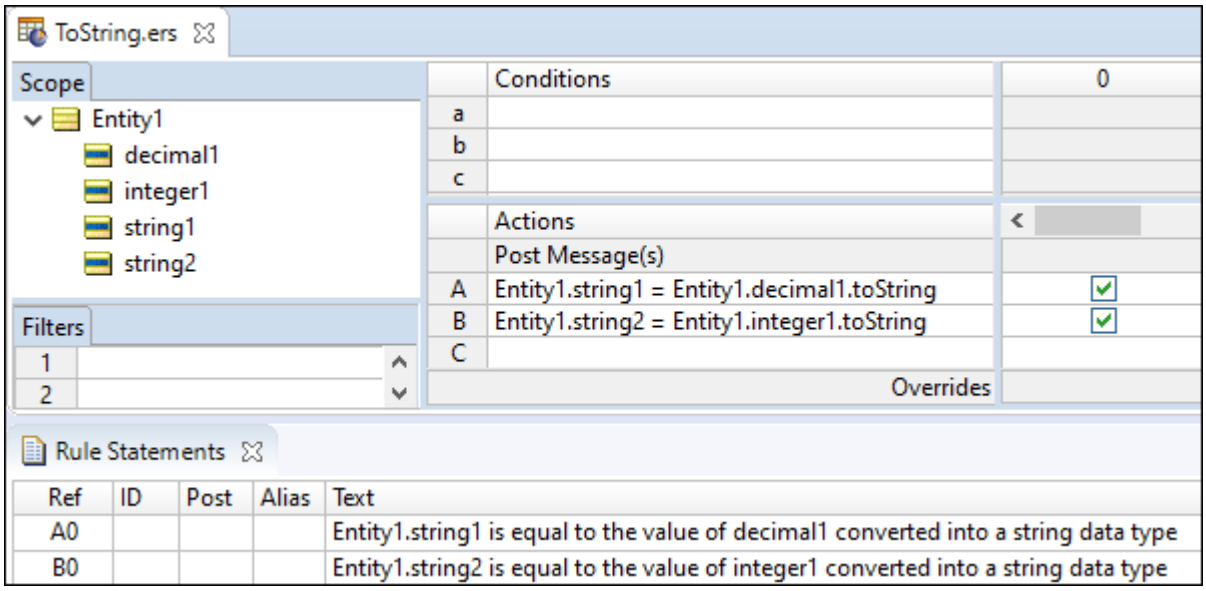
Converts a value to a data type of String.

USAGE RESTRICTIONS

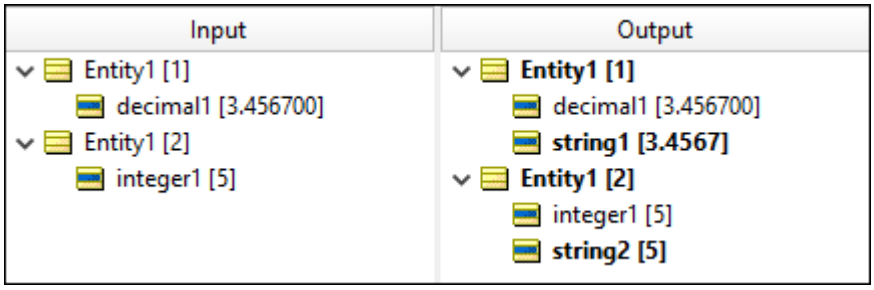
The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.toString** to convert numeric data types to strings.



SAMPLE TEST



Trim spaces

SYNTAX

<String>.trimSpaces

DESCRIPTION

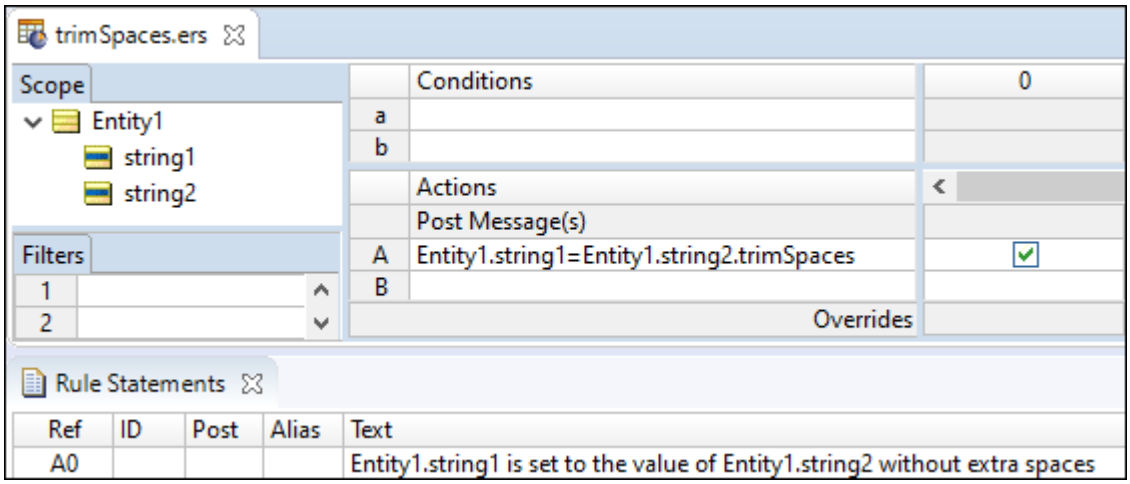
Returns <String>.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

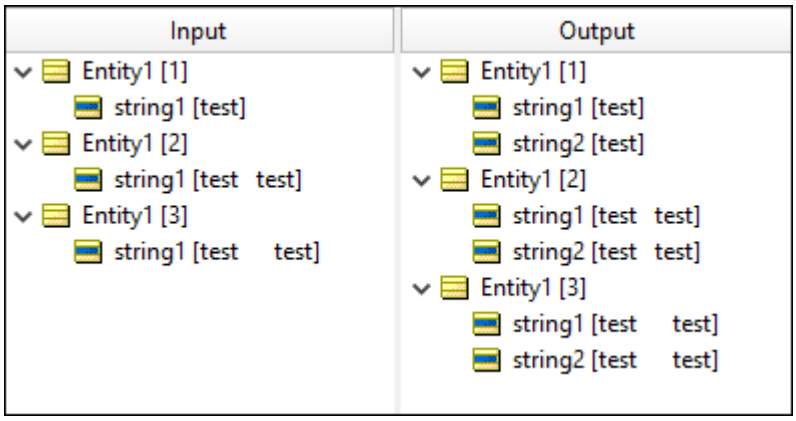
This sample Rulesheet uses **trimSpaces**.



SAMPLE RULETEST

A sample Ruletest provides a collection of three elements, each with a String value. Input and Output panels are shown below.

Note: As the Studio Tester trims spaces in the input area, you cannot really test this operation here!



