RiverWare

Technical Documentation Version 7.0

RPL Predefined Functions

CADSWES

Center for Advanced Decision Support for Water and Environmental Systems

RPL Predefined Functions Table of Contents

RPL Predefined Functions

Introduction: This section describes the predefined RiverWare functions which are available for use in any RiverWare Policy Language (RPL) set.

Predefined functions perform a wide range of calculations common to water management policy and its expression in a rule context. Some predefined functions provide rules with access to the same algorithms used in RiverWare simulation for mass balance calculations, flow/volume conversions, and table lookups. Other predefined functions are available for common mathematical operations, date/time manipulations, topographical evaluations, and some specialized river basin management calculations.

Predefined functions are used in RPL sets in the same way as user defined custom functions. They are selected from the palette, and inserted into a rule, internal function, or other expression. Each predefined function is an expression which evaluates to one of the 7 rules data types:

1. NUMERIC
2. BOOLEAN
3. DATETIME
4. OBJECT
5. SLOT
6. STRING
7. LIST

Predefined functions may or may not have arguments. The computational algorithms and arguments to predefined functions may not be modified.

1. Abs

This function evaluates to the absolute value of its single numeric argument.

Description	Absolute value operator	
Type	NUMERIC	
Arguments		
1	NUMERIC	value to evaluate
Evaluation	Determines the absolute value of the numeric argument	

Comments	none

Syntax Example:

```
Abs(-11 "cfs") returns 11 "cfs"
```

Use Examples:

```
IF(Abs(res.Inflow[] - res.Inflow[@"Next Timestep"]) < 1 "cms") THEN TRUE
```

2. AccountAttributes

Description	Given a string representing an account's full name (object^account), returns a list containing the account's attributes, i.e., the account's water type, water owner, and account type.	
Type	LIST {STRING, STRING, STRING}	
Arguments		
1	STRING	The name of the account.
Evaluation		
Comments		

Syntax Example:

```
AccountAttributes("ResA^GoodWater")
```

Return Example:

```
{"Intra-basin Transfer", "Big City", "StorageAccount"}
```

3. AccountNameFromPriorityDate

This function evaluates to the name of the account having the specified priority date.

Description	This function returns the name of the account having the specified priority date.	
Type	STRING	
Arguments		
1	DATETIME	The priority date.

Evaluation	The Accounts in the system are examined; the Account having the indicated priority date is returned.
Comments	Priority dates are a property of Accounts. It's an error if no account has the specified priority date.

Syntax Example:

```
AccountNameFromPriorityDate (@"12:00:00 August 12, 2004")
```

Return Example:

```
"Account1"
```

4. AccountNamesByAccountType

This function evaluates to the list of names of Accounts on the specified Object having the indicated Account type.

Description	This function returns a list of names of Accounts on a specified Object having the indicated Account type, sorted in ascending Account priority date order. Accounts which don't have a priority date are at the end of the list, sorted in ascending name order.	
Type	LIST {STRING}	
Arguments		
1	OBJECT	The Object.
2	STRING	Account type name (currently, one of "Diversion", "Storage", or "PassThrough") or "ALL".
Evaluation	The set of Accounts on the Object are examined. The names of the Accounts having the specified account type are added to the returned list. If the Account type argument is "ALL," then that attribute is ignored. The returned list will contain the names of ALL Accounts on the Object. The list is sorted as described above.	
Comments	Priority dates are properties of Accounts.	

Syntax Example:

```
AccountNamesByAccountType (%"Heron Reservior", "Storage")
```

Return Example:

```
{"Account1", "Account2"}
```

5. AccountNamesByWaterOwner

This function evaluates to the list of names of Accounts on the specified Object having the indicated WaterOwner.

Description	This function returns a list of names of Accounts on a specified Object having the indicated WaterOwner, sorted in ascending Account priority date order. Accounts which don't have a priority date are at the end of the list, sorted in ascending name order.	
Type	LIST {STRING}	
Arguments		
1	OBJECT	The Object.
2	STRING	WaterOwner name or "NONE" or "ALL"
Evaluation	The set of Accounts on the Object are examined. The names of the Accounts having the specified WaterOwner are added to the returned list. If the WaterOwner argument is "NONE," then only Accounts having the default (unassigned) WaterOwner are included in the returned list. If the WaterOwner argument is "ALL," then that attribute is ignored. The returned list will contain the names of ALL Accounts on the Object. The list is sorted as described above.	
Comments	WaterOwners and priority dates are properties of Accounts.	

Syntax Example:

```
AccountNamesByWaterOwner (%"Heron Reservior", "Contractor2")
```

Return Example:

```
{"Account1","Account2"}
```

6. AccountNamesByWaterType

This function evaluates to the list of names of Accounts on the specified Object having the indicated WaterType.

Description	This function returns a list of names of Accounts on a specified Object having the indicated WaterType, sorted in ascending Account priority date order. Accounts which don't have a priority date are at the end of the list, sorted in ascending name order.

Type	LIST {STRING}	
Arguments		
1	OBJECT	The Object.
2	STRING	WaterType name or "NONE" or "ALL"
Evaluation	The set of Accounts on the Object are examined. The names of the Accounts having the specified WaterType are added to the returned list. If the WaterType argument is "NONE," then only Accounts having the default (unassigned) WaterType are included in the returned list. If the WaterType argument is "ALL," then that attribute is ignored. The returned list will contain the names of ALL Accounts on the Object. The list is sorted as described above.	
Comments	WaterTypes and priority dates are properties of Accounts.	

Syntax Example:

```
AccountNamesByWaterType (%"Heron Reservior", "SanJuan")
```

Return Example:

```
{"Account3","Account4"}
```

7. AccountNamesFromObjReleaseDestination and AccountNamesFromObjReleaseDestinationIntra

This function evaluates to the list of names of Accounts on the specified Object having outflow Supplies of the given ReleaseType and Destination.

Description	This function returns a list of names of Accounts on a specified Object where the attributes of the outflow Supplies of the Accounts match the given ReleaseType and Destination. The list is sorted in ascending Account priority date order; Accounts which don't have a priority date are at the end of the list, sorted in ascending name order.	
Type	LIST {STRING}	
Arguments		
1	OBJECT	The Object.
2	STRING	ReleaseType name or "NONE" or "ALL"
3	STRING	Destination name or "NONE" or "ALL"

Evaluation	The set of Accounts on the Object are examined. The outflow Supplies on those Accounts are then examined. The names of the Accounts which have Supplies which (1) link a different downstream Object, and (2) have the indicated ReleaseType, and (3) have the indicated Destination are added to the returned list. If the ReleaseType argument or the Destination argument is "NONE," then only Supplies having the default (unassigned) attribute of that type are considered. If the ReleaseType argument or the Destination argument is "ALL," then that Supply attribute is ignored. The list is sorted as described above. The “Intra” version of the function will only look at transfer supplies that are within the object.
Comments	ReleaseTypes and Destinations are properties of Supplies; priority dates are properties of Accounts.

Syntax Example:

```
AccountNamesFromObjReleaseDestination (%"Heron Reservior",
                                        "Account Fill", "Albiquiu")
```

Return Example:

```
{"DownstreamAcct1", "NaturalFlowAccount"}
```

8. AccountPriorityDate

This function evaluates to the priority date of the Account, on the specified Object, having the specified name.

Description	This function returns the priority date of the Account, on the specified object, having the specified name.	
Type	DATETIME	
Arguments		
1	OBJECT	The Object
2	STRING	The Account name

Evaluation	The Object's accounts are examined. If an Account exists with the specified name its priority date is returned.
Comments	Priority dates are a property of Accounts. It's an error if either the Object doesn't have an Account with the specified name or the Account doesn't have a priority date.

Syntax Example:

```
AccountPriorityDate (%"Reservoir1", "NaturalFlowAcct")
```

Return Example:

```
@"February 23, 1902"
```

9. AnnualEventCount

This function analyzes a slot's value over some number of years, counting the occurrence of certain "events".

Description	Return the number of events which occurred on a slot in a given period.	
Type	NUMERIC	
Arguments		
1	SLOT	a slot
2	DATETIME	analysis period start date
3	DATETIME	analysis period end date
4	DATETIME	event period start date
5	DATETIME	event period end date
6	NUMERIC	value threshold
7	BOOLEAN	value threshold is upper bound
8	NUMERIC	event threshold
9	BOOLEAN	event threshold is upper bound
Evaluation	See the on-line documentation for AnnualEventStats, which performs identical computation, but returns more information. This function returns only the number of events which occurred in the analysis period.	
Comments		

Syntax Example:

```
AnnualEventCount($ "Lottawatta Reservoir.Outflow", @"24:00:00 February 28,
1994", @"24:00:00 January 31, 2005", @"24:00:00 May 31", @"24:00:00 August 31",
100.0, TRUE, 2.0, TRUE)
```

Return Example:

```
102.0000
```

10. AnnualEventLastOccurrence

This function analyzes a slot's value over some number of years, noting the last occurrence of a certain type of event.

Description	Return the number of event periods which occurred after the last event on a slot.	
Type	NUMERIC	
Arguments		
1	SLOT	a slot
2	DATETIME	analysis period start date
3	DATETIME	analysis period end date
4	DATETIME	event period start date
5	DATETIME	event period end date
6	NUMERIC	value threshold
7	BOOLEAN	value threshold is upper bound
8	NUMERIC	event threshold
9	BOOLEAN	event threshold is upper bound
Evaluation	See the on-line documentation for AnnualEventStats, which performs identical computation, but returns more information. This function returns only the number of event periods which occurred after the last event. If no events occurred, then this is the number of event periods.	
Comments		

Syntax Example:

```
AnnualEventLastOccurrence($ "Lottawatta Reservoir.Outflow", @"24:00:00 February
28, 1994", @"24:00:00 January 31, 2005", @"24:00:00 May 31", @"24:00:00 August
31", 100.0, TRUE, 2.0, TRUE)
```

Return Example:

```
2.00000
```

11. AnnualEventStats

This function analyzes a slot's value over some number of years, noting the occurrence of certain "events".

Description	Collects and returns statistics on annual events occurring on a slot.	
Type	LIST	
Arguments		
1	SLOT	a slot
2	DATETIME	analysis period start date
3	DATETIME	analysis period end date
4	DATETIME	event period start date
5	DATETIME	event period end date
6	NUMERIC	value threshold
7	BOOLEAN	value threshold is upper bound
8	NUMERIC	event threshold
9	BOOLEAN	event threshold is upper bound

Evaluation	The analysis period start and end dates define the period during which the analysis will be performed. Within the analysis period, only the timesteps which occur on or between the day and month of the event period start and end dates are considered. Each of these periods within the analysis period is called an event period. At each event period, an event can either occur or not. An event is defined by the value threshold and comparison type and the subevent count threshold and comparison type. At each timestep within an event analysis period, the slot's value is compared to the threshold value. If the value threshold is an upper bound and the slot's value is greater than the value threshold, then a subevent is said to have occurred at that timestep; similarly, if the value comparison is a lower bound and the slot's value is less than the value threshold, then a subevent is said to have occurred. After the subevents within an event analysis period have been noted, then they are counted up and compared to the subevent count threshold. If the subevent count threshold is an upper bound and the number of subevents which occurred in an event analysis period is greater than the subevent count threshold, then an event is said to have occurred, and similarly, if the subevent count comparison is a lower bound and the number of subevents which occurred in an event analysis period is less than the subevent count threshold, then an event is said to have occurred. The return list contains the following items (listed in order): • The total number of event periods. • The number of events which occurred. The number of event periods which occurred after the last event. If no events occurred, then this is the number of event periods.

Comments	As defined above, the first and last event periods might be of shorter duration than the other event periods. For example, if the analysis period is July 1, 1980 through June 30, 1989 and the event period is May 1 through September 30, then the first event period will be July 1, 1980 through September 30, 1980; subsequent event periods will be from May 1 through September 30, until the last event period, which will be from May 1, 1989, through June 30, 1989. If the event period contains the end of February, then event periods during leap years will also have a different duration. It is an error for the start or end date of the event period to be February 29, which does not exist in each year. Event periods can span year boundaries. For example, if the event period begin is December and the event period end is January, then each event period will be from December of one year to January of the next. One can leave the year field of the event period start or end date unspecified, if one is using a format which contains that component, such as the month/day/year format. E.g., one could specify the event start as @"6:00 May 1". The year component of the event period start and end date is ignored whether or not it is specified. Any missing value in the slot's series is treated as a non-subevent. The comparison with the value threshold is done to within 0.01% of the threshold's value. That is, values which are within 0.01% of the threshold's value are considered to have exceeded the threshold.

