RiverWare

Technical Documentation Version 7.3

RPL Data Types and Palette

Center for Advanced Decision Support for
Water and Environmental Systems (CADSWES)
UNIVERSITY OF COLORADO **BOULDER**

RPL Data Types and Palette
Table of Contents

RPL Data Types and Palette

This document describes and presents examples of each of the Data Types used in RPL. Then the RPL palette buttons are described.

1. Expression Data Types

The RiverWare Policy Language language is composed of seven data types, such as numeric values, datetimes, text strings, and slots. The data types are the building blocks of RPL. They are the only types of data which expressions and functions are allowed to evaluate to and are the structure controlling what information can be entered in different parts of the RPL. The structure editor uses expression data types to enforce block and function validity as they are constructed. Attempting to enter an expression of one type where another type is expected, immediately results in an error.

Unspecified expressions are pieces of the RPL which have not yet been defined when creating a RPL set. Unspecified expressions are shown in the **Rule Editor** and **Function Editor** as the name of the expression type, colored blue, and surrounded by small angle braces (< and >). An unspecified expression may be filled in directly with the appropriate data, or it may be replaced with a function which will evaluate to the appropriate data type. In most cases, unspecified expressions can be completed by using **Palette** functions. This is the recommended approach. The complexity and syntactical requirements of some expression types, particularly object and slot, behooves you to use the built-in tools for formatting a correct expression.

Example: A rule contains an unspecified numeric expression:
<numeric expr>

The expression may be filled in with an actual value and units by typing:
100.0 cfs

or it could be replaced by an arithmetic function which evaluates to a value and units by clicking on the **N + N** palette button:
<numeric expr> + <numeric expr>

or it could be replaced by yet another function which returns a value and units, such as a series slot lookup:
Lake Mead.Outflow []

In all of these cases, the original requirement that the expression evaluate to a number and units is satisfied.

The expression data types are listed below. More complete descriptions follow:

Expression Type	**Unspecified Representation**	**Description**	**Examples**
NUMERIC	**<numeric expr>**	A number and its units.	**150 cfs** **349.47926 “acre-feet/day”** **1 NONE**
BOOLEAN	**<boolean expr>**	True or False.	**TRUE** **FALSE**
DATETIME	**<datetime expr>**	An actual or symbolic date.	**@“t”, @“t-1”** **@“Current Timestep”** **@“Previous Timestep”** **@“January 4, Current Year”**
STRING	**<string expr>**	Quoted text and/or numbers.	**“Entering spring runoff season.”** **“Minimum flow violated by 5%“**
OBJECT	**<object expr>**	An object in the current model.	**%“Havasu”** **%“Kentucky-Barkley Canal”** **%“SJBelowNavajo::ColoradoAg”**
SLOT	**<slot expr>**	A slot on an object in the current model.	**$“Lake Mead.Outflow”** **$“Kentucky.Pool Elevation”** **$“Rio Chama.Hydrologic Inflow”**

LIST	**<list expr>**	A list of any mix of data types.	**{ 1561 [“ft”], 1573 [“ft”], 1589 [“ft”] }**
			{ @“April 15, 1999” }
			{ %“Powell”, 10 [“cfs”], TRUE }
			{ { %“Mead”, TRUE }, { $“Inflow” } }

1.1 NUMERIC

The numeric expression data type is comprised of a number and its units. Numeric expressions may be typed directly into a block or function by double-clicking the unspecified **<numeric expr>** and typing a value and units into the textfield. As in Simulation, numbers are stored and evaluated internally with twelve significant figures of precision. When a numeric expression is typed directly into a block or function, the default display precision is eight digits beyond the decimal point. The precision of displayed numbers can be modified by scrolling the **Precision:** spinner in the special options portion of the **RPL Set Editor**.

Units for numeric expressions must follow the value, separated by a single space. If no units are specified, the value is considered to be of a dimensionless unit type with units of “NONE.” When a numeric expression is typed directly into a block or function, the units may be entered without quotes if the unit name contains no special punctuation characters such as hyphens or forward slashes. When quotes are omitted, they will be filled in automatically.

Example: Typing in numeric expressions with the display precision set to the default of 8, causes the following text to be shown in the editor.

Exact Text Typed Into Textfield	Resulting Display
150 cfs	**150.00000000 “cfs”**
0.000000004 “feet/day”	**0.00000000 “feet/day”**
0.5 NONE	**0.50000000**
248650 acre-feet	“parse error” due to hyphen, therefore you must type the quotes

1.2 BOOLEAN

The boolean expression data type can be either true or false. The expressions may be entered directly in the editor by double-clicking the unspecified **<boolean expr>**, and typing into the textfield. The editor accepts lowercase, uppercase and capitalized spellings, but automatically converts these entries into an uppercase format.

Example: Typing in boolean expressions (as shown below) cause the following text to be shown in the editor.

Exact Text Typed Into Textfield	Resulting Display
true	**TRUE**
False	**FALSE**
FALSE	**FALSE**
tRuE	“parse error”

1.3 DATETIME

The datetime expression type is used to represent moments in time. Datetimes may be entered directly by double-clicking an unspecified **<datetime expr>**, and typing into the resulting textfield. All datetime expressions begin with an **@** symbol and are followed by the datetime specification in double quotes (“ and ”).

Datetime expressions may be specified symbolically. Symbolic representations of points in time are common in everyday language, but rare in programming languages. Symbolic representations include such datetimes as “next week,” “20th day of month,” and “April, next year.” These types of datetime specifications **are** allowed in the RiverWare Policy Language. They facilitate the writing of RPL expressions by using date conventions to which you are more accustomed. They are evaluated by the

RPL language, in the context in which they are used, to determine an exact moment in time. Slot access is still performed internally with exact datetime specification in number of seconds since New Year’s, 1700 C.E.

1.3.1 Fully or Partially Specified

Datetime expressions fall into two categories and each may be used in specific instances or contexts:

- **Fully specified**: Fully specified datetimes are those which can be mapped directly to an instant in time, such as @“t” or @“June 4, 1986.” Fully specified datetimes are required for all slot lookups, slot assignments and predefined function arguments.
- **Partially specified**: Partially specified datetimes are those which cannot be mapped to a specific instant in time, such as @“Current Month” or @“Tuesday.” Partially specified datetimes are used in boolean comparisons. For example, a check for whether the current timestep is a Tuesday could be done with a fully specified datetime, @“t”, and a partially specified datetime, @“Tuesday”:

  IF (@“t” == @“Tuesday”) THEN ...

If you have a partially specified datetime that you want to convert into a fully specified datetime, use the CompletePartialDate predefined function, **HERE (RPLPredefinedFunctions.pdf, Section 19)**.

1.3.2 Formats

There are four general datetime formats. Within each format, there are specific components which the user can specify. The four formats and the components which you can specify separately are shown in the following table.

Name	Time components (from lowest to highest resolution)	Components you can specify separately	Examples
1. Month, day, year	Year, Month, Day of Month, Hour, Minute, Second		@“August 23, 1997 4:00:00” @“7/22/1997 4:0000”
		Month	@“August”, @“Month 8”, @“Current Month”
		Day of Month	@“DayOfMonth 23”, @“Previous DayOfMonth”
		Year	@“Year 1997”, @“Current Year” , @“... Year”
		Time	@“4:12:00”
2. Timestep	Timesteps since the beginning of run		@“Timestep 12”, @“t+2”
		Timestep	@“Max Timestep”, @“t-1”
3. Day, week, year	Year, Week of Year, Day of Week, Hour, Minute, Second		@“Monday Week 10, 1997 4:00:00” @“DayOfWeek 5 Week 12, 1998, 12:00:00”
		Day of Week	@“Monday”, @“DayOfWeek 7”, @“Next DayOfWeek”
		Day of Week and Time	@“Monday 4:12”, @“4:12 DayOfWeek 1”, @“4:12 Finish DayOfWeek”
4. Day, year	Year, Day of Year, Hour, Minute, Second		@“DayOfYear 227, 1997 4:00:00”

Name	Time components (from lowest to highest resolution)	Components you can specify separately	Examples
		Day of Year	@“DayOfYear 23”, @“Start DayOfYear”
		Day of Year and Time	@“DayOfYear 23 4:12”, @“4:12 Current DayOfYear”

Thus, for the first format, you can specify almost any component. For the second format, you can only specify the timestep and nothing else. So you **cannot** specify @“Timestep 12, Year 1997”; an error would return.