True

SYNTAX

true or T

DESCRIPTION

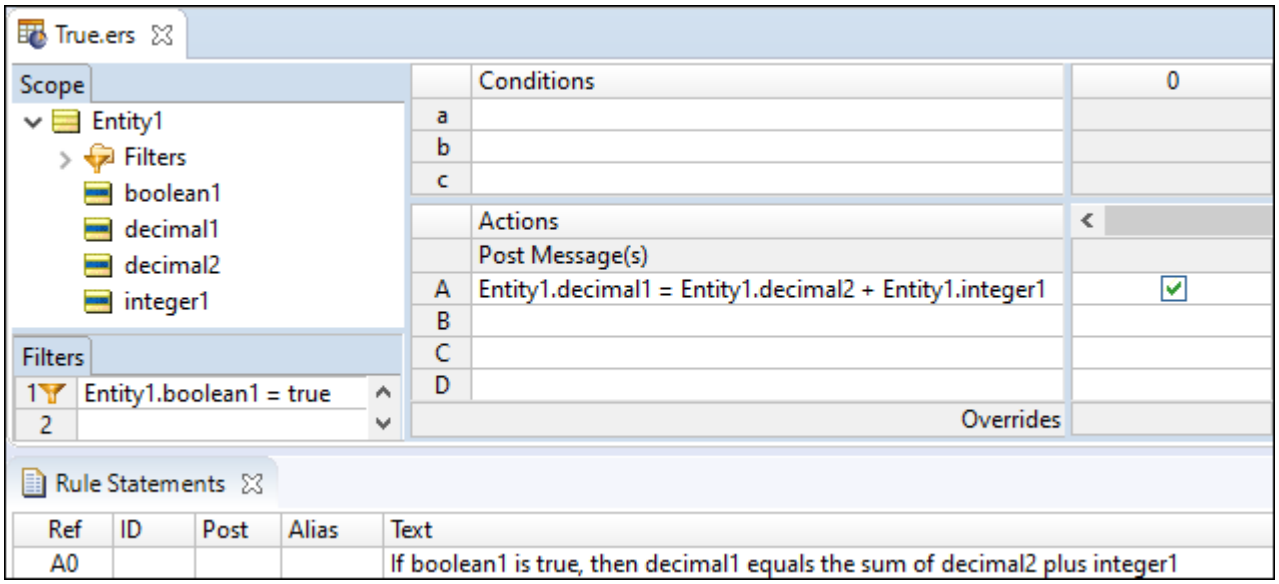
Represents Boolean value true. Recall from the discussion oftruth values that an `<expression>` is evaluated for its truth value, so the expression `Entity1.boolean1=true` will evaluate to `true` only if `boolean1=true`. But since `boolean1` is Boolean and has a truth value all by itself without any additional syntax, we do not actually need the "`=true`" piece of the expression. Many examples in the documentation use explicit syntax like `boolean1=true` or `boolean2=false` for clarity and consistency, even though `boolean1` or `not boolean2` are equivalent logical expressions.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

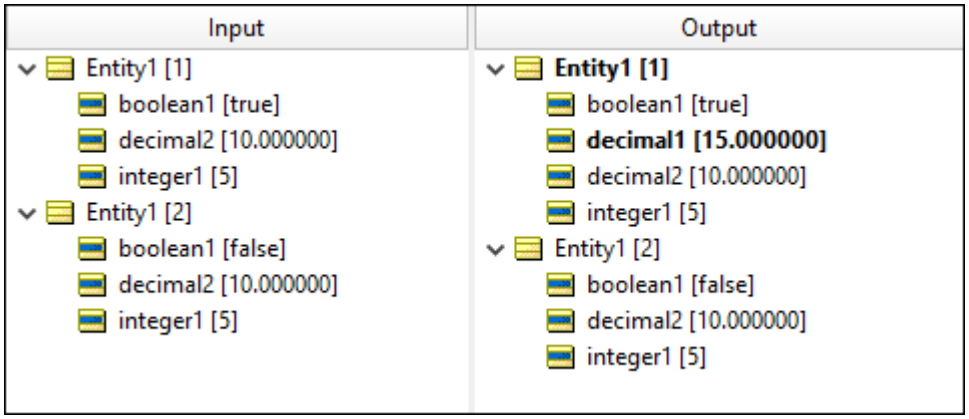
RULESHEET EXAMPLE

The following Rulesheet uses **true** in a Precondition to Ruletest whether `boolean1` is true, and perform the Nonconditional computation if it is. As discussed above, the alternative expression `Entity1.boolean1` is logically equivalent.



SAMPLE TEST

A sample Ruletest provides three examples. Assume `decimal2=10.0` and `integer1=5` for all examples. Input and Output panels are shown below:



Truncate

SYNTAX

```
<Decimal>.truncate
```

DESCRIPTION

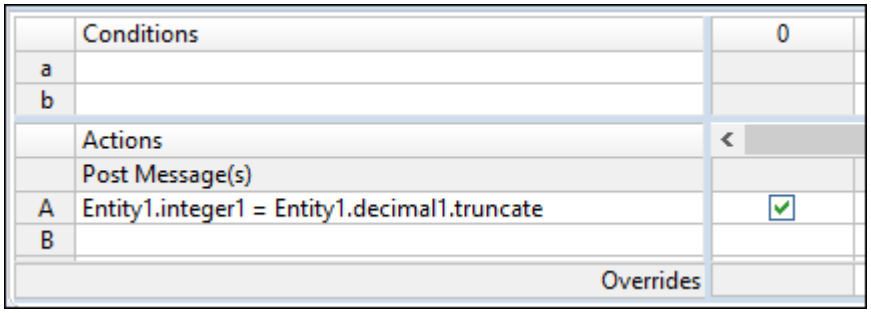
Truncates "this" Decimal value to an integer by removing the fractional portion.

USAGE RESTRICTIONS

The Operators row in the table of Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

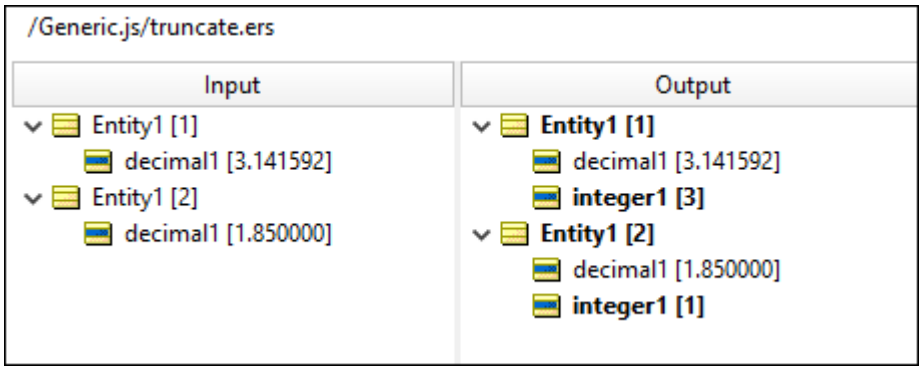
RULESHEET EXAMPLE

This sample Rulesheet uses **.truncate** to produce the integer value of `decimal1` and assign it to `integer1`



SAMPLE RULETEST

A sample Ruletest provides `decimal1` values for two scenarios. Input and Output panels are shown below.



Uppercase

SYNTAX

<String>.toUpper

DESCRIPTION

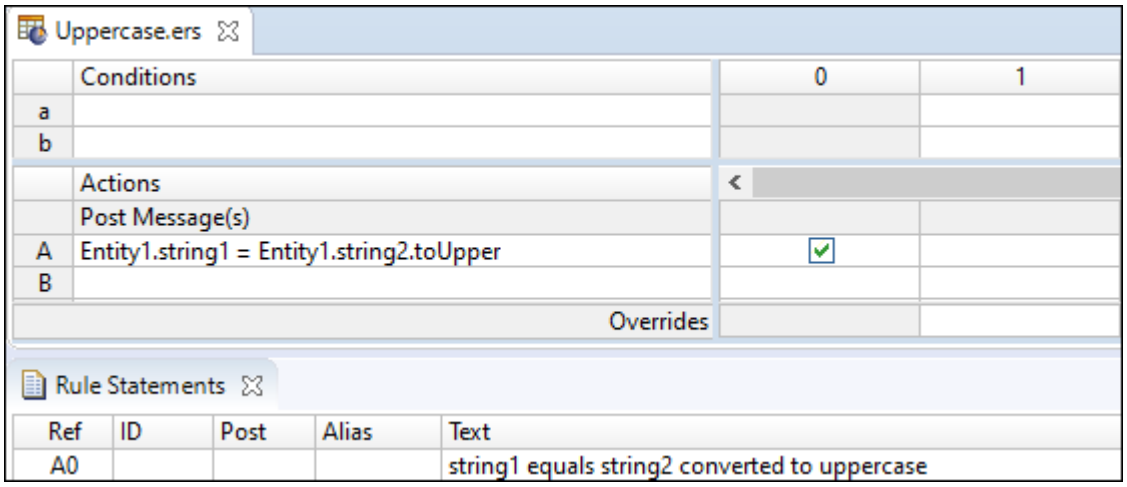
Converts all characters in <String> to uppercase.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

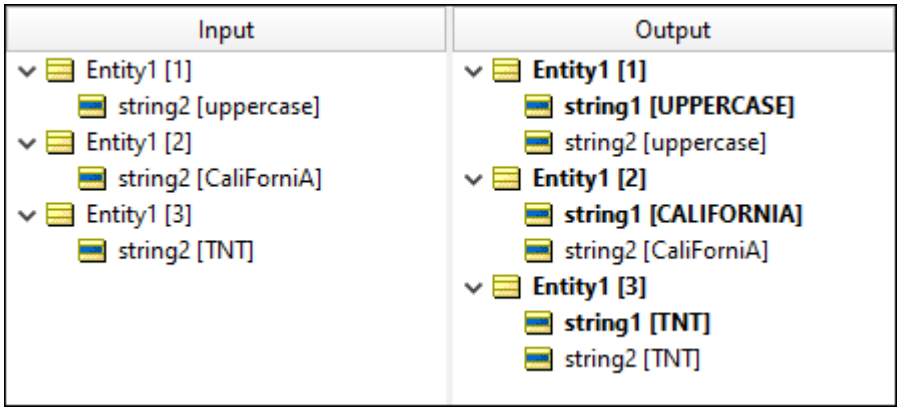
RULESHEET EXAMPLE

The following Rulesheet uses **.toUpper** to convert `string2` to uppercase and assign it to `string1`.