Syntax Example:

```
AnnualEventStats($ "Lottawatta Reservoir.Outflow",
                 @"24:00:00 February 28, 1994",
                 @"24:00:00 January 31, 2005",
                 @"24:00:00 May 31",
                 @"24:00:00 August 31",
                 100.0,
                 TRUE,
                 2.0,
                 TRUE )
Note: this call will determine how often outflow from Lottawatta Reservoir
exceeded 100 cfs more than 2 times between May and August in an eleven year
period starting in 1994.
```

Return Example:

```
{11.00, 3.00, 2.00}
```

There were eleven event periods, In 3 of those, the flow exceeded 100cfs more than 2 times, and there were 4 event periods (i.e the summers of 2001, 2002, 2003, and 2004) after the last event in 2001.

12. AvgObjectsAggregatedOverTime

This function returns a single numeric value obtained by averaging several objects' aggregated slot values. The objects' slot values may be aggregated as a **SUM**, **AVG**, **MIN**, or **MAX** over a specified time range.

Description	Aggregates several objects' values, each of which is the result of aggregating a slot's values over time.	
Type	NUMERIC	
Arguments		
1	STRING	Subbasin name
2	STRING	slot name
3	STRING	aggregation function ("SUM", "AVG", "MIN", or "MAX")
4	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
5	BOOLEAN	time conversion option ("TRUE" or "FALSE")
6	DATETIME	start date
7	DATETIME	end date
Evaluation	A list of slots is generated by searching all of the objects in the Subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values to be aggregated are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, each slot's values are aggregated according to the aggregation function argument over the time range of the datetime arguments. During each of these slot aggregations, any values which do not satisfy the aggregation filter argument are ignored. Finally, all of the object's aggregated slot values are averaged.	
Mathematical Expression	$\overline{\forall_{(obj \text{ in } subbasin)}[\forall_{(t \text{ from } start \text{ to } end)}[AggFunction_{(obj)}(obj.slotname)]]}$	
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, RiverWare aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.	

Syntax Example:

```
AvgObjectsAggregatedOverTime("upper basin",
                             "Inflow",
                             "MAX",
                             "ALL",
                             TRUE,
                             @"October, Previous Year",
                             @"September, Current Year")
```

Return Example:

```
52623.32 "cms"
```

13. AvgObjectsAtEachTimestep

This function evaluates to a list. Each item of the list is a list comprised of the datetime at which the average was performed, and the value of the average.

Description	Average several object's slot values, for each timestep in a range.	
Type	LIST{LIST{DATETIME, NUMERIC}}	
Arguments		
1	STRING	Subbasin name
2	STRING	slot name
3	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
4	BOOLEAN	time conversion option ("TRUE" or "FALSE")
5	DATETIME	start date
6	DATETIME	end date
Evaluation	A list of slots is generated by searching all of the objects in the Subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values to be averaged are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, all of the object's slot values are averaged, yielding one value for each timestep in the time range of the datetime arguments. The function returns a list of two items, where the first and second items of the inner lists are the datetime and the average value, respectively.	
Mathematical Expression	$\forall_{(t \text{ from } start \text{ to } end)}[\{t, \overline{\forall_{(obj \text{ in } subbasin)}[obj.slotname]}\}]$	

Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, RiverWare aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.

Syntax Example:

```
AvgObjectsAtEachTimestep("upper basin", "Storage", "ALL", FALSE
@"October, Previous Year",
@"September, Current Year")
```

Return Example:
For a monthly model, the above function would return something like:

```
{ { 24:00 October 31, 1996, 1233232.2 "m3" },
  { 24:00 November 30, 1996, 1067478.3 "m3" },
  ....
  { 24:00 September 30, 1997, 1563456.7 "m3" } }
```

14. AvgTimestepsAggregatedOverObjects

This function evaluates to a single numeric value. This value is the average, over time, of values resulting from aggregating several objects slot values at each timestep.

Description	Aggregate over a timeseries of values, each of which is the result of aggregating several objects' slot values.	
Type	NUMERIC	
Arguments		
1	STRING	Subbasin name
2	STRING	slot name
3	STRING	aggregation function ("SUM", "AVG", "MIN", or "MAX")
4	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
5	BOOLEAN	time conversion option ("TRUE" or "FALSE")
6	DATETIME	start datetime
7	DATETIME	end datetime

Evaluation	A list of slots is generated by searching all of the objects in the Subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values to be aggregated are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, all of the objects' slot values are aggregated according to the aggregation function argument for each timestep in the time range of the datetime arguments. During each of these slot aggregations, any values which do not satisfy the aggregation filter argument are ignored. Finally, the timeseries of object aggregated slot values are averaged.
Mathematical Expression	$\overline{\forall_{(t \text{ from } start \text{ to } end)}[\forall_{(obj \text{ in } subbasin)}[AggFunction_{(t)}(obj.slotname)]]}$
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, RiverWare aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.

Syntax Example:

```
AvgTimeStepsAggregatedOverObjects ("upper basin",
                                   "Storage",
                                   "MAX",
                                   "ALL",
                                   FALSE,
                                   @"October, Previous Year",
                                   @"September, Current Year")
```

Return Example:

```
230000 "m3"
```

15. AvgTimestepsForEachObject

This function evaluates to a list. Each item of the list is a list comprised of the object name and the average value of the slot on that object for the time range specified.

Description	Average a slot's values over a time range, for each object in a subbasin.	
Type	LIST {LIST {OBJECT, NUMERIC}}	
Arguments		
1	STRING	Subbasin name

2	STRING	slot name
3	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
4	BOOLEAN	time conversion option ("TRUE" or "FALSE")
5	DATETIME	start datetime
6	DATETIME	end datetime
Evaluation	A list of slots is generated by searching all of the objects in the Subbasin argument for slots which match the slot name argument. For each object, the slot's values are averaged over every timestep in the range of the datetime arguments. Any values which do not satisfy the aggregation filter argument are ignored during the calculation. If the time conversion option argument is TRUE, and the values to be aggregated are of the FLOW unit type, the values are first multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME.	
Mathematical Expression	$\forall_{(obj \text{ in } subbasin)}[\{obj, \overline{\forall_{(t \text{ from } start \text{ to } end)}[obj.slotname]}\}]$	
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, this function aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, this function also aborts RiverWare with an error.	

Syntax Example:

```
AvgTimestepsForEachObject("upper basin", "Storage", "ALL", TRUE,
@"October, Previous Year",@"September, Current Year")
```

Return Example:
For a monthly model, the above function would return something like:

```
{ { %"Res1", 1233232.2 "m3" }, { %"Res2", 1067478.3 "m3" },
  { %"Res3", 1997, 1563456.7 "m3" } }
```

16. Ceiling

This function rounds a numeric value up to the nearest multiple of a numeric factor.

Description	The ceiling numeric operation, to a multiple of a factor.	
Type	NUMERIC	
Arguments		
1	NUMERIC	the value
2	NUMERIC	the factor

Evaluation	Converts the value into the units of the factor, then returns the smallest integral multiple of the factor which is not less than the converted value. The returned value has the units of the factor.
Comments	Note that if the scalar portion of the factor is 1.0, then this function simply returns the ceiling of the value expressed in the units of the factor. If the two arguments are of a different unit type, this function aborts the run with an error.

Syntax Example:

```
Ceiling("Dry Reservoir.Pool Elevation"[], 100.0 "ft")
```

Return Example:

```
400 "ft"
```

17. ColumnLabel

Description	Returns the label associated with a given column of a table slot or aggregate series slot.	
Type	STRING	
Arguments		
1	SLOT	A table slot or agg series slot.
2	NUMERIC	The column index (0-based).
Evaluation	Returns the label of the column of the slot which has the given index.	
Comments	It is an error to provide an illegal index (e.g., an index of 4 with a table which has only 4 columns). If the column index is legal but there is no label for that column, then the empty string is returned: "".	

Syntax Example:

```
ColumnLabel(DataObjA.CoeffTable, 2)
```

Return Example:

```
"Coefficient 3"
```

18. ColumnLabels

Description	Returns a list containing the labels of the columns of a given table slot or agg. series slot, in order.	
Type	LIST of STRING values	
Arguments		
1	SLOT	A table slot or agg. series slot
Evaluation	Returns the label of the column of the table slot which has the given index.	
Comments	It is an error if the input slot has a type other than table slot or agg. series slot. For each column, if no label exists the empty string is returned.	

Syntax Example:

```
ColumnLabels(DeepLake.Elevation Volume Table)
```

Return Example:

```
{"Pool Elevation", "Storage"}
```

19. CompletePartialDate

Description	Fill in the missing components of a partially specified date/time.	
Type	DATETIME	
Arguments		
1	DATETIME	a partially specified date/time.
2	DATETIME	a source date/time, used to complete the other date.
Evaluation	Fills in the missing components of a partially specified date value. The missing component values are taken from the second parameter, a date value which, if not fully specified, should have at least the components which are missing from the date which is being completed. See the syntax examples below and see also related functions **HERE (PreviousDate)** and **HERE (NextDate)**.	
Comments	The behavior is not defined if the resulting date is not valid; for example, if the day of month is not valid for the month and year.	

Syntax Example:

```
CompletePartialDate(@"March", @"t")
```

Return Example:

```
24:00 March 2, 1994
```

(assuming the @”t” is the 2nd day of some month in 1994)

20. ComputeReservoirDiversions

<table>
<tr><td>Description</td><td colspan="2">Used to meet multiple water user demands using multiple reservoir diversions</td></tr>
<tr><td>Type</td><td colspan="2">LIST{LIST {SLOT, NUMERIC, OBJECT}}</td></tr>
<tr><td>Arguments</td><td></td><td></td></tr>
<tr><td>1</td><td>STRING</td><td>The computational subbasin used for the calculations</td></tr>
<tr><td>Evaluation</td><td colspan="2">Returns a LIST of slot, value triplets. Each triplet is a LIST that contains a slot (at index zero) and the value to set on that slot (at index one). The slot, value triplets computed by this function are for the subslots on the Supply From Reservoirs slot on each Water User object and the Incoming Available Water slot on each Water User object.

For each Water User in the specified subbasin:
• A list of supply reservoirs is generated by following the links to the Supply From Reservoirs slot
• The list of reservoirs is ranked by Operating Level in descending order.
• Each reservoir makes a diversion to meet the Water User’s Diversion Requested value. This value is limited by: the Maximum Delivery Rate specified on the Water User object that applies to the current reservoir, the Max Diversion specified on the Diversion object that applies to the current reservoir, and the amount of water remaining in the conservation pool.
• If the Limit by Reservoir Level method is selected (on the Water User object) a diversion cannot be made if the Demand Reservoir is in the flood pool or has a higher operating level than the supply reservoir.
• Each reservoir is visited until the Diversion Requested is met or there are no reservoirs left to consider.
• The function returns each subslot on each Supply From Reservoirs slot and the associated value. Also the Incoming Available Water slot on each Water User is returned with the value to be set on that slot. The Incoming Available Water is the sum of all the Supply From Reservoirs subslot values</td></tr>
</table>

Comments	The computational subbasin specified as the argument to this function must contain all the objects relevant to these calculations (Water Users, Diversion Objects, Reservoirs, etc.) The computational subbasin must have a method selected in the Diversions from Reservoirs category. Please consult the help file for the Computational Subbasin object (under Simulation Objects) for more details on this method category. The use of this function requires a specific configuration of objects and method selections. The schematic diagram below displays the required object and link configurations. Use of this function for USACE-SWD: **HERE (USACE_SWD.pdf, Section 3.8)**.