As you specify components, you only specify as much as is needed to uniquely reference the date. So, if in a boolean comparison you are comparing a date to see it is in 1997, you would enter (date == @“Year 1997”). But, if you want to compare if it is March of 1997, you enter @“March 1997” but do not include the word “Year” before 1997.

The list of symbolic specifiers include the following: Min, Max, Start, Finish, Previous, Current, and Next. Any of these specifiers can be used in the examples above.

If no time (hours, minutes, seconds) is specified at all, it defaults to 24:00:00. If a time is specified, it may appear either before or after the specification of the Month, Day, and Year. Times may be specified with or without seconds, if they are not specified then they default to 0 (e.g., both 17:23 and 17:23:00 refer to the same time of day).

1.3.3 Examples

Mixing of symbolic datetime elements results in an almost infinite number of datetime specifications. Examples for some of these acceptable fully specified expressions and their interpretations are provided below. The symbolic datetime elements shown can be combined into other forms not explicitly enumerated.

Fully Specified Datetime Expression	Interpretation
@“4/01/1996 14:00:00”	April 1, 1996, 2 p.m.
@“April 1, 1996 14:00”	April 1, 1996, 2 p.m.
@“April 1, 1996 14:00:00”	April 1, 1996, 2 p.m.
@“t” or @“Current Timestep”	the current timestep
@“t-1” or @“Previous Timestep”	the previous timestep
@“t+1” or @“Next Timestep”	the next timestep
@“Start Timestep”	the begin timestep in the run or the begin timestep in an expression slot (in the evaluation range).
@“Finish Timestep”	the finish timestep in the run or the last timestep in an expression slot (in the evaluation range).
@“April 1, Current Year 14:00:00”	April 1 of the current year, 2 p.m.

@“April 1, 1996 Current Hour:00:00”	April 1, 1996 at the hour of the current timestep.
@“April Next DayOfMonth, 1996 Previous Hour:00:00”	Depends on the current timestep
@“7/22/1997 1:34:00 + 2 Days”	the date/time which is 48 hours after the given date (i.e., 7/24/97 1:34:00)
@“+ 25”	the date/time 25 timesteps beyond the current timestep
@“Next Timestep - 1”	Confusing way to refer to current timestep
@“2 hours before Current Timestep”	2 hours before current timestep

Partially specified datetime expressions include:

Partially Specified Datetime Expression	Interpretation
@“Year 1997”	the year specified as 1997
@“Previous Year”	the year before the current year
@“Current Year”	the year specified as the year of the current timestep
@“Month 4”	the month specified as April
@“April”	the month specified as April
@“April 22”	the 22nd day in April
@“6:00 April 22”	6:00 AM on April 22
@“April 22 6:00”	6:00 AM on April 22
@“6:00”	6:00 AM
@“DayOfMonth 4”	the day of the month specified as the 4th
@“Next DayOfMonth”	the day of the month specified as the day after the day of the current timestep
@“Max DayOfMonth”	the day of the month that is the last day in the month
@“Monday”	the day of week specified as Monday
@“WeekOfYear 2”	the week of year specified as the second
@“DayOfWeek 2”	the day of week specified as Monday
@“6:00 Previous DayOfWeek”	6:00 AM on the previous day of the week
@“DayOfYear 157”	the day of year specified as 157
@“Max DayOfYear”	the day corresponding to the last day of the year

The previous examples do not capture all the possible combinations of partially specified datetime expressions.

1.3.4 Datetime Math

When performing addition or subtraction math on datetime values (for example, adding 3 timesteps to the current timestep), there are three approaches that can be used:

A. Math within a datetime specification: Within a datetime literal, it is possible to add or subtract integral values from a base time. Thus, possible specifications for a datetime value three timesteps beyond the current timestep include:

@“t + 3” or

@“t + 3 timestep” or

@“t + 3 day” or

@“t + 72 hour”

Note that the date time math occurs within the quoted literal expression, which can not include references to variables, that the integral increment is assumed to have units of “timestep” when the units are unspecified, and that the latter two specifications are less general because they assume that the model has a daily timestep.

This approach to datetime math is probably the easiest to read, but is of course only useful when literal specification is possible. For example, when one would like to increment a base datetime by a number of timesteps which is itself the result of expression evaluation, then one of the alternative approaches to datetime math is required.

B. Mathematical expressions involving a datetime operand: A numeric value can be added to or subtracted from a datetime value. Continuing the example from above, the following expressions would evaluate to three timesteps beyond the current timestep *in a daily model*:

@“t” + 72 “hour” or

@“t” + 3 “day”

Here are the math operations supported for datetime values:

<DATETIME> + <NUMERIC> results in a <DATETIME>

<NUMERIC> + <DATETIME> results in a <DATETIME>

<DATETIME> - <NUMERIC> results in a <DATETIME>

<DATETIME> - <DATETIME> results in a <NUMERIC> (with units of time).

In such expressions, the numeric value must have units of type Time. Note that “Timestep” is not a legal unit of Time, so this type of approach could not be used when one would like to add some number of timesteps in a model whose timestep increment is not fixed (i.e., monthly or annual timestep models).

The following operations are not supported as the result is undefined:

<NUMERIC> - <DATETIME>, <DATETIME> + <DATETIME>

Note: Within RPL evaluation, 1 "Month" is equal to 31 days and 1 "Year" is equal to 365 days as defined in the rplUnits file in the installation directory. If you are trying to compute a flow by dividing a volume by 1 "month", it will use 31 days for the month. If you then set that value on a monthly slot that only has 30 days, the values will be not be the same due to the conversion. Instead, convert the volume to a flow using VolumeToFlow and specify the month or divide by GetDaysInMonth and specify the desired month. For non constant timestep lengths, the best approach is to stay in volumes as long as possible, and then convert to a flow when ready to set the value on a flow type slot.

C. Using the OffsetDate function: The OffsetDate predefined function, **HERE (RPLPredefinedFunctions.pdf, Section 142)**, allows you to specify the increment and length of timestep to use for the addition or subtraction. Thus, you could enter:

OffsetDate(@ "t", 1, "1 Day")

This function is most useful for variable length timesteps (monthly, annual).

1.4 STRING

The string expression data type is any text surrounded by double quotes (" and "). An unspecified string expression may be entered directly in expression by double-clicking the **<string expr>**, and typing in the string surrounded by double quotes**.** String expressions may contain any combination of letters, numbers and punctuation except double quotes.

1.5 OBJECT

The object expression data type is used to reference objects in the currently loaded model. An unspecified object expression may be completed by double-clicking the **<object expr>** and typing in the object name in double quotes and preceded by a percent symbol. When an object expression is typed directly into a block or function, the object name may be entered without quotes if the object name contains no spaces or special punctuation characters. When quotes are omitted, they will be filled in automatically.