SAMPLE TEST

A sample Ruletest provides three examples. Input and Output panels are shown below:



Week of month

SYNTAX

<DateTime>.weekOfMonth

DESCRIPTION

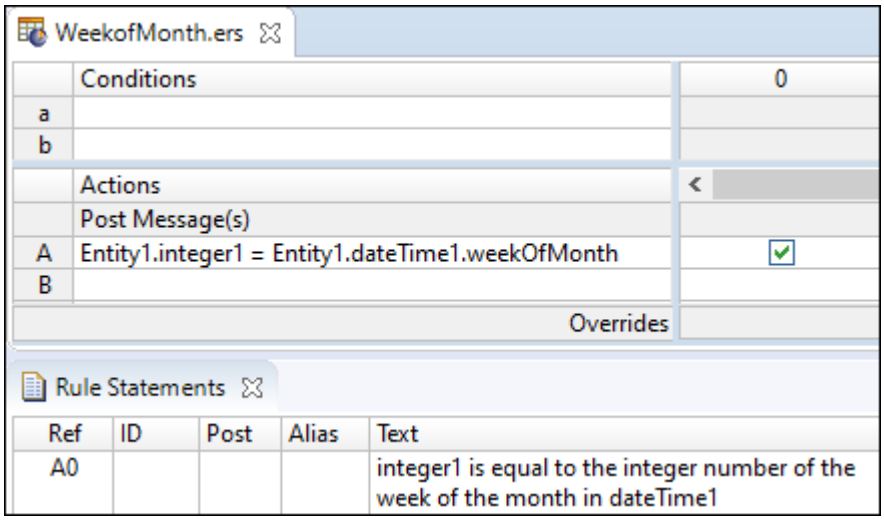
Returns an Integer from 1 to 6, equal to the week number within the month in <DateTime>. A week begins on Sunday and ends on Saturday.

USAGE RESTRICTIONS

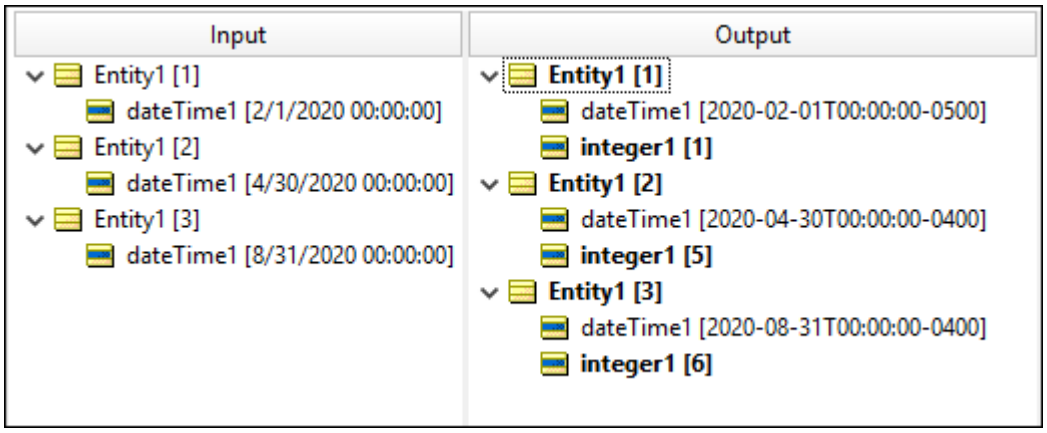
The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.weekOfMonth** to assign a value to integer1.



SAMPLE TEST



Week of year

SYNTAX

`<DateTime>`.weekOfYear

DESCRIPTION

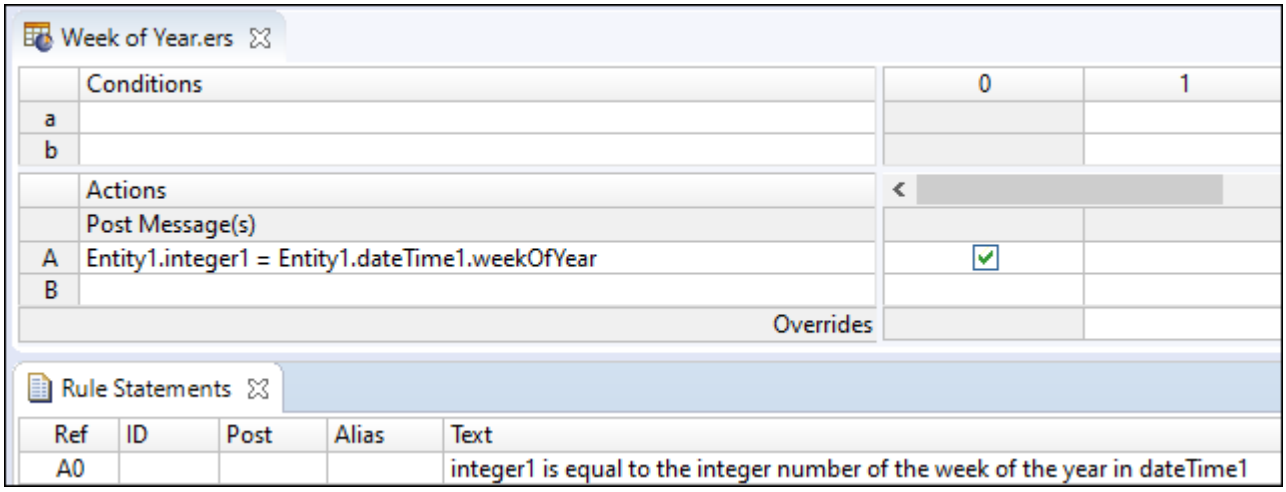
Returns an Integer from 1 to 52, equal to the week number within the year in <DateTime>. A week begins on Sunday and ends on Saturday. When a year ends between Sunday and the next Friday, or in other words when a new year begins between Monday and the next Saturday, the final day(s) of December will be included in week 1 of the new year. For example, 12/29/2013 fell on a Sunday, so 12/29-31 are included in week 1 of 2014.

USAGE RESTRICTIONS

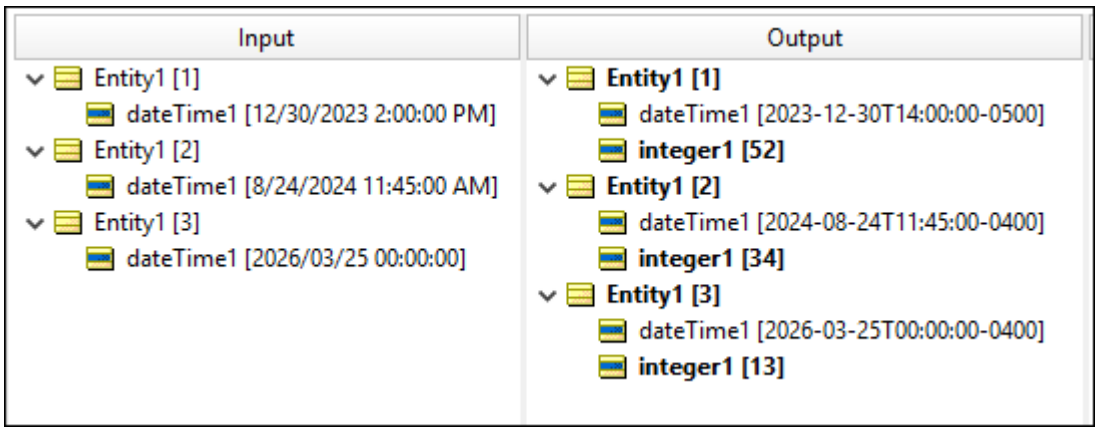
The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.weekOfYear** to assign a value to `integer1`.