Syntax Example:

```
ComputeReservoirDiversions("Diversion Basin") if Diversion Basin contains two
reservoirs and the WU1 and WU2 water users connected to those reservoirs.
```

Return Example:

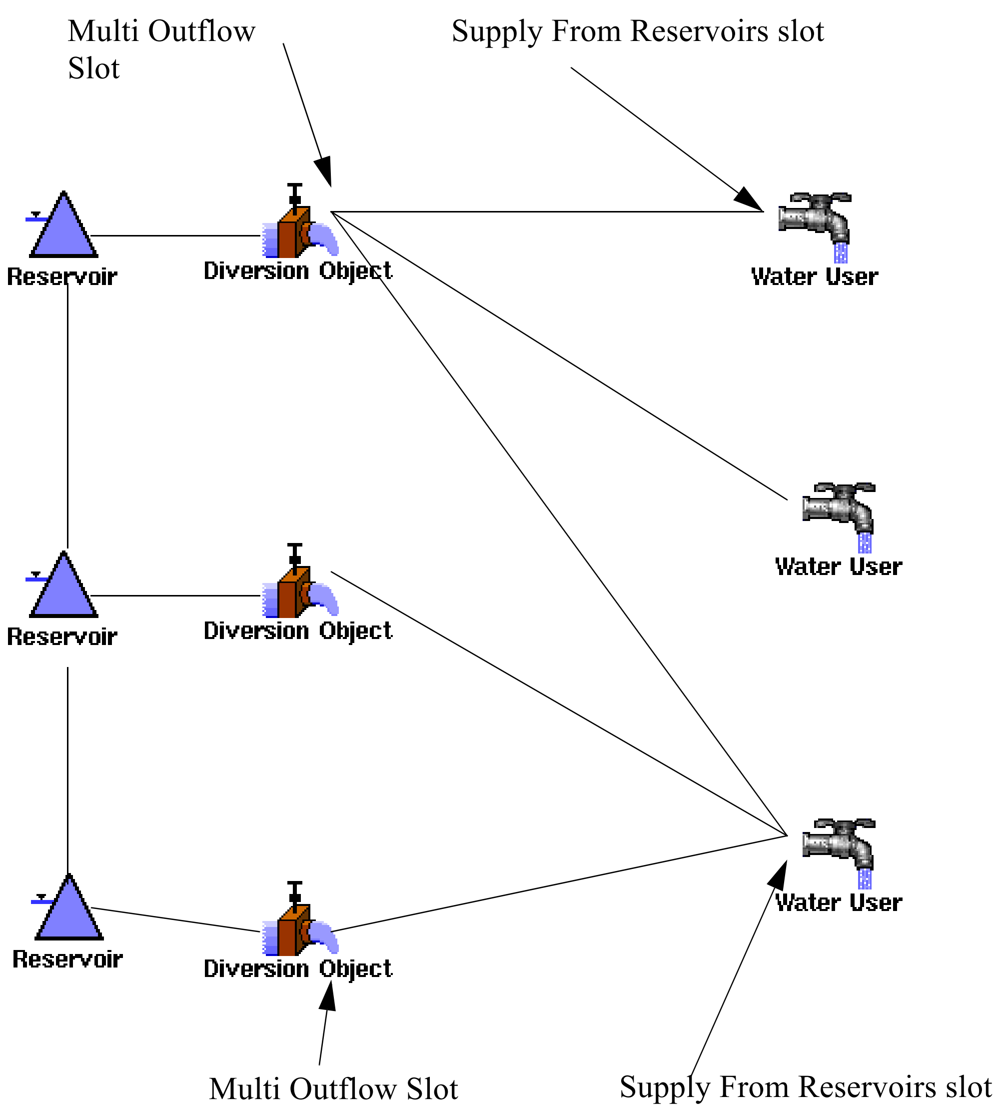
```
{ {"WU1.Supply From Reservoirs.WU1_Divert__dot__Multi Outflow",
2.26534773 "cms", "WU1"},
{"WU1.Incoming Available Water", 2.26534773 "cms", "WU1"},
{"WU2.Supply From Reservoirs.WU2_Divert__dot__Multi Outflow",
0.67960432 "cms", "WU2"},
{"WU2.Incoming Available Water", 0.67960432 "cms", "WU2"}  }
```

Use Examples:

```
FOR EACH ( LIST result IN ComputeReservoirDiversions("Diversion Basin")) DO
       result<0> [] = result<1>
END FOR EACH
```

In the diagram below, the Diversion slot on each reservoir is linked to the Diversion slot on the Diversion Object. The demands are represented by the Water User objects. The Supply From Reservoirs slot on each Water User is linked to the Multi Outflow slot on each Diversion Object that can act as a supply for that demand. The rule sets the values on the Supply From Reservoirs slots. These propagate to the Multi Outflow slots on the connected Diversion Objects. The Diversion objects solve for their Diversion slot. The Diversion values are passed to the Diversion slot on the Reservoir object and the water is removed from the Reservoir. On each reservoir, the Conservation and Flood Pools method in the Operating Levels category should be selected to instantiate the Bottom of Conservation Pool slot.

Schematic Diagram for ComputeReservoirDiversions Function:



21. DateMax

This function returns the later of two dates.

Description	Compare two dates and return that which is chronologically greater.	
Type	DATETIME	
Arguments		

1	DATETIME	a date
2	DATETIME	another date
Evaluation	The two dates are resolved and compared, the one which is chronologically greater is returned.	
Comments		

Syntax Example:

```
DateMax(@"t", @"January 1, 2001")
```

Return Example:

```
If current timestep is March 2, 2002: the function returns @"24:00 March 2, 2002"
If current timestep is May 3, 1999, the function returns @"24:00 January 1, 2001"
```

22. DateMin

This function returns the earlier of two dates.

Description	Compare two dates and return that which is chronologically lesser.	
Type	DATETIME	
Arguments		
1	DATETIME	a date
2	DATETIME	another date
Evaluation	The two dates are resolved and compared, the one which is chronologically lesser is returned.	
Comments		

Syntax Example:

```
DateMin(@"t", @"January 1, 2001")
```

Return Example:

```
If current timestep is May 2, 2002: the function returns @"24:00 January 1, 2001"
If current timestep is May 3, 1999, the function returns @"24:00 May 3, 1999"
```

23. DatesInPeriod

Description	Given a periodic slot and a date, this function returns an ordered list of dates representing the beginning time of each interval which begins in the specific period containing the input (reference) date.	
Type	LIST {DATETIME}	
Arguments		
1	SLOT	a periodic slot
2	DATETIME	a reference date
Evaluation	Every periodic slot has a period associated with it and this period is divided into intervals. Intervals are either regular (e.g., Days) or irregular (e.g., the beginning of one interval might be 8:00 July 3 of each period). One can map a period (divided into intervals) onto a time line, leading to several specific periods (divided into specific intervals). For example, the period "Year" maps onto specific periods corresponding to each year, such as the specific period which is the year 2003. Providing a reference date serves to indicate a specific period, and this function returns the dates corresponding to the beginning of each time interval which begins in that specific period.	
Comments	When the beginning of an interval occurs exactly at a period boundary (e.g., an interval beginning at "0:00 January 1" with an Yearly period), then we consider that interval to begin in the period occurring after midnight, not the one before. Note that not all time intervals (rows) defined in a Periodic Slot will correspond to intervals in a specific period. For example, for a period of Month, an interval might be defined which begins at "12:00 Day 30". This interval does not exist in all months and so for example if the reference date is 12:00 February 1, 2003, then the list returned by this function would not include the date 12:00 February 30, 2003.	

Syntax Example:

```
DatesInPeriod(TableA.AvePrecipitation, @"January 1, 2001")
```

Return Example:

```
If TableA.AvePrecipitation has 3 rows for 0:00 January 1, 6:00 June 15, and 24:00
September 1, Then the above function returns:
{ @"24:00 December 31, 2000", @"6:00 June 15, 2001", @"24:00 September 1, 2001"}
```

24. DateToNumber

Description	Given a Date/Time value, returns that date encoded as a numeric value of the type used by slots to containing date/time values.	
Type	NUMERIC	
Arguments		
1	DATETIME	The date/time value to encode as a numeric value.
Comments	Slots representing date/time values have unit type DateTime. The date/time value need not be fully specified, but the return value should only be assigned to a slot with appropriate units. For example, if the value @"January 1" should only be assigned to a slot with units "MonthAndDay".	

Syntax Example:

```
DateToNumber(@"t")
```

Return Example:

```
6508706400.00 - which is equivalent to "06:00 April 3, 2006" (FullDateTime)
```

Use Examples:

This function should be used in conjunction with dates on series slots **HERE (Slots.pdf, Section 4)** and the NumberToDate function **HERE (NumberToDate)**. A specific use example is shown **HERE (Slots.pdf, Section 4.3)**.

25. Destinations

This function evaluates to the list of user-defined Destinations

Description	This function returns a list of the names of all Destinations defined in the Water Accounting System Configuration.	
Type	LIST {STRING}	
Arguments		
Evaluation		
Comments	Destinations are properties of Supplies. The returned list does not include the default ("NONE") Destination.	

Syntax Example:

```
Destinations()
```

Return Example:

```
{"FarmerA", "City1", "City2"}
```

26. DestinationsFromObjectReleaseType

This function evaluates to the list of Destinations which represent outflows from an Object of a specified Release Type.

Description	This function returns a list of unique names of Destination Type of Supplies which represent outflows from a specified Object, and which have the indicated Release Type.	
Type	LIST {STRING}	
Arguments		
1	OBJECT	The Object.
2	STRING	Release Type name or "NONE" or "ALL"
Evaluation	The set of Accounts on the Object are examined. The outflow Supplies on those Accounts which link a different downstream Object and which have the indicated Release Type are considered. The names of the Destination Types of those Supplies are added to the returned list -- but any given Destination Type name will appear on the list only once. If the Release Type argument is "NONE," then only Supplies having the default (unassigned) Release Type are considered. If the Release Type argument is "ALL," then that attribute is ignored when considering Supplies.	
Comments	Destination Type and Release Types are properties of Supplies. The returned list can include the default ("NONE") Destination Type. Supplies which represent "internal flows" between two Accounts on the Object are not considered.	

Syntax Example:

```
DestinationsFromObjectReleaseType(%"Big Reservior", "Account Fill")
```

Return Example:

```
{"FarmerA", "City2"}
```

27. DispatchCount

Description	Returns the number of dispatch method executions that have occurred since the beginning of the current run.	
Type	NUMERIC	
Arguments		
Comments	Returns the number of dispatch method executions that have occurred since the beginning of the current run, if called during a dispatching run (Simulation or Rulebased Simulation). Otherwise, returns the total number of dispatch executions in the previous dispatching run.	

Syntax Example:

```
DispatchCount()
```

Return Example:

```
12,345
```

28. DispatchEndDate

This function returns the last timestep in the model for which dispatching is allowed.

Description	The last dispatch timestep.	
Type	DATETIME	
Arguments		
Evaluation	Returns the DATETIME that is the last timestep at which the current controller allows dispatching. If this function is called from a context in which the current controller does not have a last dispatch timestep (i.e. optimization), then the end date of the run is returned.	
Comments	The **Number of Post-Run Dispatch Timesteps** is set on the Run Control Parameters dialog for Simulation or Rulebased Simulation. For more information on changing the **Number of Post-Run Dispatch Timesteps**: **HERE (RunControl.pdf, Number of Post-Run Dispatch Timesteps:)**	

Syntax Example:

```
DispatchEndDate()
```

Return Example:

If the Run Finish is March 19, 2011 (daily timestep) and the **Number of Post-Run Dispatch Timesteps** is 3, the function will return:

```
24:00 March 22, 2011
```

29. DispatchTime

Description	Returns the accumulated time spent executing dispatch methods since the beginning of the current run.	
Type	NUMERIC	
Arguments		
Comments	Returns the accumulated time spent executing dispatch methods since the beginning of the current run, if called during a dispatching run (Simulation or Rulebased Simulation). Otherwise, returns the total time spent executing dispatch methods in the previous dispatching run	

Syntax Example:

```
DispatchTime()
```

Return Example:

```
67.8 seconds
```

30. Div

This function computes the integer division of two numbers.

Description	Integer division of two numbers.	
Type	NUMERIC	
Arguments		
1	NUMERIC	the numerator
2	NUMERIC	the units to which to convert the numerator
3	NUMERIC	the denominator

4	NUMERIC	the units to which to convert the denominator
Evaluation	Converts numerator and denominator into the specified units, then returns the integral division of the converted values, where integral division of x and y is defined as: $\mathrm{Div}(x, y) \equiv \left\lfloor \frac{\lfloor x \rfloor}{\lfloor y \rfloor} \right\rfloor$	
Comments	If the denominator is equal to zero, this function aborts the run with an error. Each of the units arguments must have units which are compatible with the value with which they are associated, otherwise the run is aborted with an error. Note that this function does not use the scalar portion of either of the units arguments.	

Syntax Example:

```
Div(10.5 "m", 0.0 "ft", 2.4 "sec", 0.0 "sec")
```

Return Example:

```
17.00 "0.304800 m/s"
```

31. ElevationToArea

These function performs a lookup in a Reservoir object's **Elevation Area Table** based on a given elevation and evaluates to the corresponding area.