It is highly recommended that you do not type object names directly into blocks and functions. The potential for error is great. An extra space or an incorrectly capitalized letter will invalidate the object expression. There is, however, a convenient and foolproof way to enter this data. Most commonly, an unspecified object expression is completed by using the **Object Selector** in the RPL **Palette** or by typing in a variable name whose type is an object expression.

Examples of object expressions typed directly into the Editor include:

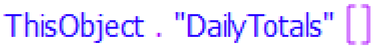
Exact Text Typed Into Textfield	Resulting Display
%"Lake Mead"	Lake Mead

% DataObj	**DataObj**
% Havasu Diversion	"parse error" due to space

In contexts where an object expression is used as part of a complete slot specification, the object expression must be followed by a <string expression> indicating the slot.

Example: The Pool Elevation of lake Powell could be filled into:
<object expr>.<string expr>
and result in the expression below:
Lake Powell."Pool Elevation"

Note: For Expression Slots and Object Level Account Methods, you can use the keyword ThisObject to access the containing object. For example, to get the slot named DailyTotals from this data object, you could create the expression shown in the image.



ThisObject . "DailyTotals" [ ]

1.6 SLOT

The slot expression data type is used to specify a specific slot on an object of the currently-loaded model. The expression must include the object on which the slot resides as well as the slot name itself. An unspecified slot expression may be completed by double-clicking the **<slot expr>**, typing in the object name, a dot, and the slot name, all in double quotes and preceded by a dollar symbol. When a slot expression is typed directly into a block or function, the object.slot name may be entered without quotes if neither of the names contain spaces or special punctuation characters. Regardless of how a slot expression is entered, the dollar sign and quotes are automatically omitted in the resulting display.

Again, it is highly recommended that you do not type object.slot names directly into blocks and functions. The same convenient and foolproof way to enter object data is available to enter object.slot data. An unspecified slot expression may be completed by using the **Slot Selector** in the RPL **Palette** or by typing in a variable name whose type is a slot expression. The **Slot Selector** is an interface similar to the one used elsewhere in RiverWare, which formats a selected object and slot correctly and inserts it into the slot expression.

Examples of slot expressions typed directly into the editor include:

Exact Text Typed Into Textfield	Resulting Display
$"Lake Mead.Outflow"	**Lake Mead.Outflow**
$ DataObj.target	**DataObj.target**
$ Havasu Diversion.Inflow	"parse error" due to space
$ "Havasu Diversion".Inflow	"parse error"

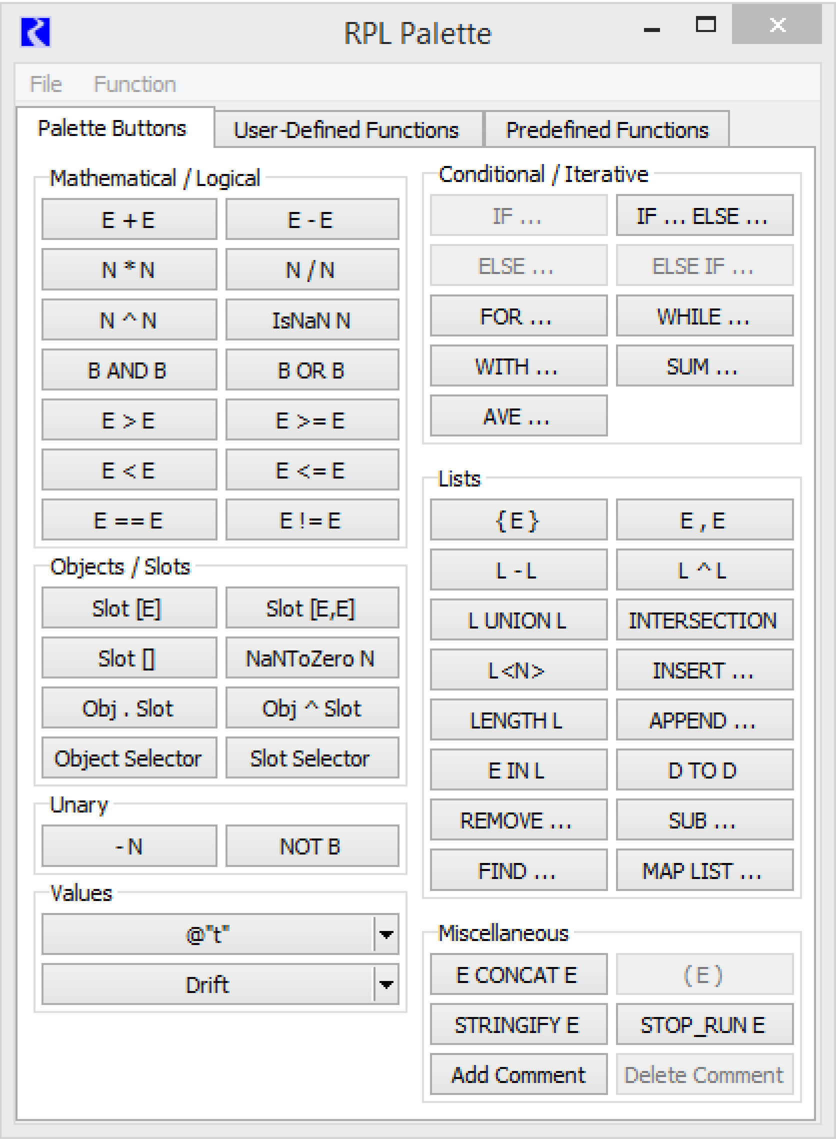
1.7 LIST

List expressions are ordered collections of other expression data types. Lists can contain zero or more elements. Elements may be of different types, and lists may even contain other lists. An unspecified list expression may be completed by double-clicking the **<list expr>** and typing in a comma-separated list of other expressions enclosed in curly braces (**{** and **}**). As with object and slot expressions, it is highly recommended that you do not type list expressions directly into the **Editor.** The **Palette** contains a multitude of functions for creating and retrieving information from lists. Examples of valid lists expressions, before the expression language expands their elements, include:

```
{ 1561 [“ft”], 1573 [“ft”], 1589 [“ft”], 3800.05 [cfs] }
{ @“April 15, 1999” }
{ }
{ %“Powell”, 10 [“cfs”], True }
{ { %“Mead”, “Max Outflow”, TRUE }, { $“Inflow” }, False }
```

When an element of a list is read, its expression type must match the type expected by the reading function. If a function finds an object where a datetime is expected, the run is halted and an error is posted. The flexibility of lists is an advantage as long as the types of their contents are known, and they are used properly within blocks and functions. Because of their flexibility, however, the types of their elements cannot always be determined prior to a model run. For this reason, list type-checking is only done during execution, and errors in configuration are not caught until a run is in progress.

2. The RPL Palette



The RPL Palette is the dialog from which the operators and functions are selected to build a block or user function. Using the Palette avoids having to type most expressions directly into the Editor, thus limiting the potential for errors.

The Palette consist of three tabs, one for the buttons, one tab for **User-Defined Functions** and one tab for the **Predefined Functions**, described as follows:

Palette Buttons: On the **Palette Buttons** tab, each of the buttons represents an operation. The buttons use the following abbreviations:

- B: BOOLEAN
- D: DATETIME
- E: Expression; can be more than one type
- L: LIST
- N: NUMERIC
- Obj: OBJECT
- Slot: SLOT

These buttons represent operations which evaluate to one of the expression data types mentioned above. Buttons on the RPL Palette are enabled and disabled dynamically. When an expression is highlighted in the Editor, the Palette buttons that satisfy the expected data type are enabled. All the buttons and the operations to which they are associated are described below, starting **HERE (Section 2.1)**.

User Defined Functions: The User Defined Functions tab shows available functions in the RPL set or any opened Global Function sets. Functions are selected by double clicking on the function name to replace the selected expression with the function.