SAMPLE TEST



Weeks between

SYNTAX

`<DateTime1>.weeksBetween(<DateTime2>)`

DESCRIPTION

Returns the Integer number of weeks between DateTimes. This function calculates the number of milliseconds between the date values and divides that number by 86,400,000 (the number of milliseconds in a day). Any fraction is truncated, leaving an Integer result. If the two dates differ by less than a full 24-hour period, the value returned is zero. A positive Integer value is returned when `<DateTime2>` occurs after `<DateTime1>`.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions

RULESHEET EXAMPLE

The following Rulesheet uses **.weeksBetween** to determine the number of weeks that have elapsed between `dateTime1` and `dateTime2`, compare it to the values in the Condition cells, and assign a value to `string1`.

weeksBetween.ers

	Conditions	1	2
a	Entity1.dateTime1.weeksBetween(Entity1.dateTime2)	<= 10	> 10
b			
	Actions		
	Post Message(s)		
A	Entity1.string1	'Not Overdue'	'Overdue'
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If dateTime1 and dateTime2 are fewer than 10 weeks apart, set Entity1.string1 to 'Not Overdue'
2				If dateTime1 and dateTime2 are more than 10 weeks apart, set Entity1.string1 to 'Overdue'

SAMPLE RULETEST

A sample Ruletest provides `dateTime1` and `dateTime2` for two examples. Input and Output panels are shown below.

Input	Output
Entity1 [1]	**Entity1 [1]**
dateTime1 [11/15/1847 00:00:00]	dateTime1 [1847-11-15T00:00:00-0500]
dateTime2 [6/15/1865 00:00:00]	dateTime2 [1865-06-14T23:00:00-0500]
Entity1 [2]	**string1 [Overdue]**
dateTime1 [11/15/1947 00:00:00]	**Entity1 [2]**
dateTime2 [12/15/1947 00:00:00]	dateTime1 [1947-11-15T00:00:00-0500]
Entity1 [3]	dateTime2 [1947-12-15T00:00:00-0500]
dateTime1 [11/15/2047 00:00:00]	**string1 [Not Overdue]**
dateTime2 [6/15/2048 00:00:00]	**Entity1 [3]**
	dateTime1 [2047-11-15T00:00:00-0500]
	dateTime2 [2048-06-15T00:00:00-0400]
	string1 [Overdue]

Year

SYNTAX

`<DateTime>.year`

DESCRIPTION

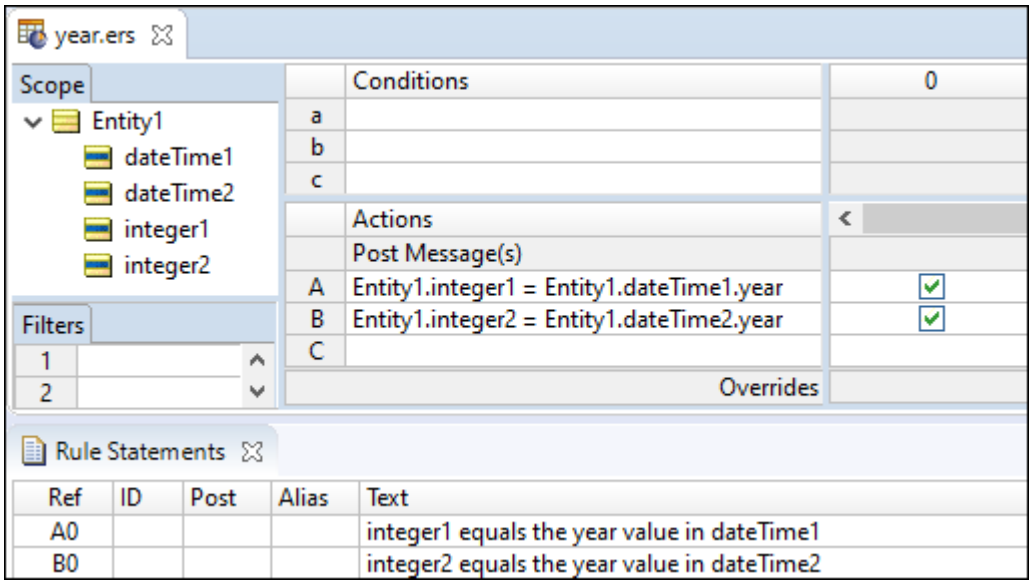
Returns the century/year portion of `<DateTime>`. The returned value is a four digit Integer.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

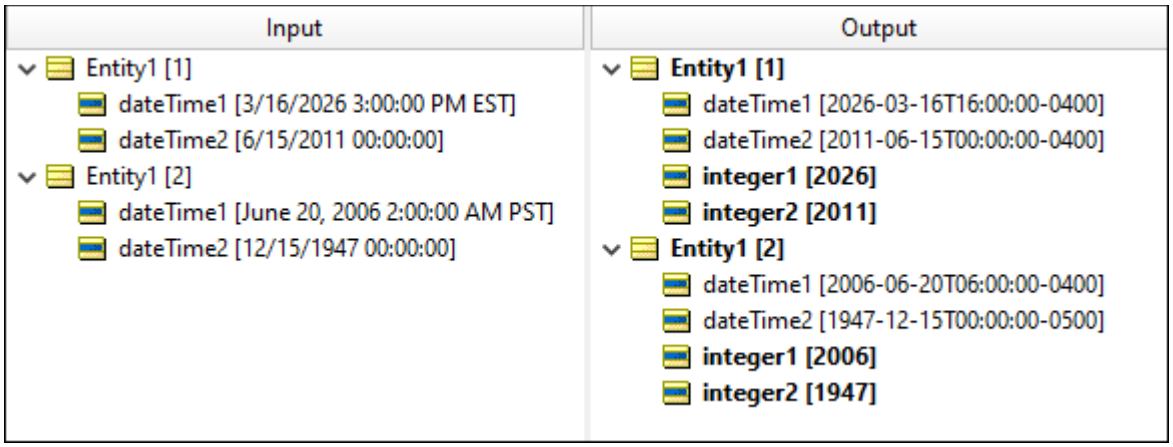
RULESHEET EXAMPLE

The following Rulesheet uses **.year** to evaluate `dateTime1` and `dateTime2` and assign the year values to `integer1` and `integer2`, respectively.



SAMPLE TEST

A sample Ruletest provides two examples of `dateTime1` and `dateTime2`. Input and Output panels are shown below:



Years between

SYNTAX

`<DateTime1>.yearsBetween(<DateTime2>)`

DESCRIPTION

Returns the Integer number of years between DateTimes. The number of months in <DateTime2> is subtracted from the number of months in <DateTime1>, and the result is divided by 12 and truncated. This function returns a positive number if <DateTime2> is later than <DateTime1>.

USAGE RESTRICTIONS

The Operators row of the table in Summary Table of Vocabulary Usage Restriction applies. No special exceptions.

RULESHEET EXAMPLE

The following Rulesheet uses **.yearsBetween** to determine the number of years that have elapsed between dateTime1 and dateTime2, compare it to the Values set, and assign a value to string1.