Description	Find the surface area corresponding to a reservoir's elevation.	
Type	NUMERIC	
Arguments		
1	OBJECT	reservoir object
2	NUMERIC	pool elevation
Evaluation	The pool elevation argument is looked up in the **Pool Elevation** column, of the **Elevation Area Table,** of the reservoir object argument, to determine the **Surface Area**. If the exact elevation is not in the table, the lookup performs a linear interpolation between the two nearest bounding elevations and their corresponding surface areas.	
Mathematical Expression	$area = area_{(lesser)} + \frac{area_{(greater)} - area_{(lesser)}}{elev_{(greater)} - elev_{(lesser)}}(elevation - elev_{(lesser)})$	

Comments	If the object is not a reservoir, or the reservoir does not have an **Elevation Area Table**, the function aborts the run with an error (**CRSSEvaporationCalc**, **DailyEvaporationCalc**, **PanAndIceEvaporation**, or **InputEvaporation** must be selected as the **Evaporation and Precipitation** Category selected Method. This function will issue an error if the “Time Varying Elevation Area” method, **HERE (Objects.pdf, Section 22.1.24.2)**, is selected. Instead, use the ElevationToAreaAtDate function described next.

Syntax Example:

```
ElevationToArea(%"Lake Mead", 1210.03 "ft")
```

Return Example:

```
634547087.2 [m2]
```

32. ElevationToAreaAtDate

This function performs a lookup in the Reservoir object’s **Elevation Area Table** or **Elevation Area Table Time Varying** based on a given elevation and datetime and evaluates to the corresponding surface area. This function must be used when the “Time Varying Elevation Area” method is selected. Otherwise, the ElevationToArea function can be used and no DATETIME argument is required.

Description	Find the surface area corresponding to a reservoir’s elevation.	
Type	NUMERIC	
Arguments		
1	OBJECT	reservoir object
2	NUMERIC	pool elevation
3	DATETIME	the datetime at which to do the conversion

Evaluation	On the specified reservoir object argument, if the "Time Varying Elevation Area" method is selected, **HERE (Objects.pdf, Section 22.1.24.2)**, the function will reference the **Elevation Area Table Time Varying** table. The function will select the appropriate column to use based on the datetime argument. On timesteps that exactly match a modification date, the previous column is used. The relationship changes only at the end of that timestep and is taken into account when the reservoir dispatches. For this algorithm the previous column relationship is used. Otherwise, the **Elevation Area Table** is used and the datetime is ignored. Then, the pool elevation argument is looked up in the **Pool Elevation** column to determine the **Surface Area** from the appropriate Surface Area column. If the exact elevation is not in the table, the lookup performs a linear interpolation between the two nearest bounding elevations and their corresponding surface areas.
Mathematical Expression	$area = area_{(lesser)} + \frac{area_{(greater)} - area_{(lesser)}}{elev_{(greater)} - elev_{(lesser)}}(elevation - elev_{(lesser)})$
Comments	If the object is not a reservoir, or the reservoir does not have an **Elevation Area Table** or **Elevation Area Table Time Varying**, the function aborts the run with an error (i.e. a method must be selected in the **Evaporation and Precipitation** Category).

Syntax Example:

```
ElevationToAreaAtDate(%"Lake Mead", 1210.03 "ft", @"t")
```

Return Example:

```
634547087.2 [m2]
```

33. ElevationToMaxRegulatedSpill

This function performs a lookup in a Reservoir object's **Regulated Spill Table** based on a given elevation and evaluates to the corresponding maximum regulated spill.

Description	Find the maximum regulated spill at a given reservoir elevation.	
Type	NUMERIC	
Arguments		
1	OBJECT	reservoir object
2	NUMERIC	pool elevation
3	DATETIME	datetime context for unit conversions

Evaluation	The pool elevation argument is looked up in the **Pool Elevation** column, of the **Regulated Spill Table,** of the reservoir object argument, to determine the **Max Regulated Spill.** If the exact elevation is not in the table, the lookup performs a linear interpolation between the two nearest bounding elevations and their corresponding maximum regulated spills.
Mathematical Expression	$max\ spill\ =\ max\ spill_{(lesser)} + \dfrac{max\ spill_{(greater)} - max\ spill_{(lesser)}}{elev_{(greater)} - elev_{(lesser)}}(elevation - elev_{(lesser)})$
Comments	If the object is not a reservoir, or the reservoir does not have a **Regulated Spill Table**, the function aborts the run with an error (**Regulated**; **Regulated and Unregulated**; **Regulated and Bypass**; **Regulated, Bypass and Unregulated**; or **Bypass, Regulated and Unregulated** must be the **Spill** category selected method).

Syntax Example:

```
ElevationToMaxRegulatedSpill(%"Lake Mead", 1210.03 "ft",
@"t")
```

Return Example:

```
1783.25 [cms]
```

34. ElevationToStorage

This function performs a lookup in a Reservoir object's **Elevation Volume Table** based on a given elevation and evaluates to the corresponding storage.

Description	Find the reservoir storage at a given elevation.	
Type	NUMERIC	
Arguments		
1	OBJECT	reservoir object
2	NUMERIC	pool elevation
Evaluation	The pool elevation argument is looked up in the **Pool Elevation** column, of the **Elevation Volume Table,** of the reservoir object argument to determine the **Storage.** If the exact elevation is not in the table, the lookup performs a linear interpolation between the two nearest bounding elevations and their corresponding storage values.	
Mathematical Expression	$storage\ =\ storage_{(lesser)} + \dfrac{storage_{(greater)} - storage_{(lesser)}}{elev_{(greater)} - elev_{(lesser)}}(elevation - elev_{(lesser)})$	

Comments	If the object is not a reservoir, the function aborts the run with an error. If the reservoir is a Slope Power Reservoir, the calculation is based only on level storage and does not include any wedge storage effects. This function will issue an error if the “Time Varying Elevation Volume” method, **HERE (Objects.pdf, Section 22.1.24.2)**, is selected. Instead, use the ElevationToStorageAtDate function described next.

Syntax Example:

```
ElevationToStorage(%"Lake Mead", 1210.03 "ft")
```

Return Example:

```
2212323.233 "m3"
```

35. ElevationToStorageAtDate

This function performs a lookup in the Reservoir object’s **Elevation Volume Table** or **Elevation Volume Table Time Varying** based on a given elevation and datetime and evaluates to the corresponding volume. This function must be used when the “Time Varying Elevation Volume” method is selected. Otherwise, the ElevationToStorage function can be used and no DATETIME argument is required.

Description	Find the volume corresponding to a reservoir’s elevation.	
Type	NUMERIC	
Arguments		
1	OBJECT	reservoir object
2	NUMERIC	pool elevation
3	DATETIME	the datetime at which to do the conversion

Evaluation	On the specified reservoir object argument, if the "Time Varying Elevation Volume" method is selected, **HERE (Objects.pdf, Section 22.1.23.3)**, the function will reference the **Elevation Volume Table Time Varying** table. The function will select the appropriate column to use based on the datetime argument. On timesteps that exactly match a modification date, the previous column is used. The relationship changes at the end of that timestep and is taken into account when the reservoir dispatches. For this algorithm, the previous column relationship is used. Otherwise, the **Elevation Volume Table** is used and the datetime is ignored. Then, the pool elevation argument is looked up in the **Pool Elevation** column to determine the **Volume** from the appropriate column. If the exact elevation is not in the table, the lookup performs a linear interpolation between the two nearest bounding elevations and their corresponding surface areas.
Mathematical Expression	$storage = storage_{(lesser)} + \frac{storage_{(greater)} - storage_{(lesser)}}{elev_{(greater)} - elev_{(lesser)}}(elevation - elev_{(lesser)})$
Comments	If the object is not a reservoir, or the reservoir does not have an **Elevation Volume Table** or **Elevation Volume Table Time Varying**, the function aborts the run with an error.

Syntax Example:

```
ElevationToStorageAtDate(%"Lake Mead", 1210.03 "ft", @”t”)
```

Return Example:

```
634547087.2 [m3]
```

36. ElevationToUnregulatedSpill

This function performs a lookup in a Reservoir object's **Unregulated Spill Table** based on a given elevation and evaluates to the corresponding unregulated spill.

Description	Find the unregulated spill at a given reservoir elevation.	
Type	NUMERIC	
Arguments		
1	OBJECT	reservoir object
2	NUMERIC	pool elevation
3	DATETIME	datetime context for unit conversions

Evaluation	The pool elevation argument is looked up in the **Pool Elevation** column, of the **Unregulated Spill Table,** of the reservoir object argument, to determine the **Unregulated Spill**. If the exact elevation is not in the table, the lookup performs a linear interpolation between the two nearest bounding elevations and their corresponding unregulated spills.
Mathematical Expression	$unreg\ spill = unreg\ spill_{(lesser)} + \frac{unreg\ spill_{(greater)} - unreg\ spill_{(lesser)}}{elev_{(greater)} - elev_{(lesser)}} \times (elevation - elev_{(lesser)})$
Comments	If the object is not a reservoir, or the reservoir does not have an **Unregulated Spill Table**, the function aborts the run with an error (**Unregulated**; **Regulated and Unregulated**; **Regulated, Bypass and Unregulated**; or **Bypass, Regulated and Unregulated** must be the **Spill** selected method).

Syntax Example:

```
ElevationToUnregulatedSpill(%"Lake Mead", 1210.03 "ft",
@"t")
```

Return Example:

```
1212.25 [cms]
```

37. Exp

Description	Exponentiation of a dimensionless quantity.	
Type	NUMERIC	
Arguments		
1	NUMERIC	the operand
2	NUMERIC	the exponent
Evaluation	Returns the result of exponentiating the operand to the power given by the exponent. The return value is dimensionless (has no units).	
Comments	The exponent is not restricted to being an integer (as with the "^" operator), but it is an error for the operand to have units.	

Syntax Example:

```
Exp(16.0, 0.5)
```

Use Examples:

```
4.0000 "None"
```

38. FilterByObjType

This function evaluates to a list of objects containing objects from the original list which match the specified types.

Description	Filter a list of objects to include only object(s) of the specified type(s).	
Type	LIST {OBJECT}	
Arguments		
1	LIST	list of objects
2	LIST	list of object types to include, where each object type is expressed as a STRING.
Evaluation	The list of object types to include is parsed and mapped to RiverWare object types. Then, the list of objects is evaluated in order, and each object which is one of the requested object types is added to the returned list.**The spellings and capitalization of objects can be found in the Subbasin Manager under the Automatic tab.**	
Mathematical Expression	$\{OBJECT\} = \{OBJECT\} \cap \{OBJECT\ TYPE\}$	
Comments	The order of objects is preserved from the argument object list to the returned object list. The list arguments may contain any number of items. If either of the arguments is an empty list, the function evaluates to an empty list.	

Syntax Example:

```
FilterByObjType({%"Lake Mead",%"Lake Powell",%"Virgin River"},
                {"LevelPowerReservoir"})
```

Syntax Example:

```
{%"Lake Mead",%"Lake Powell"}
```

39. FlattenList

Description	This function takes a list and replaces any lists contained within that list with the individual items from those lists.	
Type	NUMERIC	
Arguments		
1	LIST	the list to be "flattened"
Evaluation	For each item in the input list, if the item is not a list, it is appended to the answer list, if it is a list, then it is flattened and then all of its items are appended to the answer list in turn.	
Comments		

Syntax Example:

```
FlattenList( {1, {2, 3}, {{4}}})
```

Return Example:

```
{ 1, 2, 3, 4}
```

40. FloodControl

This function invokes the selected Flood Control method on a computational subbasin **HERE (Objects.pdf, Section 7.1.3)**.

Description	Invokes computational subbasin's selected Flood Control method.	
Type	LIST { LIST { SLOT, NUMERIC, OBJECT } }	
Arguments		
1	STRING	the name of the computational subbasin
Evaluation	Runs the selected Flood Control method on the subbasin. Returns a list of { slot, value, object } sets. For each reservoir in the subbasin, three sets may be returned: one for the Outflow slot, one for the Flood Control Release slot on the reservoir, and one for the Target Balance Level on the reservoir.	
Comments	The calling rule is expected to make the assignments of the values to the slots. Typically, this function should be called only once per timestep. To constrain this, use the following as an execution constraint: NOT(HasRuleFiredSucessfully("Rule Name")) Use of this function for USACE-SWD: **HERE (USACE_SWD.pdf, Section 3.7)**.	

Syntax Example:

```
FloodControl("Flood Basin") where "Flood Basin" contains Res1 and Res2.
```

Return Example:

```
{ {"Res1.Outflow", 6344.32 "cfs", "Res1"},
  {"Res1.Flood Control Release", 6344.32 "cfs", "res1"},
  {"Res1.Target Balance Level", 8.32, "res1"} ,
  {"Res2.Outflow", 3243.02 "cfs", "Res2"},
  {"Res2.Flood Control Release", 2312.20 "cfs", "Res2"},
  {"Res2.Target Balance Level", 8.32, "Res2"} }
```

Use Examples:

```
FOREACH (LIST triplet IN FloodControl( "Flood Basin" )) DO
       ( triplet<0> )[] = triplet<1>
ENDFOREACH
```

41. Floor

This function rounds a numeric value down to the nearest multiple of a numeric factor.