There is a toggle to show only functions which have the return type of the selected RPL expression:

☐ Show only functions with a return type matching the selected expression.

In addition, there is a toggle to retain the function arguments, when possible, when replacing an existing function.

☑ Retain arguments where possible when replacing functions.

When checked, the newly selected function will retain arguments from left to right as long as the types match. If an argument's type does not match, or if there are not enough existing arguments, the remaining arguments in the new function will remain as empty expressions.

Use the **Function ➥ Show Description** menu to show the selected function's Description as specified on the function's RPL editor.

Predefined Functions: This tab shows the list of predefined functions available in RiverWare. These functions are further described **HERE (RPLUserInterface.pdf, Section 2.6)**. Shown are sortable columns for the **Return Type**, **Name**, and the list of **Arguments** for each function.

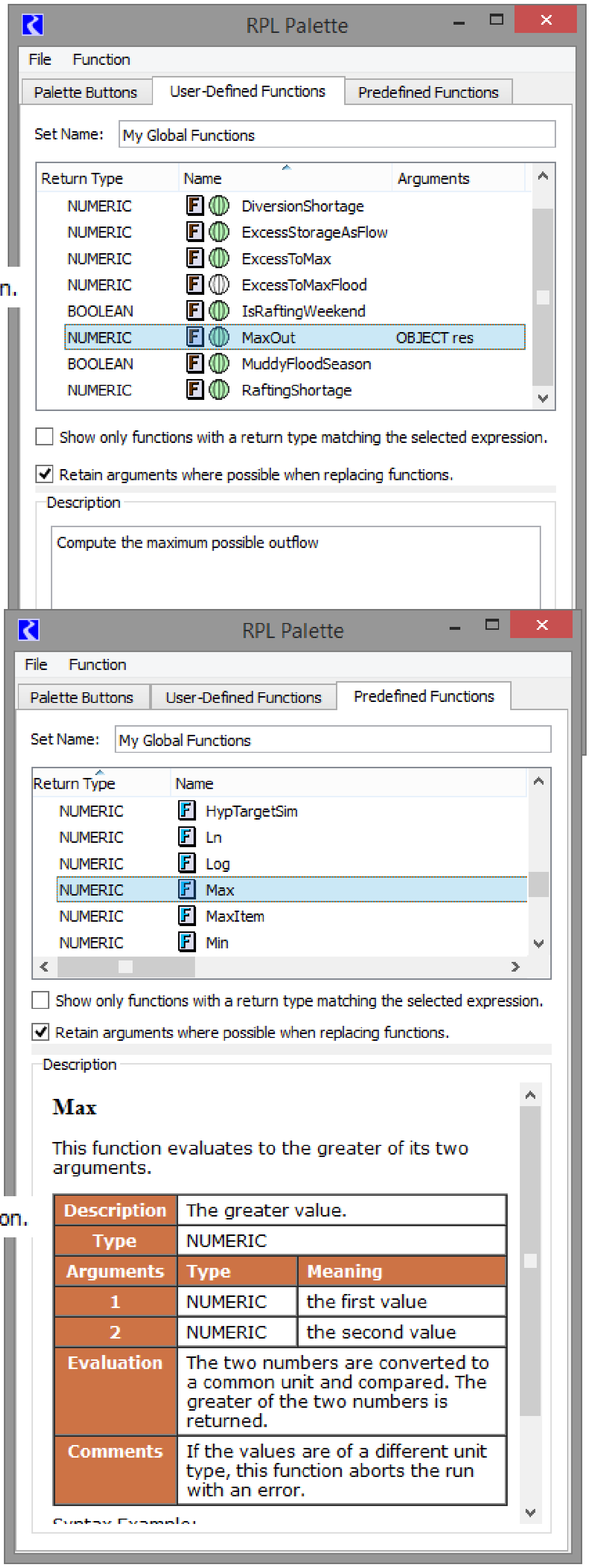
There is a toggle to show only functions which have the return type of the selected RPL expression:

☐ Show only functions with a return type matching the selected expression.

When the check box is off, all predefined functions are shown even if no expression is selected.

This tab also has a toggle to retain function arguments. as possible, when replacing an existing function. The behavior is the same as described above for the user-defined function tab.

☑ Retain arguments where possible when replacing functions.



Set Name: My Global Functions

Return Type	Name	Arguments
NUMERIC	DiversionShortage	
NUMERIC	ExcessStorageAsFlow	
NUMERIC	ExcessToMax	
NUMERIC	ExcessToMaxFlood	
BOOLEAN	IsRaftingWeekend	
NUMERIC	MaxOut	OBJECT res
BOOLEAN	MuddyFloodSeason	
NUMERIC	RaftingShortage	

☐ Show only functions with a return type matching the selected expression.

☑ Retain arguments where possible when replacing functions.

Description

Compute the maximum possible outflow

Set Name: My Global Functions

Return Type	Name
NUMERIC	HypTargetSim
NUMERIC	Ln
NUMERIC	Log
NUMERIC	Max
NUMERIC	MaxItem
NUMERIC	Min

☐ Show only functions with a return type matching the selected expression.

☑ Retain arguments where possible when replacing functions.

Description

Max

This function evaluates to the greater of its two arguments.

Description	The greater value.	
Type	NUMERIC	
Arguments	**Type**	**Meaning**
1	NUMERIC	the first value
2	NUMERIC	the second value
Evaluation	The two numbers are converted to a common unit and compared. The greater of the two numbers is returned.	
Comments	If the values are of a different unit type, this function aborts the run with an error.	

Use the **Function ➥ Show Description** menu to show the documentation for the selected function. This shows the same documentation as presented **HERE (RPLPredefinedFunctions.pdf, Section 1)**.

Note: Note, references are shown as blue but do not hyperlink elsewhere.

Included is a description, return type, list of the arguments and what they mean, description of evaluation, comments, syntax example and return example.

2.1 Mathematical Operation Buttons:

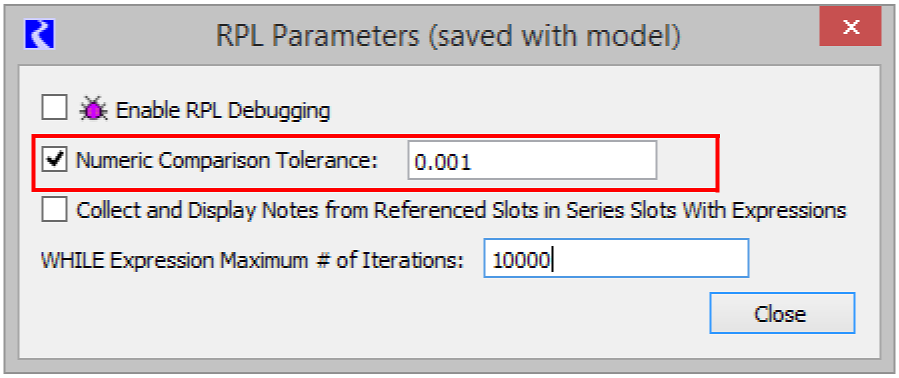
Button	Evaluates to:	Unspecified Form and Description
E + E	NUMERIC or DATETIME	**<expr> + <expr>** Addition of two expressions. The two arguments can be numeric or fully specified datetime expressions. If numeric expressions are used, they must be of the same unit type. For more information on the use of this operation with datetimes, click **HERE (Section B)**
E - E	NUMERIC or DATETIME	**<expr> - <expr>** Subtraction of two expressions. The two arguments can be numeric or fully specified datetime expressions. If numeric expressions are used, they must be of the same unit type. For more information on the use of this operation with datetimes, click **HERE (Section B)**
N * N	NUMERIC	**<numeric expr> x <numeric expr>** Multiplication of two numeric expressions.
N / N	NUMERIC	**<numeric expr> / <numeric expr>** Division of two numeric expressions.
N ^ N	NUMERIC	**<numeric expr> ^ <numeric expr>** Exponentiation of one numeric expression (the base) to a power of another numeric expression (the exponent). The exponent is truncated to an integer. The units of the base are raised to the power of the exponent.