YearsBetween.ers

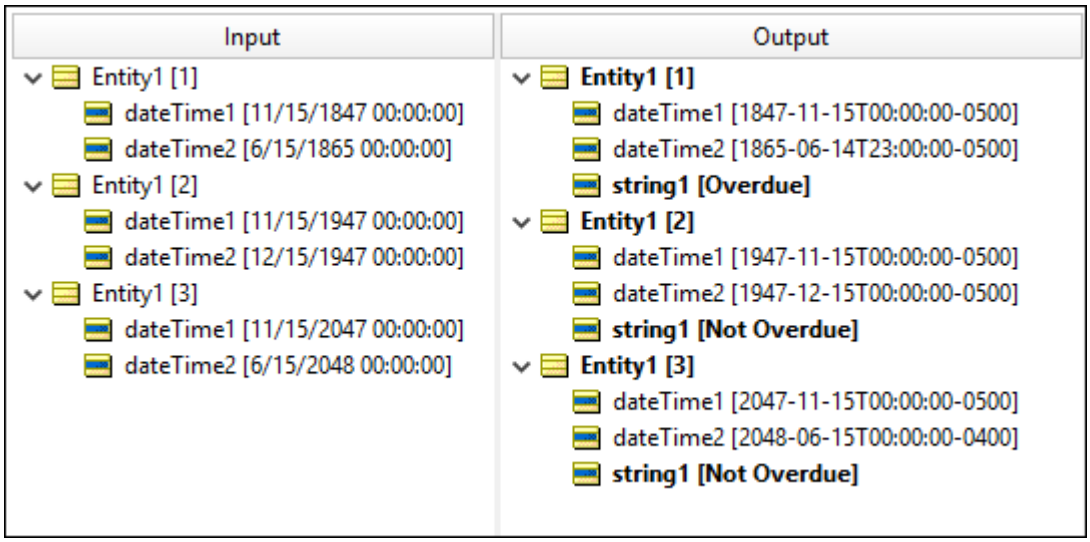
	Conditions	1	2
a	Entity1.dateTime1.yearsBetween(Entity1.dateTime2)	<= 10	> 10
b			
	Actions		
	Post Message(s)		
A	Entity1.string1	'Not Overdue'	'Overdue'
B			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If 10 or fewer years have elapsed between dateTime1 and dateTime2, then Entity1 is not overdue
2				If more than 10 years have elapsed between dateTime1 and dateTime2, then Entity1 is overdue

SAMPLE TEST

A sample Ruletest provides dateTime1 and dateTime2 for three examples. Input and Output panels are shown below.



A

Standard Boolean constructions

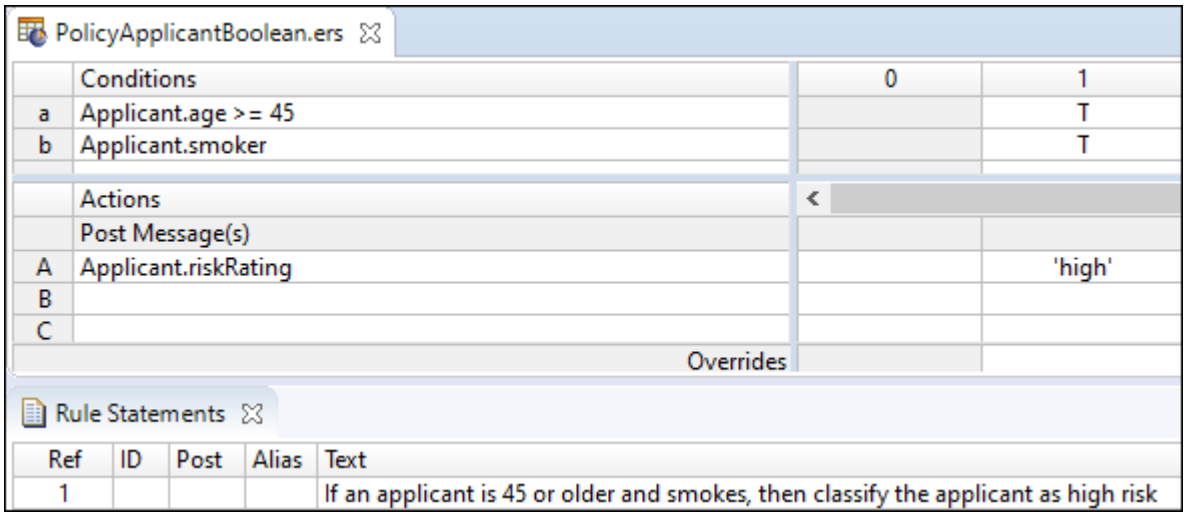
The topics in this section presents several standard truth tables (AND, NAND, OR, XOR, NOR, and XNOR) with examples of usage in a Rulesheet.

For details, see the following topics:

- Boolean AND
- Boolean NAND
- Boolean OR
- Boolean XOR
- Boolean NOR
- Boolean XNOR

Boolean AND

In a decision table, a rule with **AND'**ed Conditions is expressed as a single column, with values for each Condition aligned vertically in that column. For example:



PolicyApplicantBoolean.ers

	Conditions	0	1
a	Applicant.age >= 45		T
b	Applicant.smoker		T
	Actions		
	Post Message(s)		
A	Applicant.riskRating		'high'
B			
C			
	Overrides		

Rule Statements

Ref	ID	Post	Alias	Text
1				If an applicant is 45 or older and smokes, then classify the applicant as high risk

In this scenario, each Condition has a set of 2 possible values:

person is 45 or older: `{true, false}`

person is a smoker: `{true, false}`

and the outcome may also have two possible values:

person's risk rating: `{low, high}`

These Conditions and Actions yield the following truth table:

age >= 45	smoker	risk rating
`true`	`true`	`high`
`true`	`false`	
`false`	`true`	
`false`	`false`	

Note that we have only filled in a single value of risk rating, because the business rule above only covers a single scenario: where `age >= 45` and `smoker = true`. Running *The completeness checker* as described in the *Rule Modeling section* quickly identifies the remaining three scenarios:

PolicyApplicantBoolean.ers

	Conditions	1	2	3
a	Applicant.age >= 45	T	T	F
b	Applicant.smoker	T	{F, null}	-
c				
	Actions			
	Post Message(s)			
A	Applicant.riskRating	'high'		
B				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1				If an applicant is 45 or older and smokes, then classify the applicant as high risk

Completing the truth table and the Rulesheet requires the definition of 2 additional business rules:

PolicyApplicantBoolean.ers

	Conditions	1	2	3
a	Applicant.age >= 45	T	T	F
b	Applicant.smoker	T	{F, null}	-
	Actions			
	Post Message(s)			
A	Applicant.riskRating	'high'	'low'	'low'
B				
C				
	Overrides			

Rule Statements

Ref	ID	Post	Alias	Text
1				If an applicant is 45 or older and smokes, then classify the applicant as high risk
2				If an applicant is 45 or older and does NOT smoke, then classify the applicant as low risk
3				If an applicant is NOT 45 or older, then ignore whether or not the applicant smokes and classify them as low risk

and updating the truth table, we recognize the classic **AND** Boolean function.

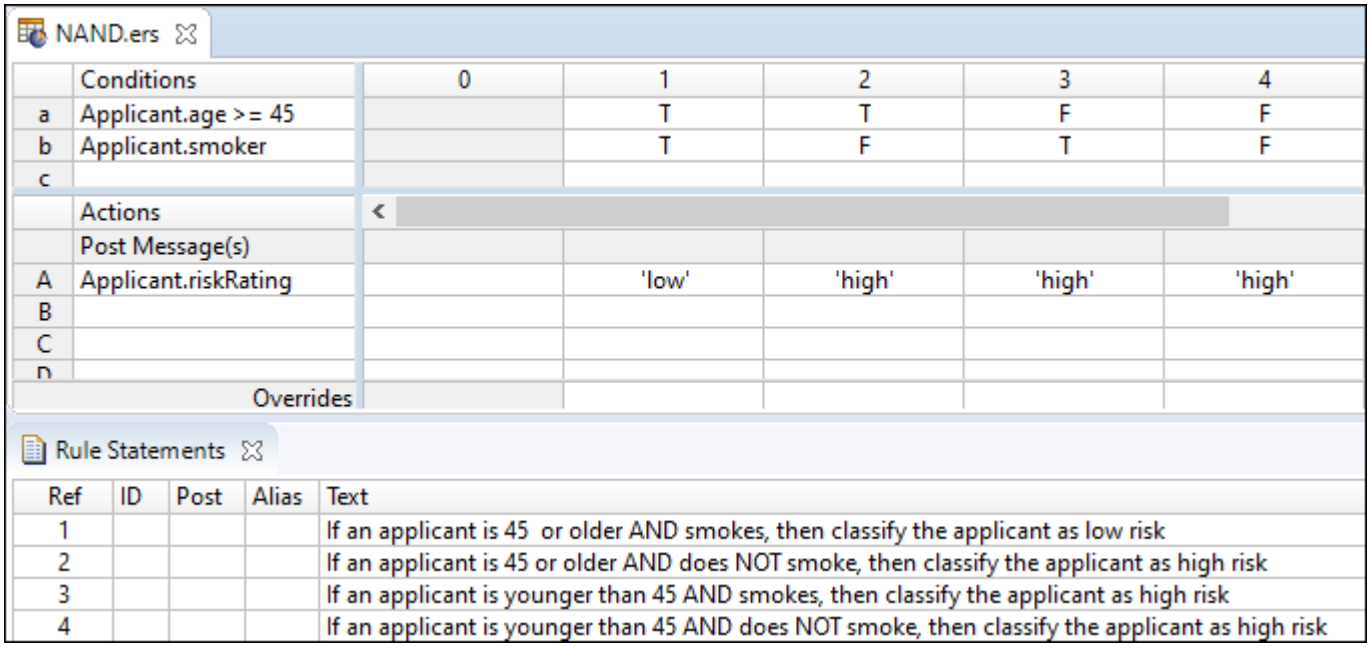
age >= 45	smoker	risk rating
`true`	`true`	`high`
`true`	`false`	`low`
`false`	`true`	`low`
`false`	`false`	`low`