Description	The floor numeric operation, to a multiple of a factor.	
Type	NUMERIC	
Arguments		
1	NUMERIC	the value
2	NUMERIC	the factor
Evaluation	Converts the value into the units of the factor, then returns the largest integral multiple of the factor which is not greater than the converted value. The returned value has the units of the factor.	
Comments	Note that if the scalar portion of the factor is 1.0, then this function simply returns the floor of the value expressed in the units of the factor. If the two arguments are of a different unit type, this function aborts the run with an error.	

Syntax Example:

```
Floor("Wet Reservoir.Pool Elevation"[], 100.0 "ft")
```

Return Example:

```
If Wet Reservoir.Pool Elevation[] is 5343.35ft, then the above example will
evaluate to 5300.0 ft
```

42. FlowToVolume

This function evaluates to the volume of water corresponding to a flow over a timestep.

Description	The volume of water resulting from a steady flow over a timestep.	
Type	NUMERIC	
Arguments		
1	NUMERIC	constant flow to be converted
2	DATETIME	timestep over which to convert
Evaluation	The number of seconds in the timestep of the datetime argument is determined. Then, the flow argument is multiplied by this number of seconds. Returns value in units of volume.	
Mathematical Expression	$volume = flow \times \Delta t_{(current\ timestep)}$	
Comments	If the flow argument is entered in units containing a "/month" component, it is scaled to reflect the length of the month indicated by the timestep argument before being multiplied by this timestep length.	

Syntax Example:

```
FlowToVolume(Lake Powell.Inflow[], @"t")
```

Return Example:

```
6155584.04 [m3]
```

43. Fraction

This function returns the fractional remainder after dividing two numbers.

Description	The fractional remainder after division.
Type	NUMERIC

Arguments		
1	NUMERIC	the numerator
2	NUMERIC	the denominator
Evaluation	Converts the numerator into the units of the denominator, divides the result by the denominator, then returns the fractional portion of the division. In other words: $\mathrm{Fraction}(x, f) \equiv \frac{x}{f} - \mathrm{Floor}(x, f)$ The returned value has the units of "factor".	
Comments	Note that if the scalar portion of the denominator is 1.0, then this function simply returns the fractional portion of the first argument when it is expressed in the units of the denominator. If the values are of a different unit type, this function aborts the run with an error.	

Syntax Example:

```
Fraction("Whitewater Creek.Inflow"[], 1.0 "cms")
```

Return Example:

```
If Whitewater Creek.Inflow is 134.3 cfs, the above function returns:
0.80295250 "cms"
```

44. Get3DTableVals

Description	Return the contents of a Table Slot that is structured for 3D table interpolation.	
Type	LIST{LIST {NUMERIC LIST{NUMERIC} LIST{NUMERIC}}}	
Arguments		
1	SLOT	the table slot whose values are to be returned.
2	NUMERIC	z column index (zero-based)
3	NUMERIC	x column index (zero-based)
4	NUMERIC	y column index (zero-based)
Evaluation	Returns the contents of a 3D table as a list of the table values associated with successive z value. For each distinct z value in the table slot, the returned list contains a sublist with the following values: 1) The current z value 2) The list of x values associated with the current z value 3) The list of y values associated with the current z value	
Comments	Units are not required for row and column indices and, if provided, will be ignored. In the context of rulebased simulation, if one of the slot's values is NaN, the function exits the rule with an early termination.	

Syntax Example:

```
Get3DTableVals(Wet Reservoir.Plant Power Table, 0, 1, 2)
```

Operating Head (m)	Turbine Release (cms)	Power (HP)
320	0.00	0
320	120.32	470
340	5.00	10
340	127.32	500

Return Example:

```
{{320.00 "m", {0.00 "cms", 120.32 "cms"}, {0 "HP", 470 "HP"}},
 {340.00 "m", {5.00 "cms", 127.32 "cms"}, {10 "HP", 500 "HP"}}}
```

45. GetAccountFromSlot

Description	Return the name of a slot's account.	
Type	STRING	
Arguments		
1	SLOT	The slot whose account is returned.
Comments	It is an error if the slot is not on an account.	

Syntax Example:

```
GetAccountFromSlot($"ResA^Municipal.Inflow")
```

Return Example:

```
"Municipal"
```

46. GetAllNamedBasins

This function evaluates to a list containing the names of the user defined subbasins in a model.

Description	The names of all user defined subbasins in the current model, expressed as strings.	
Type	LIST {STRING}	
Arguments	none	
Evaluation	The function first retrieves a list of all defined subbasins in the model, then filters out any automatic subbasins (object type basins generated by RiverWare).	
Comments	If there are no user defined subbasins in the model, this function evaluates to an empty list.	

Syntax Example:

```
GetAllNamedBasins()
```

Return Example:

```
{"Upper", "Flood Control", "Lower"}
```

47. GetColMapVal

Description	Get a column map value from a periodic slot given a date and a value. This is the inverse of the way values are usually accessed in periodic slots with column maps (i.e., given a date and column map value, find the corresponding slot value).	
Type	NUMERIC	
Arguments		
1	SLOT	The periodic slot to be accessed.
2	DATETIME	The date to be used to index into the time dimension of the Periodic Slot (its row map).
3	NUMERIC	The value to use for the lookup, having the same type of units as the values in the periodic slot itself.
Evaluation	If the default access method for the table is "lookup", then we first find the row whose associated time interval contains the input date. We then find the two consecutive values in that row whose values bracket the input value. We then find the column map values associated with these two values, and return a value interpolated between them according to where the input value falls between its two bracketing values. If the default access method is "interpolation" then the procedure described above is followed for the row whose time interval follows the given date, and the return value is interpolated between the values found for the two rows.	
Comments	The input slot must be a periodic slot with a column map, the numeric value must have units compatible with the units of the periodic slot, for the relevant time interval(s), the slot values must be either a monotonically non-decreasing or monotonically non-increasing function of the column map values, and the input value must fall within the domain of that function. If there are multiple possible return values, i.e., if the input value appears for multiple columns, then the largest column map value is returned.	

Syntax Example:

```
GetColMapVal(Meander Res.Operating Level Table, @"t", 1.0)
```

Return Example:

```
2.323
```

48. GetColumnIndex

Description	The index of the table slot or agg. series slot column whose name matches a string.	
Type	NUMERIC	
Arguments		
1	SLOT	the table slot or agg. series slot in which to find a column
2	STRING	the name of the column to match
Evaluation	The labels of the slot columns are compared to the string argument until a match is found.	
Comments	Slot column and row indices are zero based and have units of type [NONE]. If the specified slot is not a table slot or agg. series slot, or the specified string is not the label of a column on the slot, this function aborts the run with an error. If several columns of the slot match the string argument, this function evaluates to the index of the left-most matching column.	

Syntax Example:

```
GetColumnIndex(RiverData.Minimum Flow,"Dolores")
```

Return Example:

```
0.000
```

49. GetDate

Description	Interpret a string as a date.	
Type	DATETIME	
Arguments		
1	STRING	Textual representation of a date/time.
Evaluation	Returns the date which corresponds to the input text. Legal text is the same as is legal for symbolic date/times. For example, the expression: GetDate("January 1, Current Year") is exactly equivalent to the expression: @"January 1, Current Year".	

Syntax Example:

```
GetDate("January 20, 1996")
```

Return Example:

```
@"24:00 January 20, 1996"
```

50. GetDates

This function evaluates to a list of datetimes; from a start datetime to an end datetime, with a given interval separating the dates.

Description	Generate a list of datetimes between two datetimes at a given interval.	
Type	LIST {DATETIME}	
Arguments		
1	DATETIME	starting datetime
2	DATETIME	ending datetime
3	STRING	string representation of a datetime interval expressed as an integer, a space, and a time unit.
Evaluation	The starting datetime and ending datetime; which may be specified symbolically, are converted into actual datetimes. The string representation of the interval is resolved into a time length. Then, a list is created beginning with the starting datetime. The time length is added to each previous datetime in the list until the resulting datetime is later than the ending datetime.	

Mathematical Expression	$\{DATETIME\} = \{start \le end, (datetime_{(n-1)} + interval) \le end, \ldots\}$
Comments	If the ending date is before the starting date, the function evaluates to an empty list. If the ending date is equal to the starting date, or if the time interval is larger than the interval between the starting and ending dates, the function evaluates to a list which only contains the start date. The accepted datetime interval units are: • hours or Hours • days or Days • weeks or Weeks • months or Months • years or Years

Syntax Example:

```
GetDates(@"January 20, 1996", @"January Max DayOfMonth, 1996",
"6 Hours"}
```

Return Example:

```
{@"24:00 January 20, 1996", @"06:00 January 21, 1996", @"12:00 January 21, 1996",
...}
```

51. GetDatesCentered

This function evaluates to a list of datetimes, centered around a given date.

Description	Generate a list of datetimes separated by a given interval, and centered at a given date. If desired, dates not within the run duration are filtered out of the list.	
Type	LIST {DATETIME}	
Arguments		
1	DATETIME	center datetime
2	NUMERIC	number of dates to return in the list
3	STRING	string representation of a datetime interval expressed as an integer, a space, and a time unit
3	BOOLEAN	whether to limit return dates to those within the run

Evaluation	The center datetime, which may be specified symbolically, is converted into an actual datetime. The string representation of the interval is resolved into a time length. Then a list is created with the given number of dates, each separated by the given time interval. The center date is always included in this list, with an equal number of dates appearing before and after it (in the case of an odd number of dates). If their are an even number of dates, then there is one more date appearing before the center date than appear after. After the list has been created, if the user has specified that they only want dates within the run duration, then all other dates are filtered out of the return list.
Mathematical Expression	$\{DATETIME\} = \left\{ d + -\left\lceil \frac{n}{2} \right\rceil \text{offset},\ d + -\left(\left\lceil \frac{n}{2} \right\rceil - 1\right)\text{offset},\ \ldots,\ d + \left\lfloor \frac{n}{2} \right\rfloor \text{offset} \right\}$
Comments	The accepted datetime interval units are: • hours or Hours • days or Days • weeks or Weeks • months or Months • years or Years

Syntax Example:

```
GetDatesCentered(@"January 20, 1996", 3, "6 Hours", true}
```

Return Example:

```
{@"18:00 January 20, 1996", @"24:00 January 20, 1996", @"06:00 January 21, 1996"}
```

52. GetDayOfMonth

This function evaluates to a number which represents the day of the month of the given datetime in units of time

Description	The day of the month as a unit of time.	
Type	NUMERIC	
Arguments		
1	DATETIME	the datetime whose day of month to determine
Evaluation	The datetime argument; which may be specified symbolically, is converted into an actual datetime. Then, the day of the month in which the datetime is, is determined. This function requires that the datetime be at least partially specified with a valid month and day, E.g. @"January 1" or @"Current Month 23" will work.	

Comments	When displayed, the return value will be displayed according to the active unit scheme's time unit type rule. For example, if the active unit scheme displays Time values as Hours, then the return value for @"January 2" will be displayed as 48 "hours". To convert the return value into a dimensionless value representing the number of days, divide it by 1 "day". As elsewhere in RiverWare 24:00 hours is considered to be the day which is ending, and 00:00 hours is considered to be the day which is just beginning.

Syntax Example:

```
GetDayOfMonth(@"February 23, 1996")
```

Return Example:

```
23.0 "day" or 553 "hour"
```

53. GetDayOfYear

This function evaluates to a number which represents the day of the year of the given datetime.

Description	The day of the year as a one-based integer in units of time.	
Type	NUMERIC	
Arguments		
1	DATETIME	the datetime whose day of the year to determine
Evaluation	The datetime argument; which may be specified symbolically, is converted into an actual datetime. Then, the day of the year in which the datetime is contained, is determined. This function requires that the specified datetime resolve to a fully specified datetime or an error will occur.	
Comments	When displayed, the return value will be displayed according to the active unit scheme's time unit type rule. For example, if the active unit scheme displays Time values as Hours, then the return value for @"January 2" will be displayed as 48 "hours". To convert the return value into a dimensionless value representing the number of days, divide it by 1 "day". As elsewhere in RiverWare, 24:00 hours is considered to be the day which is ending, and 00:00 hours is considered to be the day which is just beginning.	

Syntax Example:

```
GetDayOfYear(@"February 23, 1996")
```

Return Example:

```
54.0 "day" or 1296 "hour"
```

54. GetDaysInMonth

This function evaluates to the number of days in the month of the given datetime.