2.2 Logical Operation Buttons:

Button	Evaluates to:	Unspecified Form and Description
IsNaN N	BOOLEAN	**IsNaN <numeric expr>** Logical validity check for numeric data types. The result is TRUE if the numeric expression evaluates to a NaN, i.e. Not a Number. The result is FALSE if the numeric expression has a value.
B AND B	BOOLEAN	**<boolean expr> AND <boolean expr>** Logical AND comparison of two expressions. The result is TRUE if both expressions are TRUE. The result is FALSE if one expression is TRUE and the other is FALSE. Note, if the first expression is FALSE, the entire AND is FALSE and the second expression is not evaluated.
B OR B	BOOLEAN	**<boolean expr> OR <boolean expr>** Logical OR comparison of two expressions. The result is TRUE if either expression is TRUE. Otherwise it is FALSE.
E > E	BOOLEAN	**<expr> > <expr>** Greater than comparison. The result is TRUE if the first expression is strictly greater than the second expression. Otherwise it is FALSE. The two arguments can be numeric expressions of the same unit type or datetime expressions (fully or partially specified). Numeric expressions are automatically converted to common units before comparison. A datetime expression is considered greater if it is later in time. See below for more information on tolerance during the comparison.
E >= E	BOOLEAN	**<expr> >= <expr>** Greater than or equal comparison. The result is TRUE if the first expression is greater than or equal to the second expression. Otherwise it is FALSE. The two arguments can be numeric expressions of the same unit type or datetime expressions (fully or partially specified). Numeric expressions are automatically converted to common units before comparison. A datetime expression is considered greater if it is later in time. See below for more information on tolerance during the comparison.
E < E	BOOLEAN	**<expr> < <expr>** Less than comparison. The result is TRUE if the first expression is strictly less than the second expression. Otherwise it is FALSE. The two arguments can be numeric expressions of the same unit type or datetime expressions (fully or partially specified) Numeric expressions are automatically converted to common units before comparison. A datetime expression is considered less than if it is earlier in time. See below for more information on tolerance during the comparison.

Button	Evaluates to:	Unspecified Form and Description
E <= E	BOOLEAN	**<expr> <= <expr>** Less than or equal to comparison. The result is TRUE if the first expression is less than or equal to the second expression. Otherwise it is FALSE. The two arguments can be numeric expressions of the same unit type or datetime expressions (fully or partially specified). Numeric expressions are automatically converted to common units before comparison. A datetime expression is considered less than if it is earlier in time. See below for more information on tolerance during the comparison.
E == E	BOOLEAN	**<expr> == <expr>** Equality comparison. The result is TRUE if the first expression is exactly equal to the second expression. Otherwise it is FALSE. The two arguments can be numeric expressions of the same unit type, fully or partially specified datetime expressions, object expressions, slot expressions, boolean expressions, string expressions, or list expressions. Numeric expressions are automatically converted to common units before comparison. A fully specified datetime expression can be compared to a partially specified datetime expression; in this case, only the largest time unit is compared. See below for more information on tolerance during the comparison.
E != E	BOOLEAN	**<expr> != <expr>** Inequality comparison. The result is TRUE if the first expression is not equal to the second expression. Otherwise it is FALSE. The two arguments can be numeric expressions of the same unit type, fully or partially specified datetime expressions, object expressions, slot expressions, boolean expressions, string expressions, and list expressions. Numeric expressions are automatically converted to common units before comparison. Datetime expressions are considered greater as they are later in time. A fully specified datetime expression can be compared to a partially specified datetime expression; in this case, only the largest time unit is compared. See below for more information on tolerance during the comparison.

2.2.1 Setting Tolerance for use in the logical comparison operators



☐ Enable RPL Debugging

☑ Numeric Comparison Tolerance: 0.001

☐ Collect and Display Notes from Referenced Slots in Series Slots With Expressions

WHILE Expression Maximum # of Iterations: 10000

In the above operators, a comparison is made between two values. If the values are numeric, there is the possibility that due to unit conversion or other internal numerical computation, RiverWare could try to compare two numbers that are, in all practical purposes the same, yet are different in the comparison. For example, a RPL expression says (0.0 “cfs” == Flow[]) and the Flow evaluates to $1X10^{-13}$ cfs due to the addition of two double precision values. In this example, the statement will return FALSE but probably should return TRUE. To avoid this situation, the user is able to define a tolerance value. From the main RiverWare workspace, select **Policy ➥ RPL Parameters...** to open the dialog shown.

Click on the Numeric Comparison Tolerance toggle. The box becomes active with a default value of zero. Enter a new value in the box to specify the tolerance. The tolerance represents an absolute value in standard units. Thus, in the example above, if the tolerance is 0.001, it represents that Abs(0.0 “cfs” - Flow[]) <= 0.001 cms for the comparison to return TRUE. In other words, if two values are within the comparison tolerance of one another, then they are considered equal for the purposes of RPL comparison operations.

The following table provides details about how the comparison tolerance is applied for each of the RPL comparison operators.

Operator	Application of Tolerance	RPL Example (Tolerance = 0.01)	Comparison With Tolerance Applied	Result
E == E	|A - B| <= Tolerance	5.00 cms == 5.00 cms	|5.00 cms - 5.00 cms| <= 0.01 cms	**TRUE**
		5.00 cms == 4.99 cms	|5.00 cms - 4.99 cms| <= 0.01 cms	**TRUE**
		4.98 cms == 5.00 cms	|4.98 cms - 5.00 cms| <= 0.01 cms	**FALSE**
E != E	|A - B| > Tolerance	5.00 cms != 5.00 cms	|5.00 cms - 5.00 cms| > 0.01 cms	**FALSE**
		5.00 cms != 4.99 cms	|5.00 cms - 4.99 cms| > 0.01 cms	**FALSE**
		4.98 cms != 5.00 cms	|4.98 cms - 5.00 cms| > 0.01 cms	**TRUE**
E > E	A > B + Tolerance	5.02 cms > 5.00 cms	5.02 cms > 5.01 cms	**TRUE**
		5.01 cms > 5.00 cms	5.01 cms > 5.01 cms	**FALSE**
		5.00 cms > 5.00 cms	5.00 cms > 5.01 cms	**FALSE**
E >= E	A + Tolerance >= B	5.01 cms >= 5.00 cms	5.02 cms >= 5.00 cms	**TRUE**
		4.99 cms >= 5.00 cms	5.00 cms >= 5.00 cms	**TRUE**
		4.98 cms >= 5.00 cms	4.99 cms >= 5.00 cms	**FALSE**
E < E	A + Tolerance < B	4.98 cms < 5.00 cms	4.99 cms < 5.00 cms	**TRUE**
		4.99 cms < 5.00 cms	5.00 cms < 5.00 cms	**FALSE**
		5.00 cms < 5.00 cms	5.01 cms < 5.00 cms	**FALSE**
E <= E	A <= B + Tolerance	4.99 cms <= 5.00 cms	4.99 cms <= 5.01 cms	**TRUE**
		5.00 cms < = 4.99 cms	5.00 cms <= 5.00 cms	**TRUE**
		5.00 cms <= 4.98 cms	5.00 cms <= 4.99 cms	**FALSE**

The RPL Numeric Comparison Tolerance parameter is saved with the model file and is applied to all RPL sets used by the model including Rulesets, User Defined Accounting Method sets, Optimization Goalsets, Initialization Rules, Iterative MRM sets, Expression Slot sets and Global Functions sets.