Once the basic truth table framework has been established in the Rulesheet by the Completeness Checker – in other words, all logical combinations of Conditions have been explicitly entered as separate columns in the Rulesheet – we can alter the outcomes to implement other standard Boolean constructions. For example, the **NAND** construction has the following truth table:

Boolean NAND

age >= 45	smoker	risk rating
true	true	low
true	false	high
false	true	high
false	false	high

Also known as “Not And”, this construction is shown in the following Rulesheet:



NAND.ers

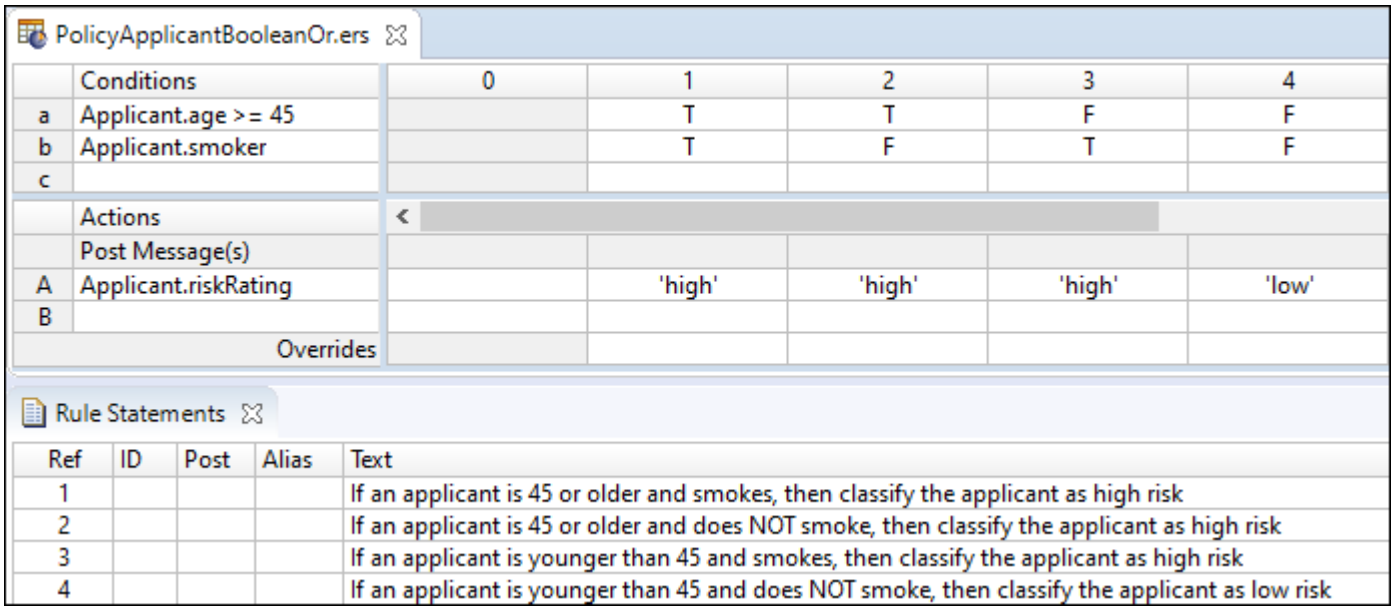
	Conditions	0	1	2	3	4
a	Applicant.age >= 45		T	T	F	F
b	Applicant.smoker		T	F	T	F
c						
	Actions					
	Post Message(s)					
A	Applicant.riskRating		'low'	'high'	'high'	'high'
B						
C						
D						
	Overrides					

Rule Statements

Ref	ID	Post	Alias	Text
1				If an applicant is 45 or older AND smokes, then classify the applicant as low risk
2				If an applicant is 45 or older AND does NOT smoke, then classify the applicant as high risk
3				If an applicant is younger than 45 AND smokes, then classify the applicant as high risk
4				If an applicant is younger than 45 AND does NOT smoke, then classify the applicant as high risk

Boolean OR

age >= 45	smoker	risk rating
true	true	high
true	false	high
false	true	high
false	false	low



PolicyApplicantBooleanOr.ers

	Conditions	0	1	2	3	4
a	Applicant.age >= 45		T	T	F	F
b	Applicant.smoker		T	F	T	F
c						
	Actions					
	Post Message(s)					
A	Applicant.riskRating		'high'	'high'	'high'	'low'
B						
	Overrides					

Rule Statements

Ref	ID	Post	Alias	Text
1				If an applicant is 45 or older and smokes, then classify the applicant as high risk
2				If an applicant is 45 or older and does NOT smoke, then classify the applicant as high risk
3				If an applicant is younger than 45 and smokes, then classify the applicant as high risk
4				If an applicant is younger than 45 and does NOT smoke, then classify the applicant as low risk

Boolean XOR

Using “Exclusive Or” logic, `riskRating` is high whenever the age or smoker test, but not both, is satisfied. This construction is shown in the following Rulesheet:

age >= 45	smoker	risk rating
`true`	`true`	`low`
`true`	`false`	`high`
`false`	`true`	`high`
`false`	`false`	`low`

XOR.ers

	Conditions	0	1	2	3	4
a	Applicant.age >= 45		T	T	F	F
b	Applicant.smoker		T	F	T	F
c						
	Actions					
	Post Message(s)					
A	Applicant.riskRating		'low'	'high'	'high'	'low'
B						
	Overrides					