Description	The number of days in the month in units of time.	
Type	NUMERIC	
Arguments		
1	DATETIME	the datetime of any time within the month
Evaluation	The datetime argument; which may be specified symbolically, is converted into an actual datetime. Then, the number of days in the month in which the datetime is contained, is determined. This function requires that the specified datetime resolve to a fully specified datetime or an error will occur.	
Comments	When displayed, the return value will be displayed according to the active unit scheme's time unit type rule. For example, if the active unit scheme displays Time values as Hours, then the return value for @"January 2" will be displayed as 744 "hours". To convert the return value into a dimensionless value representing the number of days, divide it by 1 "day". As elsewhere in RiverWare, 24:00 hours is considered to be the day which is ending, and 00:00 hours is considered to be the day which is just beginning.	

Syntax Example:

```
GetDaysInMonth(@"February 23, 1996")
```

Return Example:

```
29.0 "day" or 696 "hour"
```

55. GetDisplayVal

Description	This function takes a series or periodic slot and a date and returns the value of the slot at the given date, in units based on the display scale and units for that slot.	
Type	NUMERIC	
Arguments		
1	SLOT	the series or periodic slot whose value is to be returned
2	DATETIME	the datetime of the value to be returned
Evaluation		
Comments	The function returns and error and aborts the run if the input slot is not a series or periodic slot, if the date is not fully specified, or if the date is not contained in the series.	

Syntax Example:

```
GetDisplayVal(MyReservoir.Outflow, @"24:00 February 23, 1996")
```

Return Example:

```
3.03012926 "1000 * cfs"
```

56. GetDisplayValByCol

Description	This function takes an agg series slot or periodic slot, a date, and a column, and returns the value of the slot in the given column and at the given date, in units based on the display scale and units for that slot.	
Type	NUMERIC	
Arguments		
1	SLOT	the series or periodic slot whose value is to be returned
2	DATETIME	the datetime of the value to be returned
3	NUMERIC	the column, interpreted as a 0-based integral index
Evaluation	The function returns and error and aborts the run if the input slot is not of an appropriate type, if the date is not fully specified, or if the date is not contained in the series.	

Syntax Example:

```
GetDisplayValByCol(MyData.Flow, @"February 23, 1996", 1.0)
```

Return Example:

```
3.03012926 "1000*cfs"
```

57. GetElementName

Given an element in an aggregate object, this function returns its name.

Description	Return the name of an element in an aggregate object, without the name of the object's name prepended.	
Type	STRING	
Arguments		
1	OBJECT	the element of an aggregate object (e.g., a WaterUser within an AggDiversionSite) whose name is to be returned.
Evaluation	The function returns the name of the element object.	

Comments	This function returns only the name of the element itself, without the name of the parent aggregate object. If the full name is desired, then one may use the built-in RPL operation STRINGIFY. If the object argument is not an element of an aggregate object, then the run is aborted with an error.

Syntax Example:

```
GetElementname(% "Below Abiquiu Diversions:Chamita")
```

Return Example:

```
"Chamita"
```

58. GetEnsembleTraceValue

Given a keyword name for trace metadata when using an ensemble, return the keyword value for the current trace executing in a run.

Description	Return the value for a trace keyword for the current trace executing in a run.	
Type	STRING	
Arguments		
1	STRING	the name of a trace metadata keyword.
Evaluation	The function returns the value for the trace keyword for the currently executing run.	
Comments	If the function is called outside of a run or if the trace metadata keyword cannot be found, then the function fails. This function would typically be called during a multiple run when input ensembles are used in the MRM configuration. If a single trace is configured for an ensemble dataset to use outside of a multiple run and a DMI with this dataset is invoked during a single run, the metadata for that trace would also be available to query.	

Syntax Example:

```
GetEnsembleTraceValue("Hydrology")
```

Return Example:

```
"Dry"
```

59. GetEnsembleValue

Given a keyword name for ensemble metadata when using an ensemble, return the keyword value for the metadata for that run.

Description	Return the value for an ensemble keyword for the current run.	
Type	STRING	
Arguments		
1	STRING	the name of an ensemble metadata keyword.
Evaluation	The function returns the value for the ensemble keyword for the currently executing run.	
Comments	If the function is called outside of a run or if the ensemble metadata keyword cannot be found, then the function fails. This function would typically be called during a multiple run when input ensembles are used in the MRM configuration. If a single trace is configured for an ensemble dataset to use outside of a multiple run and a DMI with this dataset is invoked during a single run, the metadata for that ensemble would also be available to query.	

Syntax Example:

```
GetEnsembleValue("Hydrology")
```

Return Example:

```
"Historical"
```

60. GetJulianDate

This function evaluates to the Julian date of the given datetime.

Description	The Julian date of the timestep in units of "NONE".	
Type	NUMERIC	
Arguments		
1	DATETIME	the datetime whose Julian date to evaluate to

Evaluation	The datetime argument; which may be specified symbolically, is converted into an actual datetime. Then, the Julian date of this timestep is determined. This function requires that the specified datetime resolve to a fully specified datetime or an error will occur.
Comments	Julian Dates are represented as the number of days from noon GMT on January 1, 4713 B.C. (47120101 12:00 P.M. GMT). Julian Dates in RiverWare also include the decimal fraction of the day down to 0.00001, the equivalent of 1 second.

Syntax Example:

```
GetJulianDate(@"14:31:59 February 23, 1996")
```

Return Example:

```
2450137.10554398
```

61. GetLinkedObjs

Description	Given a slot, returns a list of the Objects which contain the slots to which the input slot is linked.	
Type	LIST {OBJECT}	
Arguments		
1	SLOT	the slots whose links are to be explored
Evaluation	For each slot to which the input slot is linked, we determine if that slot is managed by a Objects; if so, it is added to the return list. Thus, an empty list is returned if the slot is not linked to any slots on a Objects.	
Comments	It is considered an error if the input slot is not a Series Slot.	

Syntax Example:

```
GetLinkedObjs("Res A.Inflow")
```

Return Example:

```
{%"Reach 1", %"Reach 2"}
```

62. GetLowerBound

Description	Returns the lower bound for the specified series slot	
Type	NUMERIC	
Arguments		
1	SLOT	the slot whose bound is to be returned.
Evaluation		
Comments	It is considered an error if the specified slot is not a Series Slot with a valid lower bound. The lower bound is specified in the slot configuration.	

Syntax Example:

```
GetLowerBound("Res A.Power")
```

Return Example:

```
0.0 MW
```

63. GetLowerBoundByCol

Description	Returns the lower bound for the column of the specified agg series slot.	
Type	NUMERIC	
Arguments		
1	SLOT	the agg series slot whose bound is to be returned.
2	NUMERIC	the column index (0-based).
Evaluation		
Comments	It is considered an error if the input slot is not an Agg Series Slot with a valid lower bound for the given column. The lower bound is specified in the slot configuration.	

Syntax Example:

```
GetLowerBoundByCol("Res A.Hydro Block Use", 3)
```

Return Example:

```
0.0 MWH
```

64. GetMaxOutflowGivenHW

This function evaluates to the maximum Outflow from a StorageReservoir, LevelPowerReservoir, or SlopePowerReservoir with the given Pool Elevation at the specified timestep.

Description	The maximum combined outflow of a reservoir, including outlet works or turbine release, and any possible regulated, unregulated, and/or bypass spills.	
Type	NUMERIC	
Arguments		
1	OBJECT	the reservoir object for which to calculate
2	NUMERIC	the end of timestep pool elevation (HW or headwater)
3	DATETIME	the timestep at which to calculate
Evaluation	This function calls the getMaxOutGivenHW() function on the given reservoir object at the given timestep, and provides it with the end of timestep pool elevation. This ending HW is averaged with the previous timestep’s ending **Pool Elevation** to yield an average HW over the timestep. The average HW is then used to compute the following outflows: • release (if the object is a StorageReservoir): The maximum release is determined from a table interpolation in the **Max Release** table using the average HW as the lookup in the **Pool Elevation** column. • turbine release (if the object is a LevelPowerReservoir or SlopePowerReservoir): The maximum turbine release is determined based on the selected **Power** method. This calculation is iterative, since the maximum outflow impacts the reservoir tailwater elevation and operating head, which affect the maximum turbine release. The selected **Tailwater** method is used to determine the tailwater elevation. • **No Power Turbine Flow**: The turbine release is determined from a table interpolation in the **Max Flow Through Turbines** table using the average HW as the lookup in the **Reservoir Elevation** column. • **Plant Power Coefficient**: The turbine release is determined from a table interpolation in the **Max Turbine Q** table using the operating head as the lookup in the **Operating Head** column. If the average HW is less than the **Minimum Power Elevation**, the turbine release is zero.	

Evaluation cont.	• **Unit Generator Power**: The turbine release is the sum of the maximum releases for each available turbine, as specified in the **Generators Available and Limit** table. Each turbine's maximum release is determined from a table interpolation in the **Full Generator Flow** table using the • unregulated spill (if an unregulated spillway Method is selected on the object): The maximum unregulated spill is determined from a table interpolation in the **Unregulated Spill Table** using the average HW as the **Pool Elevation.** • regulated spill (if a regulated spillway Method is selected on the object): The maximum regulated spill is the user input **Regulated Spill** at the given timestep or is determined from a table interpolation in the **Regulated Spill Table** using the average HW in the **Pool Elevation** column. Note, if the **MonthlySpill** method is selected, the result of getMaxOutflowGivenHW is the value in the **Maximum Controlled Release** table slot. • bypass (if a bypass spillway Method is selected on the object): The maximum bypass is the user input **Bypass** at the given timestep or zero, if no bypass is input. • All of the individual outflows are then summed to calculate the maximum outflow. • operating head as the lookup in the appropriate unit type's **Head for Type n** column. If the average HW is less than the Minimum Power Elevation, the turbine release is zero. • **Peak Power** and **Peak and Base**: The turbine release is the peak flow over the entire timestep. This is calculated by iterating the selected **Tailwater** method and operating head calculation with a table interpolation in the **Best Generator Flow** table using the operating head as the lookup in the **Head for Type 1** column. • **LCRPowerCalc**: Because this power Method has no turbine release limit, a maximum Outflow cannot be calculated. RiverWare issues an error message and aborts the execution of this rule.
Mathematical Expression	$Outflow_{max} = release_{max} \text{ or } turbine\ release_{max} + unregulated\ spill_{max} + regulated\ spill_{max} + bypass_{max}$
Comments	The **Tailwater Base Value** is added as an automatic dependency of this function. Since the function evaluation depends on this slot, any change to its values, may impact the function result.

Syntax Example:

```
GetMaxOutflowGivenHW(%"Glen Canyon Dam", 3704 "ft", @"June 3, 1983"}
```

Return Example:

```
1283.7047 "cms"
```

65. GetMaxOutflowGivenInflow

This function evaluates to the maximum Outflow from a StorageReservoir, LevelPowerReservoir, or SlopePowerReservoir with the given Inflow at the specified timestep.This function takes into account all side flows, sinks and sources. The inflow argument should be the inflow that would go into the Inflow slot on the reservoir. Since this considers Hydrologic Inflow, the hydrologic inflow value should NOT be included in the inflow argument.