2.3 Object and Slot Lookup and Assignment Buttons:

Button	Evaluates to:	Unspecified Form and Description
Slot [E]	NUMERIC	**<expr> [<expr>]** Series slot value at a particular timestep or slot value at a particular row. The first unspecified <expr> must be completed by an expression which evaluates to a specific slot on an object. The second <expr> must be a fully specified datetime which lies on an increment of the model run timestep or a row on the slot.
Slot [E, E]	NUMERIC	**<expr> [<expr>, <expr>]** Table slot value in a particular row and column. The first unspecified <expr> must be completed by an expression which evaluates to a specific table slot on an object. The two comma-separated <expr> are the row and column of the table slot, respectively. The row and column may each be specified as a **zero**-based numeric value with units of [NONE] or a string expression which matches the column or row label. OR Agg series slot or periodic slot value for a particular date and a particular column. The first unspecified <expr> must be completed by an expression that evaluates to a specific slot on an object. The two comma-separated <expr> are the date and column of the slot, respectively. The column may be specified as a zero-based numeric value with units of [NONE] or a string expression which matches the column label. Click **HERE (Slots.pdf, Section 4.12.3)** for more info.
Slot [ ]	NUMERIC	**<expr> [ ]** Series slot value at the current timestep or scalar slot value. The unspecified <expr> must be completed by an expression which evaluates to a specific slot on an object.

Button	Evaluates to:	Unspecified Form and Description
NaNToZero N	NUMERIC	**NaNToZero <expr>** This operator has a single numeric sub expression; if that sub expression is a lookup of an invalid value (NaN), then NaNToZero evaluates to 0.0 in the units of the lookup slot, otherwise it simply returns the value unchanged. This operator provides a simple and efficient way to treat a missing slot value as zero, a common requirement of water accounting policy. To illustrate, the following expression **NaNToZero "Res A.Outflow" [ ]** is equivalent to

```
IF ( IsNaN "Res A.Outflow" [ ] )
   0.0 [<Res A.Outflow’s units>]
ELSE
   "Res A.Outflow" [ ]
END IF
```

Note that, unlike the IsNaN operator, this operator is only useful when the sub-expression is an object/slot lookup expression; if any other type of numeric sub expression encounters a NaN during evaluation, then evaluation of the entire containing expression will halt as usual. For example, if we assume that "Res A.Outflow" has units of cms but has no valid values, then the expression

NaNToZero ("Res A.Outflow" []) + 0.0 [cms]

would evaluate to 0.0 [cms], but the expression

NaNToZero ("Res A.Outflow" [] + 0.0 [cms])

would not evaluate successfully; it would terminate when the invalid value on Res A was encountered.

Button	Evaluates to:	Unspecified Form and Description
Obj . Slot	SLOT	**<object expr> . <string expr>** Object and slot expression. This expression can be used to specify the object and slot required by the three slot lookup/assignment expressions above. The object expression must be an object in the model and the string expression must be the name of a slot, exactly as it appears on the object.
Obj ^ Slot	SLOT	**<object expr> ^ <string expr>** Object and accounting slot expression. This expression can be used to specify the object and slot required by the three slot lookup/assignment expressions above. The object expression must be an object in the model and the string expression must be the full name of an accounting slot, in the form: *account name.slot name*.

Button	Evaluates to:	Unspecified Form and Description
Object Selector	OBJECT	**Invokes the object selector dialog** Object selector. This button invokes the object selector found elsewhere in RiverWare to choose a single object in the model.
Slot Selector	SLOT	**Invokes the slot selector dialog** Object and slot or Object and accounting slot selector. This button invokes the slot selector found elsewhere in RiverWare to choose a single object and slot in the model.

2.4 Unary Operation Buttons:

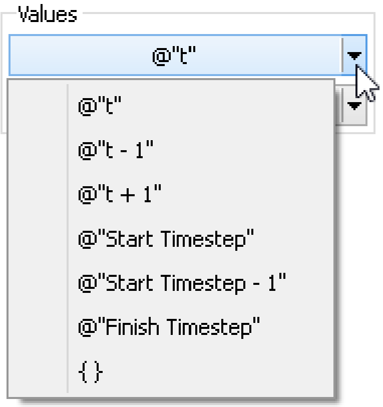
Button	Evaluates to:	Unspecified Form and Description
- N	NUMERIC	**- <numeric expr>** Reverse the sign of a numeric expression. Magnitude and units are maintained, but a positive value becomes negative and a negative value becomes positive.
NOT B	BOOLEAN	**NOT <boolean expr>** Reverse the value of a boolean expression. TRUE becomes FALSE and FALSE becomes TRUE.

2.5 Values

The Values section is used to access common values and set flags on slots.

2.5.1 Buttons for Common Values

The common values buttons are used to access values that have traditionally been typed by the user. These include common DATETIME values and list expressions.

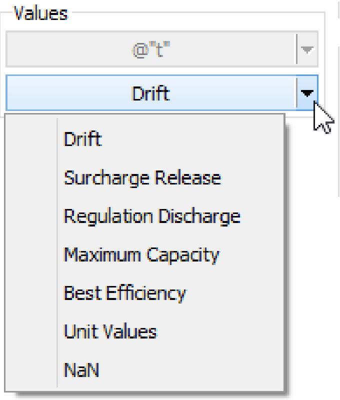


- @“t” - Current Timestep
- @“t-1” - Previous Timestep
- @“t+1” - Next Timestep
- @“Start Timestep” - First timestep in the run
- @“Start Timestep - 1” - Initial timestep
- @“Finish Timestep” - Final Timestep in the run period
- { } - Empty list expression

A similar list of common values can be found by right clicking on an expression and choosing the **Common Values** menu.

2.5.2 Buttons for Setting Flags on Slots

The following RPL operations can only be used to directly set a slot value. They cannot be passed to functions or referenced in block or function logic. They cannot replace any unspecified NUMERIC expression; they can only be used to set a slot.



Menu Button	Description	Notes
Drift	Used to set the DRIFT (D) flag on the Regulated Spill or Bypass slots on reservoir objects	The behavior of these operators is identical to the user setting the particular flag through the user interface. The only exception in this case is that the slot can be overwritten by a higher priority rule. When these operators are used, the slot will have both the R flag and the flag associated with the palette button selected. When the object dispatches, the slot will receive the appropriate value. The presence of the these flags will cause the slot to be considered as a known value (for choosing a dispatch method) even though it will not actually have a value until after the object dispatches.
Maximum Capacity	Used to set the MAX CAPACITY (M) flag on the Outflow or Energy slots on reservoir objects.	
Surcharge Release	Used to set the SURCHARGE RELEASE (S) flag on the Surcharge Release slot on reservoir objects.	
Best Efficiency	Used to set the BEST EFFICIENCY (B) flag on the Energy slot on reservoir objects.	
Regulation Discharge	Used to set the REGULATION_DISCHARGE (G) flag on the Reg Discharge Calculation slot on Control Point objects.	
Unit Values	Used to set the UNIT_VALUES (U) flag on Turbine Release or Energy slots on power reservoirs. This flag is used (with the Unit Power Table method) to specify that either Unit Turbine Release or Unit Energy will be specified.	