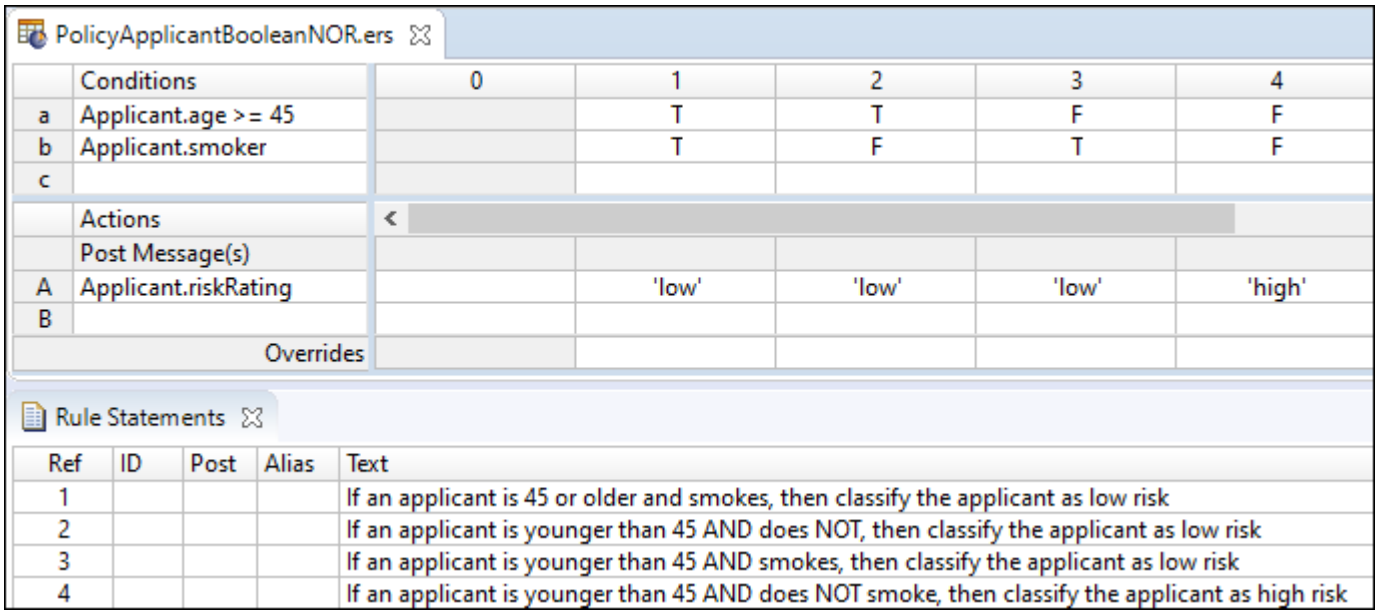
Rule Statements

Ref	ID	Post	Alias	Text
1				If an applicant is 45 or older AND smokes, then classify the applicant as low risk
2				If an applicant is 45 or older AND does NOT smoke, then classify the applicant as high risk
3				If an applicant is younger than 45 AND smokes, then classify the applicant as high risk
4				If an applicant is younger than 45 AND does NOT smoke, then classify the applicant as low risk

Boolean NOR

Also known as “Not Or”, this construction is shown in the following Rulesheet:

age >= 45	smoker	risk rating
`true`	`true`	`low`
`true`	`false`	`low`
`false`	`true`	`low`
`false`	`false`	`high`



PolicyApplicantBooleanNOR.ers

	Conditions	0	1	2	3	4
a	Applicant.age >= 45		T	T	F	F
b	Applicant.smoker		T	F	T	F
c						
	Actions					
	Post Message(s)					
A	Applicant.riskRating		'low'	'low'	'low'	'high'
B						
	Overrides					

Rule Statements

Ref	ID	Post	Alias	Text
1				If an applicant is 45 or older and smokes, then classify the applicant as low risk
2				If an applicant is younger than 45 AND does NOT, then classify the applicant as low risk
3				If an applicant is younger than 45 AND smokes, then classify the applicant as low risk
4				If an applicant is younger than 45 AND does NOT smoke, then classify the applicant as high risk

Boolean XNOR

Also known as “Exclusive NOR”, this construction is shown in the following Rulesheet:

age >= 45	smoker	risk rating
`true`	`true`	`high`
`true`	`false`	`low`
`false`	`true`	`low`
`false`	`false`	`high`

PolicyApplicantBooleanXNOR.ers

	Conditions	0	1	2	3	4
a	Applicant.age <= 45		F	T	F	T
b	Applicant.smoker		F	T	T	F
c						
	Actions					
	Post Message(s)					
A	Applicant.riskRating		'high'	'high'	'low'	'low'
B						
C						
	Overrides					

Rule Statements

Ref	ID	Post	Alias	Text
1				If an applicant is 45 or older AND does NOT smoke, then classify the applicant as high risk
2				If an applicant is younger than 45 AND smokes, then classify the applicant as high risk
3				If an applicant is 45 or older AND smokes, then classify the applicant as low risk
4				If an applicant is younger than 45 AND does NOT smoke, then classify the applicant as low risk

B

Precedence of rule operators

The precedence of operators affects the grouping and evaluation of expressions. Expressions with higher-precedence operators are evaluated first. Where several operators have equal precedence, they are evaluated from left to right. The following table summarizes Corticon.js's operator precedence.

Operator precedence	Operator	Operator Name	Example
1	( )	Parenthetic expression	(5.5 / 10)
2	-	Unary negative	-10
	not	Boolean test	not 10
3	*	Arithmetic: Multiplication	5.5 * 10
	/	Arithmetic: Division	5.5 / 10
	**	Arithmetic: Exponentiation (Powers and Roots)	5 ** 2 25 ** 0.5 125 ** (1.0/3.0)
4	+	Arithmetic: Addition	5.5 + 10
	-	Arithmetic: Subtraction	10.0 – 5.5
5	<	Relational: Less Than	5.5 < 10
	<=	Relational: Less Than Or Equal To	5.5 <= 5.5
	>	Relational: Greater Than	10 > 5.5
	>=	Relational: Greater Than Or Equal To	10 >= 10
	=	Relational: Equal	5.5=5.5
	<>	Relational: Not Equal	5.5 <> 10
6	(*expression* and *expression*)	Logical: AND	(ent1.dec1 > 5.5 and ent1.dec1 < 10)
	(*expression* or *expression*)	Logical: OR	(ent1.dec1 > 5.5 or ent1.dec1 < 10)

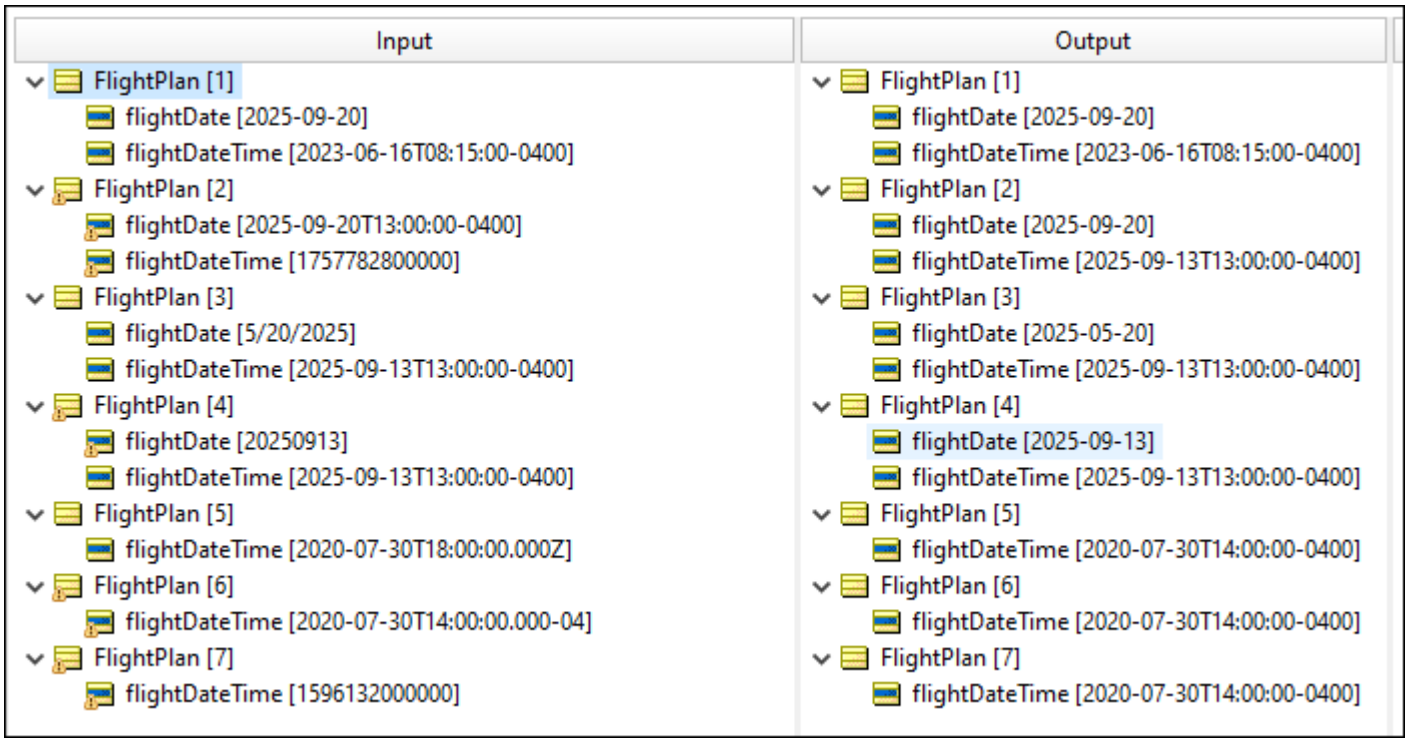
Note: While expressions within parentheses that are separated by logical AND / OR operators are valid, the component expressions are not evaluated individually when testing for completeness, and might cause unintended side effects during rule execution. Best practice within a Corticon.js Rulesheet is to represent AND conditions as separate condition rows and OR conditions as separate rules -- doing so allows you to get the full benefit of Corticon.js's logical analysis.