<table>
<tr><th>Description</th><td colspan="2">The maximum combined outflow of a reservoir, including outlet works or turbine release, and any possible regulated, unregulated, and/or bypass spills.</td></tr>
<tr><th>Type</th><td colspan="2">NUMERIC</td></tr>
<tr><th>Arguments</th><td></td><td></td></tr>
<tr><th>1</th><td>OBJECT</td><td>the reservoir object for which to calculate</td></tr>
<tr><th>2</th><td>NUMERIC</td><td>the average inflow over the timestep</td></tr>
<tr><th>3</th><td>DATETIME</td><td>the timestep at which to calculate</td></tr>
<tr><th>Evaluation</th><td colspan="2">This function calls the getMaxOutGivenIn() function on the given reservoir object at the given timestep, and provides it with the average inflow over the timestep. A convergence algorithm is used in this function and is detailed in HERE (Objects.pdf, Section 26.1). The function iterates to convergence by computing the end of timestep storage and pool elevation, the average HW over the timestep, and the following outflows:
<u>release</u> (if the object is a StorageReservoir): The maximum release is determined from a table interpolation in the Max Release table using the average HW as the lookup in the Pool Elevation column.
<u>turbine release</u> (if the object is a LevelPowerReservoir or SlopePowerReservoir): The maximum turbine release is determined based on the selected Power method. This calculation is iterative, since the maximum outflow impacts the reservoir tailwater elevation and operating head, which affect the maximum turbine release. The selected Tailwater method is used to determine the tailwater elevation.</td></tr>
</table>

Evaluation (continued)	• **No Power Turbine Flow**: The turbine release is determined from a table interpolation in the **Max Flow Through Turbines** table using the average HW as the lookup in the **Reservoir Elevation** column. • **Plant Power Coefficient**: The turbine release is determined from a table interpolation in the **Max Turbine Q** table using the operating head as the lookup in the **Operating Head** column. If the average HW is less than the **Minimum Power Elevation**, the turbine release is zero. • **Unit Generator Power**: The turbine release is the sum of the maximum releases for each available turbine, as specified in the **Generators Available and Limit** table. Each turbine's maximum release is determined from a table interpolation in the **Full Generator Flow** table using the operating head as the lookup in the appropriate unit type's **Head for Type n** column. If the average HW is less than the Minimum Power Elevation, the turbine release is zero. • **Peak Power Equation with Off Peak Spill**: The turbine release is the peak release over the entire timestep. This is calculated by iterating the selected **Tailwater** method and operating head calculation with a table interpolation in the **Operating Head vs. Generator Capacity** table. • **Peak Power** and **Peak and Base**: The turbine release is the peak flow over the entire timestep. This is calculated by iterating the selected **Tailwater** method and operating head calculation with a table interpolation in the **Best Generator Flow** table using the operating head as the lookup in the **Head for Type 1** column. • **LCRPowerCalc**: Because this power Method has no turbine release limit, a maximum Outflow cannot be calculated. RiverWare issues an error message and aborts the execution of this rule. • <u>unregulated spill</u> (if an unregulated spillway Method is selected on the object): The maximum unregulated spill is determined from a table interpolation in the **Unregulated Spill Table** using the average HW as the **Pool Elevation**. • <u>regulated spill</u> (if a regulated spillway Method is selected on the object): The maximum regulated spill is the user input **Regulated Spill** at the given timestep or is determined from a table interpolation in the **Regulated Spill Table** using the average HW in the **Pool Elevation** column. Note, if the **MonthlySpill** method is selected, the result of getMaxOutflowGivenInflow is the value in the **Maximum Controlled Release** table slot. • <u>bypass</u> (if a bypass spillway method is selected on the object): The maximum bypass is the user input **Bypass** at the given timestep or zero, if no bypass is input. Once the iteration has converged on an ending storage and pool elevation, all of the individual outflows are summed to calculate the maximum outflow

Mathematical Expression	$Outflow_{max} = release_{max} \text{ or } turbine\ release_{max} + unregulated\ spill_{max} + regulated\ spill_{max} + bypass_{max}$
Comments	This function takes into account the following sources and sinks automatically, and thus they should not be included in the inflow value for Argument 2. • The **Evaporation and Precipitation** category selected Method. • The **Bank Storage** category selected Method. • The **Seepage** category selected Method. • Side inflows including: **Inflow 2** (Slope Power Reservoir only), **Hydrologic Inflow Net**, **Diversion**, **Return Flow**, **Canal Flow**, **Flow FROM Pumped Storage**, and **Flow TO Pumped Storage**. These slots in addition to **Tailwater Base Value** are automatically added as dependencies to the calling rule.

Syntax Example:

```
GetMaxOutflowGivenInflow(%"Hoover Dam", 68651 "cfs", @"June, 1983}
```

Return Example:

```
1283.7047 "cms"
```

66. GetMaxOutflowGivenStorage

This function evaluates to the maximum Outflow from a StorageReservoir, LevelPowerReservoir, or SlopePowerReservoir with the given Storage at the specified timestep.

Description	The maximum combined outflow of a reservoir, including outlet works or turbine release, and any possible regulated, unregulated, and/or bypass spills.	
Type	NUMERIC	
Arguments		
1	OBJECT	the reservoir object for which to calculate
2	NUMERIC	the end of timestep storage
3	DATETIME	the timestep at which to calculate

Evaluation	This function calls the getMaxOutGivenStorage() function on the given reservoir object at the given timestep, and provides it with the end of timestep storage. This ending HW is determined from the ending storage, and is averaged with the previous timestep's ending **Pool Elevation** to yield an average HW over the timestep. The average HW is then used to compute the following outflows: • <u>release</u> (if the object is a StorageReservoir): The maximum release is determined from a table interpolation in the **Max Release** table using the average HW as the lookup in the **Pool Elevation** column. • <u>turbine release</u> (if the object is a LevelPowerReservoir or SlopePowerReservoir): The maximum turbine release is determined based on the selected **Power** method. This calculation is iterative, since the maximum outflow impacts the reservoir tailwater elevation and operating head, which affect the maximum turbine release. The selected **Tailwater** method is used to determine the tailwater elevation. • **No Power Turbine Flow**: The turbine release is determined from a table interpolation in the **Max Flow Through Turbines** table using the average HW as the lookup in the **Reservoir Elevation** column. • **Plant Power Coefficient**: The turbine release is determined from a table interpolation in the **Max Turbine Q** table using the operating head as the lookup in the **Operating Head** column. If the average HW is less than the **Minimum Power Elevation**, the turbine release is zero.

Evaluation (continued)	• **Unit Generator Power**: The turbine release is the sum of the maximum releases for each available turbine, as specified in the **Generators Available and Limit** table. Each turbine's maximum release is determined from a table interpolation in the **Full Generator Flow** table using the operating head as the lookup in the appropriate unit type's **Head for Type n** column. If the average HW is less than the Minimum Power Elevation, the turbine release is zero. • **Peak Power** and **Peak and Base**: The turbine release is the peak flow over the entire timestep. This is calculated by iterating the selected **Tailwater** method and operating head calculation with a table interpolation in the **Best Generator Flow** table using the operating head as the lookup in the **Head for Type 1** column. • **Peak Power Equation with Off Peak Spill**: The turbine release is the peak release over the entire timestep. This is calculated by iterating the selected **Tailwater** method and operating head calculation with a table interpolation in the **Operating Head vs. Generator Capacity** table. • **LCRPowerCalc**: Because this power Method has no turbine release limit, a maximum Outflow cannot be calculated. RiverWare issues an error message and aborts the execution of this rule. • <u>unregulated spill</u> (if an unregulated spillway Method is selected on the object): The maximum unregulated spill is determined from a table interpolation in the **Unregulated Spill Table** using the average HW as the **Pool Elevation.** • <u>regulated spill</u> (if a regulated spillway Method is selected on the object): The maximum regulated spill is the user input **Regulated Spill** at the given timestep or is determined from a table interpolation in the **Regulated Spill Table** using the average HW in the **Pool Elevation** column. Note, if the **MonthlySpill** method is selected, the result of getMaxOutflowGivenStorage is the value in the **Maximum Controlled Release** table slot. • <u>bypass</u> (if a bypass spillway Method is selected on the object): The maximum bypass is the user input **Bypass** at the given timestep or zero, if no bypass is input. • The individual outflows are then summed to calculate the maximum outflow.
Mathematical Expression	$Outflow_{max} = release_{max}\ or\ turbine\ release_{max} + unregulated\ spill_{max} + regulated\ spill_{max} + bypass_{max}$

Comments	The **Tailwater Base Value** is added as an automatic dependency of this function. Since the function evaluation depends on this slot, any change to its values, may impact the function result. This function will issue an error if the “Time Varying Elevation Volume” method is selected, **HERE (Objects.pdf, Section 22.1.23.3)**, and the specified timestep is a modification date on the table.

Syntax Example:

```
GetMaxOutflowGivenStorage(%"Hoover Dam", 17321.400"1000 acre-feet",@"May, 1998"}
```

Return Example:

```
1283.7047 "cms"
```

67. GetMaxReleaseGivenInflow

This function evaluates to the maximum Release, or Turbine Release from a StorageReservoir, LevelPowerReservoir, or SlopePowerReservoir with the given Inflow at the specified timestep.

Description	The maximum release of a reservoir, through outlet works or turbine release.	
Type	NUMERIC	
Arguments		
1	OBJECT	the reservoir object for which to calculate
2	NUMERIC	the average inflow over the timestep
3	DATETIME	the timestep at which to calculate
Evaluation	This function calls the getMaxRelGivenIn() function on the given reservoir object at the given timestep, and provides it with the average inflow over the timestep. A convergence algorithm is used in this function as detailed **HERE (Objects.pdf, Section 26.1)**.The function iterates to convergence by computing the end of timestep storage and pool elevation, the average HW over the timestep, and the release: • <u>release</u> (if the object is a StorageReservoir): The maximum release is determined from a table interpolation in the **Max Release** table using the average HW as the lookup in the **Pool Elevation** column.	

Evaluation (continued)	• turbine release (if the object is a LevelPowerReservoir or SlopePowerReservoir): The maximum turbine release is determined based on the selected **Power** method. This calculation is iterative, since the maximum outflow impacts the reservoir tailwater elevation and operating head, which affect the maximum turbine release. The selected **Tailwater** method is used to determine the tailwater elevation. • **No Power Turbine Flow**: The turbine release is determined from a table interpolation in the **Max Flow Through Turbines** table using the average HW as the lookup in the **Reservoir Elevation** column. • **Plant Power Coefficient**: The turbine release is determined from a table interpolation in the **Max Turbine Q** table using the operating head as the lookup in the **Operating Head** column. If the average HW is less than the **Minimum Power Elevation**, the turbine release is zero. • **Unit Generator Power**: The turbine release is the sum of the maximum releases for each available turbine, as specified in the **Generators Available and Limit** table. Each turbine's maximum release is determined from a table interpolation in the **Full Generator Flow** table using the operating head as the lookup in the appropriate unit type's **Head for Type n** column. If the average HW is less than the Minimum Power Elevation, the turbine release is zero. • **Peak Power** and **Peak and Base**: The turbine release is the peak flow over the entire timestep. This is calculated by iterating the selected **Tailwater** method and operating head calculation with a table interpolation in the **Best Generator Flow** table using the operating head as the lookup in the **Head for Type 1** column. • **Peak Power Equation with Off Peak Spill**: The turbine release is the peak release over the entire timestep. This is calculated by iterating the selected **Tailwater** method and operating head calculation with a table interpolation in the **Operating Head vs. Generator Capacity** table. • **LCRPowerCalc**: Because this power Method has no turbine release limit, a maximum Outflow cannot be calculated. RiverWare issues an error message and aborts the execution of this rule.
Mathematical Expression	$Release_{max} = release_{max} \text{ or } turbine\ release_{max}$

Comments	This function takes into account the following sources and sinks automatically, and thus they should not be included in the inflow value for Argument 2. • The **Evaporation and Precipitation** category selected Method. • The **Bank Storage** category selected Method. • The **Seepage** category selected Method. • Side inflows including: **Inflow 2** (Slope Power Reservoir only), **Hydrologic Inflow Net**, **Diversion**, **Return Flow**, **Canal Flow**, **Flow FROM Pumped Storage**, and **Flow TO Pumped Storage**. These slots in addition to **Tailwater Base Value** are automatically added as dependencies to the calling rule.

Syntax Example:

```
GetMaxReleaseGivenInflow(%"Hoover Dam", 68651 "cfs", @"June, 1983"}
```

Return Example:

```
1283.7047 "cms"
```

68. GetMinSpillGivenInflowRelease

This function evaluates to the minimum spill from a StorageReservoir, LevelPowerReservoir, or SlopePowerReservoir with the given inflow and release at the specified timestep.

Description	The minimum required spill through unregulated and regulated spillways.	
Type	NUMERIC	
Arguments		
1	OBJECT	the reservoir object for which to calculate
2	NUMERIC	the average inflow over the timestep
3	NUMERIC	the average release over the timestep
4	DATETIME	the timestep at which to calculate

Evaluation	This function calls the getMinSpillGivenInflowRelease() function on the given reservoir object at the given timestep, and provides it with the average inflow and release over the timestep. A convergence algorithm is used in this function and is detailed in **HERE (Objects.pdf, Section 26.1)**.The function iterates to convergence by computing the end of timestep storage and pool elevation, the average HW over the timestep, and the spill: • unregulated spill: calculated from the Unregulated Spill Table based on the average Pool Elevation. See the spill method for more details on how this is computed • regulated and bypass spills: assumed to be zero unless input by the user. • outflow: sum of the calculated spill and the release specified in the function. pool elevation: solved for by mass balance using the specified inflow and calculated outflow.
Mathematical Expressions	$spill_{min} = unregulated\ spill + regulated\ spill(\text{ if input}) + bypass\ spill(\text{ if input})$ $outflow_{min} = spill_{min} + release$
Comments	This function takes into account the following sources and sinks automatically, and thus they should not be included in the inflow value for Argument 2. • The **Evaporation and Precipitation** category selected Method. • The **Bank Storage** category selected Method. • The **Seepage** category selected Method. • Side inflows including: **Inflow 2** (Slope Power Reservoir only), **Hydrologic Inflow Net**, **Diversion**, **Return Flow**, **Canal Flow**, **Flow FROM Pumped Storage**, and **Flow TO Pumped Storage.** These slots in addition to the previous timestep's **Pool Elevation** and **Storage** are automatically added as dependencies to the calling rule. Since the function evaluation depends on these slots, any change to their values at the indicated timestep, may impact the function result.