2.6 Conditional and Iterative Operations Buttons:

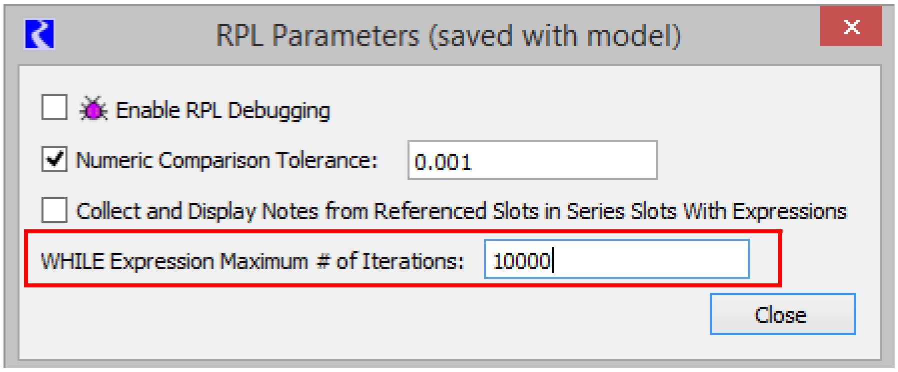
Button	Evaluates to:	Unspecified Form and Description

IF ...	any	**IF (<boolean expr>) THEN**
		<expr>
		END IF
		Adds a conditional expression with NO else clause. If the boolean expression evaluates to TRUE, the expression on the second line is evaluated. Otherwise it is not, and the entire conditional statement does not evaluate to a value.

IF ... ELSE ...	any	**IF (<boolean expr>) THEN**
		<expr>
		ELSE
		<expr>
		END IF
		Adds a conditional expression including an else clause. If the boolean expression evaluates to TRUE, the expression on the second line is evaluated. If the boolean expression evaluates to FALSE, the expression on the fourth line is evaluated. The entire conditional statement evaluates to one of the two expressions.

ELSE ...	any	**IF (<boolean expr>) THEN** **<expr>** **ELSE** **<expr>** **END IF** Adds an Else branch to a conditional expression. It is generated by highlighting the either the boolean condition <boolean expr> or consequence expression <expr> of an IF expression or ELSE IF expression and clicking the ELSE ... button. If the boolean expression evaluates to FALSE, the expression on the fourth line is evaluated. The entire conditional evaluates to one of the two expressions.
ELSE IF ...	any	**IF (<boolean expr>) THEN** **<expr>** **ELSE IF (<boolean expr>) THEN** **<expr>** **ELSE IF (<boolean expr>) THEN** **<expr>** **END IF** Adds an ELSE IF branch to a conditional expression. It is generated by highlighting the boolean condition <boolean expr> or consequence <expr> of an IF or ELSE IF and clicking the ELSE IF... button. The ELSE IF branch is added immediately after the branch containing the selection. In the example above, if the first boolean expression evaluates to FALSE, the boolean expression on the third line is evaluated. If that boolean is true, the expression on the fourth line is evaluated. You may add as many ELSE IF... branches as needed.

Button	Evaluates to:	Unspecified Form and Description
FOR ...	any	**FOR (NUMERIC index IN <list expr>) WITH NUMERIC result = <numeric expr> DO** **result = <numeric expr>** **END FOR** Structure for looping over the items in a list. The looping variable, index, is set to the value of the next item in the list expression each time through the loop. The value variable, result, is initialized on the first line of the structure. Each time through the loop, the expression on the second line is evaluated and its value is set on the result variable. The previous value of the result variable may be used inside the expression on the second line. The items in the list and the result may be of any expression type even though the default structure uses the numeric type for the looping variable and the result. To change the type of the looping variable, index, and/or the value variable, result, double-click on the "NUMERIC" element and enter the new expression type. The names of the looping and value variables may also be changed. There is also a FOR statement used to loop over a list and execute multiple statements. It is described **HERE (RPLUserInterface.pdf, Section 2.3)**.
WITH ...	any	**WITH NUMERIC var = <numeric expr> DO** **<expr>** **END WITH** Structure for defining a variable to be used multiple times within an expression. The variable, var, is set on the first line of the structure. This variable can be used as many times as desired within the expression(s) between the DO and END WITH. Because the variable is not recalculated each time it is used, this structure can make a block more efficient. The variable may be of any expression type even though the default structure uses the numeric type. To change the type of the variable, var, double-click on the "NUMERIC" element and enter the new expression type. The name of the variable may also be changed. There is also a WITH statement used to set a temporary variable outside of multiple statements. It is described **HERE (RPLUserInterface.pdf, Section 2.3)**.

Button	Evaluates to:	Unspecified Form and Description
WHILE ...	any	**WHILE (<boolean expr>) WITH NUMERIC result = <numeric expr> DO** **result = <numeric expr>** **END WHILE** Structure for looping as long as a condition is TRUE. The conditional boolean expression is evaluated prior to each loop. If the boolean expression is TRUE, the numeric expression on the second line is evaluated. If the boolean expression is FALSE, the structure stops looping and returns the value of the value variable. The value variable, result, is initialized on the first line of the structure. Each time through the loop, the expression on the second line is evaluated and its value is set on the result variable. The previous value of the result variable may be used inside the expression on the second line. The result may be of any expression type even though the default structure uses the numeric type. To change the type of the value variable, result, double-click on the “NUMERIC” element and enter the new expression type. The name of the value variable may also be changed. When a WHILE expression is executed, it counts the number of times the body is evaluated, and fails with a message if the iteration count ever exceeds the value of the RPL parameter. The default max iterations is 10,000, but may be modified as needed in the RPL Parameters dialog. (From the main workspace, select **Policy ➥ RPL Parameters...** to open the parameters).  RPL Parameters (saved with model) ☐ Enable RPL Debugging ☑ Numeric Comparison Tolerance: 0.001 ☐ Collect and Display Notes from Referenced Slots in Series Slots With Expressions WHILE Expression Maximum # of Iterations: 10000

Button	Evaluates to:	Unspecified Form and Description
SUM ...	NUMERIC	**FOR (NUMERIC index IN <list expr>) SUM** **<numeric expr>** **END FOR** Specialized FOR loop structure for looping over the items in a list and summing a variable. For each item in the list, the looping variable, index, is set to the value of the next item in the list expression. Next, the expression on the second line is evaluated and its value is added to the result variable. The items in the list may be of any expression type even though the default structure uses the numeric type for the looping variable and the result. To change the type of the looping variable, index, double-click on the “NUMERIC” element and enter the new expression type. The names of the looping variables may also be changed.
AVE ...	NUMERIC	**FOR (NUMERIC index IN <list expr>) AVE** **<numeric expr>** **END FOR** Specialized FOR loop structure for looping over the items in a list and averaging their values. For each item in the list, the looping variable, index, is set to the value of the next item in the list expression. Next, the expression on the second line is evaluated and its value is included in the total resulting average. The items in the list may be of any expression type even though the default structure uses the numeric type for the looping variable and the result. To change the type of the looping variable, index, double-click on the “NUMERIC” element and enter the new expression type. The names of the looping variable may also be changed.

2.7 List Operation Buttons:

Button	Evaluates to:	Unspecified Form and Description
{ E }	LIST	**{ <expr> }** This button creates a list with a single item of an unspecified type.
E , E	any	**<expr>, <expr>** This button adds a new item of an unspecified type to an existing list. To use it, highlight one item and click the button. The new item is added immediately after the highlighted item in the list.
L - L	LIST	**<list expr> - <list expr>** The set difference expression takes two lists and returns a single list that contains the items that are in the left hand list but not in the right hand list. Duplicate items are removed.