Note: It is recommended that you place arithmetic exponentiation expressions in parentheses.

C

Formats for date and dateTime in Corticon.js Studio tester

When you have Ruletests in your Corticon.js project, they require adherence to dates and dateTimes that are ISO 8601 compliant, or as the number of milliseconds since the epoch. For testing , the Corticon.js Studio Tester lets you enter a more permissive date and dateTime as Input values; however, the Output values are always strict ISO 8601, as illustrated:



The default date mask in the `brms.properties` file is:

```
Default DateMask to be applied to DateTime values in Tester Output Tree
Default is yyyy-MM-dd'T'HH:mm:ssZ (e.g. 2020-01-01T00:00:00-0000)
com.corticon.javascript.studio.tester.dateformat=yyyy-MM-dd'T'HH:mm:ssZ
```

D

Formats for date and dateTime in JSON payloads

Date and DateTime values in a Corticon.js payload must conform to the ISO 8601 standard, or as the number of milliseconds since the epoch. The Corticon.js runtime does all its computation using UTC as the time reference. At runtime, Corticon.js always returns date or dateTime formatted as ISO 8601 regardless of the input format you used.

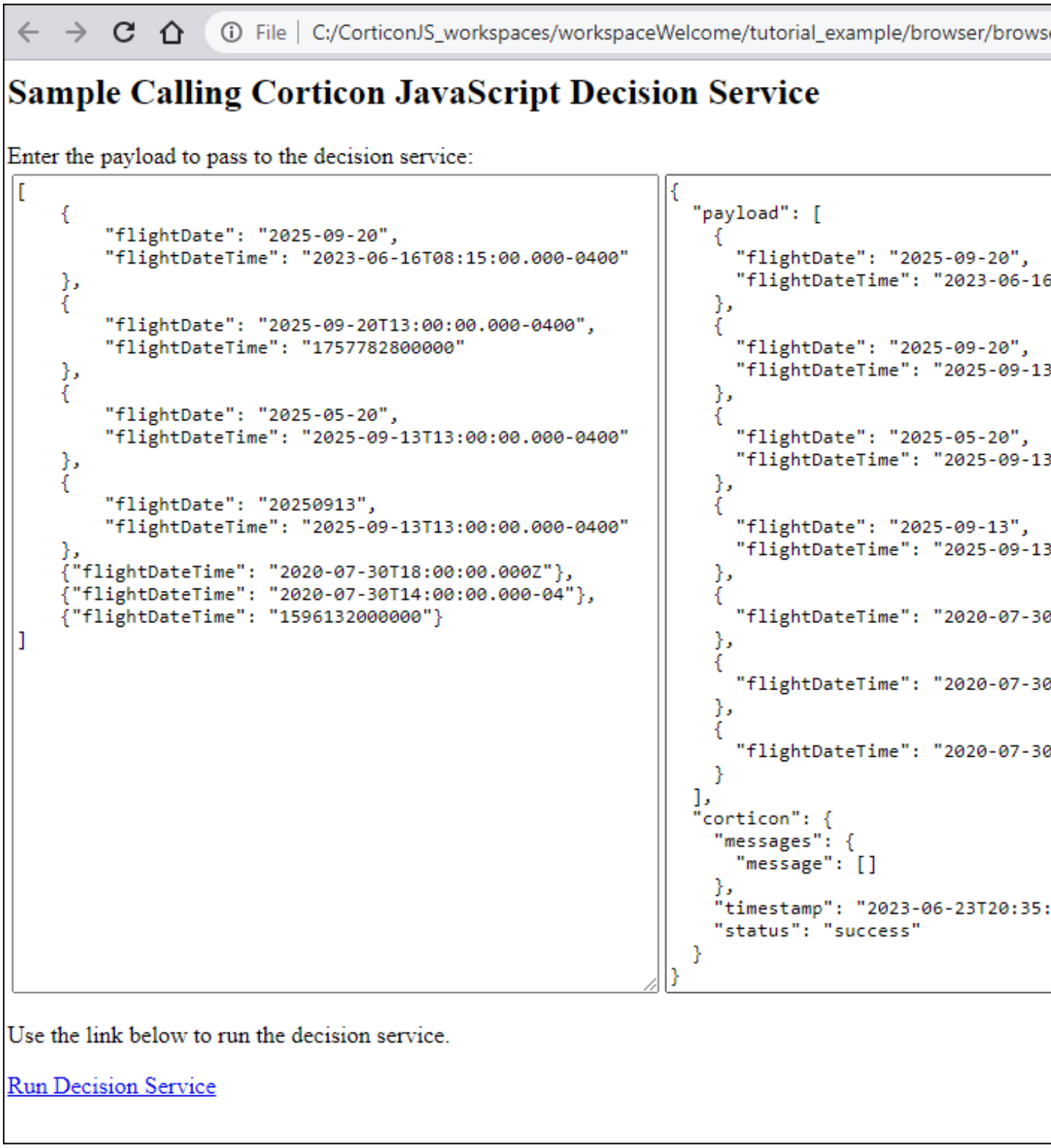
Date and dateTime values that are passed into Corticon.js at runtime must conform to the ISO 8601 standard. If a non-ISO 8601 date or dateTime is passed inside the payload to the Decision Service, the JavaScript runtime will fail to parse it, and the execution will terminate.

There are many different formats included in the ISO 8601 standard, not limited to, “Date”, “Date and Time”, and “Date and Time with Time Zone offset”. When you use a full Date and Time with Time Zone offset value inside your payload, you remove any potential misinterpretation of the dateTime value in the JavaScript runtime, which may influence how certain Rules will fire. If Time is missing, it defaults to midnight (00:00:00). If Time Zone is missing, it defaults to the Time Zone of the machine running the JavaScript runtime.

If you wanted to pass the date time (UTC) “Thursday, July 30, 2020 6:00:00 PM” into a decision service as a JSON payload, you could format it as:

1. **2020-07-30T18:00:00.000Z**—This represents the dateTime as ISO 8601 directly with the UTC reference (that is, with no time zone specified).
2. **2020-07-30T14:00:00.000-04**—This represents the dateTime as ISO 8601 with a time zone (notice instead of 18 hour, it is 14). This format is accepted as input but will not return the results in the same notation.
3. **1596132000000**—This format represents the dateTime as the number of milliseconds since the epoch, 1/1/1970.

When you package the project for JS deployment as Browser, and then **Run Decision Service**, you get the following HTML output:



Note: For more information, see the blogs Dealing With Date and Time With Corticon.js and Understanding ISO 8601 Date and Time Format

E

Escape sequences on JavaScript string literals

A character preceded by a backslash (\) is an *escape sequence* and has special meaning to the compiler. The following table shows the JavaScript escape sequences:

Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a form feed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

When an escape sequence is encountered in a print statement, the compiler interprets it accordingly. For example, if you want to put quotes within quotes you must use the escape sequence, \", on the interior quotes.

JavaScript specifics:

- **Template Literals:** JavaScript's template literals (backticks ``) introduce additional escaping rules for embedded expressions (`${}`) and backticks themselves (\` `).
- **Vertical Tab** (`\v`): While both support `\v`, older versions of Internet Explorer might interpret it as a literal 'v' instead of a vertical tab, making `\x0B` a more robust cross-browser alternative.
- **Null Character** (`\0`): JavaScript's `\0` represents the null character (U+0000) only if the next character is not a decimal digit; otherwise, it's interpreted as an octal escape sequence.
- **Unicode Escapes:** Both support Unicode escapes, but JavaScript also allows for extended Unicode escape sequences using curly braces, e.g., `\u{1F600}` for emojis.