Button	Evaluates to:	Unspecified Form and Description
L ^ L	LIST	**<list expr> ^ <list expr>** The set symmetric difference expression takes two lists and returns a single list that contains the items that are not in both lists. This is equivalent to the expression: (list1 U list2) - (list1 intersect list2). This is also equivalent to the expression: (list1 - list2) U (list2 - list1). Duplicate items are removed.
L UNION L	LIST	**<list expr> UNION <list expr>** The set union expression takes two lists and returns a single list that contains the items that are in either list or both lists. Duplicate items are removed.
INTERSECTION	LIST	**<list expr> INTERSECTION <list expr>** The set intersection expression takes two lists and returns a single list that contains the items that are in both lists. Duplicate items are removed.
L<N>	any	**<list expr>< <numeric expr> >** Evaluates to the expression located at the given numeric index in the given list. The index numeric expression is zero-based and has units of [NONE]. If the item at the given index in the list is not the expected type when the expression is evaluated or if there is no item at the given index in the list, the run aborts with an error.
INSERT	LIST	**INSERT <expr> INTO <list expr>** Insertion of a new item of unspecified type into the first location of an existing list. If the list is initially empty, the new list contains only one item. If the list initially contains items, all items are shifted down to allow the new item to be inserted at the front of the list.
LENGTH L	NUMERIC	**LENGTH <list expr>** Determines the number of items in the list expression. The numeric value has units of [NONE].
APPEND ...	LIST	**APPEND <expr> ONTO <list expr>** Append a new item of unspecified type onto the end of an existing list. If the list is initially empty, the new list contains only one item. If the list initially contains items, the new item is appended as the last item of the list.
E IN L	BOOLEAN	**<expr> IN <list expr>** Tests for existence of the given expression as a member of the list expression. If the given expression is one of the items in the list, this function evaluates to TRUE. If the expression does not match any of the items of the list, the function is FALSE.

Button	Evaluates to:	Unspecified Form and Description
D TO D	LIST	**<datetime expr> TO <datetime expr>** Creates a list of datetimes beginning with the first datetime expression and ending on or before the second datetime expression. The datetime arguments must be fully specified. The interval between datetime items of the list is equal to the run timestep (or the slot's timestep for an expression slot). If the second datetime argument does not correspond to a model run timestep, the last item of the list is the last model run timestep before the argument.
REMOVE ...	LIST	**REMOVE ITEM @INDEX <numeric expr> FROM <list expr>** Removes the item at a given index from an existing list. All of the items which followed the removed item in the list are shifted up. If there is no item at the given index when the expression is evaluated, the run aborts with an error.
SUB ...	LIST	**SUB <expr> FOR ITEM @INDEX <numeric expr> FROM <list expr>** Substitutes the item at a given index in an existing list with the given expression. If there is no item at the given index when the expression is evaluated, the run aborts with an error.
FIND ...	NUMERIC	**FIND <expr> WITHIN <list expr>** This is used to find the index of a given item in a list. If the item is not contained in the list, -1 (negative one) is returned.
MAP LIST ...	LIST	**MAPLIST (NUMERIC index IN <list expr>) DO** **<expr>** **END MAPLIST** This expression is used to take a list and perform some action to each element in the list thereby resulting in a new, modified list. This expression was developed primarily for performance reasons. Prior to the existence of this function, the user would use a FOR loop to rotate through the elements of a list, modify them, and create a new list with the modified elements. With the MAP LIST expression, this is much more efficient and more readable.

2.8 Miscellaneous Buttons:

Button	Evaluates to:	Unspecified Form and Description
E CONCAT E	LIST or STRING	**<expr> CONCAT <expr>** Concatenates two expressions into one. If the two expressions are of type LIST, the resulting expression is a single list containing all of the items of the two original lists. The order of the items is preserved, with the first list's items before the second list's items. If the two expressions are of any other type, the resulting expression is a string made up by concatenating the STRINGIFY'd second expression onto the end of the STRINGIFY'd first expression. See the description of STRINGIFY below. Note, it is not necessary to explicitly STRINGIFY an expression before using it in a CONCAT expression. If the expression is not already a STRING, it will be automatically converted. Thus, **1000 CONCAT "cfs"** is equivalent to **(STRINGIFY 1000) CONCAT "cfs"** but the first expression is cleaner and more efficient
(E)	any	**(<expr>)** Adds parenthesis around any expression. Parenthesis can make complicated expressions more readable. Parenthesis may also be required to remove ambiguity in the order of expression evaluation.

STRINGIFY E	STRING	**STRINGIFY <expr>**
		Converts any expression into a string. The STRING representations of other expression types are:
		NUMERIC => "*number* [*units*]" or "*number*" if units are {NONE}
		DATETIME => "*hours*:*minutes month day*, *year*" or less, depending on model run timestep
		BOOLEAN => "TRUE" or "FALSE"
		OBJECT => "*object name*"
		SLOT => "*object name*.*slot name*"
		LIST => "{ *item*, *item*, *item* ... }"

STOP_RUN E	Aborts the Run	**STOP_RUN <expr>** This operator takes any expression type as an argument. When it is evaluated, it aborts the run with an error message which contains the argument as part of the message. If executed from within an iterative MRM rule, it aborts the MRM run.

Button	Evaluates to:	Unspecified Form and Description
Add Comment	comment	**<comment>** Inserts a user specified in-line comment above the selected RPL expression or RPL statement. A separate dialog is opened that allows the user to type in a comment. In the RPL editor, the comment is displayed with # characters on the left, lines wrapped as they were in the comment editor dialog, and text in red (or user-specified comment color **HERE (Section 6.2.2)**). Double clicking the comment reopens the edit dialog. For a given RPL dialog, the inline comments can be hidden or shown using the **View -> Show Comments** menu or by checking the **Show: Comments** toggle at the bottom of the dialog.
Delete Comment	Deletes comment	Deletes the selected comment. Comments can also be deleted by selecting the comment, then using the **Edit -> Delete** menu or using the Delete key.

2.9 Units in RPL

In Simulation, all computations are done in internal units, that is cms, m, m3, etc. In RPL, computations are performed in user units. Further:

- You can use any units, even different than slot units.
- Unit types must always be consistent
- When units are not consistent one value is converted to the other (i.e. 20 cfs + 40 cms)

The unit syntax is “units” with the quotes. But quotes are not necessary unless a “ – ” or “ / ” is used, like “acre-ft”.

2.9.1 Unit operators

Following are the operators available when specifying RPL units:

- “-” multiplication operator (ex. “acre-ft”)
- “/” division operator (ex. “m/s”)
- “^” raised to the power of". (ex. “m^2”)
- giga = 1e9 (ex. “giga-cfs”)
- mega = 1e6
- kilo = 1000
- pico = 1e-12

2.9.2 Slot Value Units

When a RPL expression accesses a value on a slot, there are several options for how the slot value is represented internally. You have some control by specifying a command line argument, click **HERE**

(BatchMode.pdf, Section 2) for more information on command line arguments. Following is a description of the various ways to represent the slot values:

Type	Description	--rplslotvalunits argument
mixed	The slot's user scale and standard units. This was the original representation, the only possibility until now, and is the current **default**.	mixed
standard	A scale of 1.0 and the slot's standard units	std
user	The slot's user scale and user units, unless the units are time-varying (e.g., acre-feet/month), in which case a scale of 1.0 and standard unit is used.	user

The RPL slot value representation scheme has two primary impacts:

- Numerical accuracy of the computation - the representation scheme affects the magnitude of the RPL values and so impacts numerical accuracy. Generally speaking the greatest accuracy is expected when using values of moderate magnitude. For many models, this consideration favors the "user" representation.
- Diagnostics - Sometimes diagnostics present values in a form related to their internal representation. In most cases, this consideration favors the "user" representation.

A command line argument can be used to control the RPL slot unit representation. the argument is "--rplslotvalunits" and it requires an argument as listed in the above table. Using the default “mixed” units will reproduce results in your model but you may consider using the “user” option for diagnostic and/or accuracy purposes.