Syntax Example:

```
GetMinSpillGivenInflowRelease(%"Hoover Dam", Hoover Dam.Inflow[],
0.0 "cfs", @"t"}
```

Return Example:

```
1283.7047 "cms"
```

69. GetMonth

This function evaluates to the integer number of the month of the given datetime.

Description	The month number, base 1, in units of "NONE".	
Type	NUMERIC	
Arguments		
1	DATETIME	the datetime of any time within the month
Evaluation	The datetime argument; which may be specified symbolically, is converted into an actual datetime. Then, the number of the month in which the datetime is contained, is determined. This function requires that the specified datetime resolve to at least a partially specified datetime in the “Month day, year” format with the month specified.	
Comments	As elsewhere in RiverWare, 24:00 hours is considered to be the day which is ending, and 00:00 hours is considered to be the day which is just beginning.	

Syntax Example:

```
GetMonth(@"February 23, 1996")
```

Return Example:

```
2.000
```

70. GetMonthAsString

This function evaluates to the string name of the month of the given datetime.

Description	The month name.	
Type	STRING	
Arguments		
1	DATETIME	the datetime of any time within the month
Evaluation	The datetime argument; which may be specified symbolically, is converted into an actual datetime. Then, the name of the month in which the datetime is in, is determined. This function requires that the specified datetime resolve to at least a partially specified datetime in the “Month day, year” format with the month specified.	
Comments	As elsewhere in RiverWare, 24:00 hours is considered to be the day which is ending, and 00:00 hours is considered to be the day which is just beginning.	

Syntax Example:

```
GetMonthAsString(@"February 23, 1996")
```

Return Example:

```
"February"
```

71. GetNumbers

This function evaluates to a sequence of values in a given range with a given offset.

Description	Returns a sequence of values in a given range with a given offset.	
Type	LIST	
Arguments		
1	NUMERIC	the start value
1	NUMERIC	the end value
1	NUMERIC	the offset
Evaluation	The end value and offset are converted into the units of the start value. A list is created whose first item is the start value, the second item is the start value plus the offset, and so on, until the next value to be added to the list would not be in the range defined by the start and end value.	
Comments	The units of all values must be compatible. If the offset is positive and the start value is greater than the end value, the return list is empty; similarly, if the offset is negative and the start value is less than the end value, the return list is empty.	

Syntax Example:

```
GetNumbers(0.0 [cfs], 10 [cms], 1 [cfs])
```

Return Example:

```
{0.0 "cfs", 1.0 "cfs", 2.0 "cfs", ... , 352.0 "cfs", 353.0 "cfs"}
```

72. GetObject

This function looks for an object on the global workspace with a given name and returns that object, if it exists.

Description	Return the object with a given name.
Type	OBJECT

Arguments		
1	STRING	the name of the object for which to search.
Evaluation	The function returns the object with the given name, if it exists.	
Comments	If no object with the given name exists on the global workspace, then the run is aborted with an error.	

Syntax Example:

```
GetObject("Heron Reservoir")
```

Return Example:

```
%"Heron Reservoir"
```

73. GetObjectDebt

This function evaluates to the sum of the debts to all accounting exchanges which may be paid by supplies on the given object. If there are no exchange paybacks on the given object, the debt is zero.

Description	The total debt which can be paid by this object.	
Type	NUMERIC	
Arguments		
1	OBJECT	the object who's debt to calculate
2	DATETIME	the timestep at which to calculate the debt
Evaluation	The function loops over all supplies, on all accounts, on the given object. If a supply is an exchange payback source, the value of its debt slot at the given timestep (if known) is added to the cumulative debt of the object.	
Mathematical Expression	$NUMERIC = \sum_{payback\ source\ supplies_{(object)}} Debt_{(payback,\ timestep)}$	
Comments	If the debt slot of a payback supply at the given timestep is not known, its debt is assumed to be zero. If there are no payback supplies on the given object, the total debt is zero.	

Syntax Example:

```
GetObjectDebt(%"Heron Reservoir", @"t")
```

Return Example:

```
1.823 "m3"
```

74. GetObjectFromSlot

Description	Return a slot's parent object.	
Type	OBJECT	
Arguments		
1	SLOT	The slot whose object is returned.
Comments	It is an error if the slot is not on a Simulation Object or an account (which is on an Object).	

Syntax Example:

```
GetObjectFromSlot($"ResA^Municipal.Inflow")
```

Return Example:

```
%"ResA"
```

75. GetPaybackDebt

This function evaluates to the value of the debt slot of the given exchange payback source at the given timestep.

Description	The debt at a payback source.	
Type	NUMERIC	
Arguments		
1	STRING	the payback source supply whose debt to calculate
2	DATETIME	the timestep at which to calculate the debt
Evaluation	The function begins by verifying that the string argument is an accounting supply, and that this supply is a payback source. If so, the function evaluates to the value of the payback's debt slot at the given timestep.	
Mathematical Expression	$NUMERIC = Debt_{(payback,\ timestep)}$	
Comments	If the string argument is not a valid supply, or the supply is not a payback source, this function aborts the run with an error. If the debt slot of the payback for which this supply is a source does not contain a value at the given timestep, the function evaluates to zero.	

Syntax Example:

```
GetPaybackDebt("Heron SanJuan to WillowAndRioChama SanJuan.Supply",
@"t")
```

Return Example:

```
1.823 "m3"
```

76. GetRowIndex

This function evaluates to the index of the row with the given name in a table slot.

Description	The index of the table slot row whose name matches a string.	
Type	NUMERIC	
Arguments		
1	SLOT	the table slot in which to find a row
2	STRING	the name of the row to match
Evaluation	The labels of the slot rows are compared to the string argument until a match is found.	
Comments	Table row and column indices are zero based and have units of type [NONE]. If the specified slot is not a table slot or the specified string is not the label of a row on the slot, this function aborts the run with an error. If several rows of the table slot match the string argument, this function evaluates to the index of the topmost matching row.	

Syntax Example:

```
GetRowIndex(RiverData.Minimum Flow,"Dolores")
```

Return Example:

```
1.00000
```

77. GetRowIndexByDate

Description	Given a slot with rows indexed by date, this function returns the 0-based index corresponding to a given date.	
Type	NUMERIC	
Arguments		
1	SLOT	the slot in which to find a row
2	DATETIME	date of the row to match
Comments	The value -1 is returned if the given date is not within the date range of the slot. This function is applicable to the following types of slot: • Series Slots • Table Series Slots • Periodic Slot It is considered an error if the slot is not indexed by date (i.e, not one of these types).	

Syntax Example:

```
GetRowIndexByDate(DeepReservoir.Inflow,@"t")
```

Return Example:

```
5.00000
```

78. GetRunCycleIndex

Description	Returns the 1-based index of the current cycle through the timesteps in the run time range, in units of "NONE".	
Type	NUMERIC	
Arguments		
none		
Comments	Rulebased simulations can cycle through the run timesteps more than once each run (see the Rulebased Simulation section **HERE (RulebasedSimulation.pdf, Section 1.7.4)**). This function provides access to the current run cycle, which can be used, for example, within execution constraints to control the cycle on which a rule should execute. If called from outside of a run or when the controller is not Rulebased Simulation or Inline Rulebased Simulation and Accounting, the behavior of this function is undefined.	

Syntax Example:

```
GetRunCycleIndex()
```

Return Example:

```
2.0000
```

79. GetRunIndex

This function evaluates to the number of the model run. It is commonly used within a Multiple Run Management ruleset to determine the run which is currently executing.

Description	The number of the currently executing model run, base 1, in units of "NONE".	
Type	NUMERIC	
Arguments		
none		
Comments	If the current run is not a Multiple Run Management run, this function evaluates to 1. If called from within a pre-MRM run rule of an iterative MRM run, this function evaluates to 1. If called from within a post-run rule of an iterative MRM, this function evaluates to the index of the run which just completed.	

Syntax Example:

```
GetRunIndex()
```

Return Example:

```
3.0000
```

80. GetSelectedUserMethod

Description	Given an object and a user method category name, return the name of the selected method.	
Type	STRING	
Arguments		
1	OBJECT	the simulation object
2	STRING	name of the user method category
Comments	An error is issued if the object or the category name is not found.	

Syntax Example:

```
GetSelectedUserMethod(DeepReservoir,"Power")
```

Return Example:

```
"Peak Power"
```

81. GetSeriesSlots

Description	Returns a list of all of the visible Series Slots on an object.	
Type	LIST{SLOT}	
Arguments		
1	OBJECT	The object whose Series Slots are to be returned.
Comments	If no object with the given name exists on the global workspace, then the run is aborted with an error.	

Syntax Example:

```
GetSeriesSlot(%"My Data Object")
```

Return Example:

```
{$"My Data Object.Series 1", $"My Data Object.Series 2"}
```

82. GetSlot

This function looks for a slot on the global workspace with a given name and returns that slot, if it exists.

Description	Return the slot with a given name.	
Type	SLOT	
Arguments		
1	STRING	the name of the slot for which to search.
Evaluation	The function returns the slot with the given name, if it exists.	
Comments	If no slot with the given name exists on the global workspace, then the run is aborted with an error.	

Syntax Example:

```
GetSlot("Heron Reservoir.Inflow")
GetSlot("Abiquiu Reservoir^RioGrande.Inflow")
```

Return Example:

```
$"Heron Reservoir.Inflow"
$"Abiquiu Reservoir^RioGrande.Inflow"
```

83. GetSlotName

Description	Return the slot name portion of a slot's full name.	
Type	STRING	
Arguments		
1	SLOT	The slot whose name is returned.

Syntax Example:

```
GetSlotName($"ResA^Municipal.Inflow")
```

Return Example:

```
"Inflow"
```

84. GetSlotVals and GetSlotValsNaNToZero

Description	This function evaluates to a list composed of the values of a given series slot within a time range. GetSlotVals can also be used on a periodic slot, while GetSlotValsNaNToZero, cannot.	
Type	LIST{NUMERIC}	
Arguments		
1	SLOT	the series (or periodic slot) whose values to get
2	DATETIME	start datetime
3	DATETIME	end datetime
Evaluation	A list is generated by looking up each value in the given slot, beginning with the start datetime, and ending with the end datetime. All slot values in the range are returned, regardless of the slot data's timestep resolution vis-a-vis that of the run control.	
Mathematical Expression	$\{NUMERIC\} = \{slot_{(start\ datetime)}, \ldots, slot_{(end\ datetime)}\}$	
Comments	If the start datetime or end datetime does not match one of the slot's values', or if the start datetime is after the end datetime, this function aborts the run with an error. For GetSlotVals, if one of the slot values within the desired time range is a NaN, the function exits the rule with an early termination. For GetSlotValsNaNToZero, it converts any NaNs into zero. For periodic slots and GetSlotVals, the dates used are those within the range and falling on a run timestep; the column used is the first (column 0).	

Syntax Example:

```
GetSlotVals(Dolores.Inflow, @"t",
@"September 31,Current Year")
GetSlotValsNaNToZero(Mead.Seepage, @"Start Timestep", @"t")
```

Return Example:

```
{ 1.43"cms", 2.12 "cms" }
```

85. GetSlotValsByCol and GetSlotValsByColNaNToZero

Description	This function evaluates to a list composed of the values in a column of a given Agg Series Slots (or for GetSlotValsByCol, it could be a periodic slot) within a time range.	
Type	LIST{NUMERIC}	
Arguments		
1	SLOT	the agg series slot (or for GetSlotValsByCol, it could be a periodic slot) whose values to get
2	DATETIME	start datetime
3	DATETIME	end datetime
4	NUMERIC	the column (interpreted as a 0-based integral index)
Evaluation	A list is generated by looking up each value in the given column of the slot, beginning with the start datetime, and ending with the end datetime. All slot values in the range are returned, regardless of the slot data's timestep resolution vis-a-vis that of the run control.	
Mathematical Expression	$\{NUMERIC\} = \{slot_{(start\ datetime)}, \ldots, slot_{(end\ datetime)}\}$	
Comments	If the slot is an Agg Series Slot and the start datetime or end datetime does not match one of the slot's values', or if the start datetime is after the end datetime, this function aborts the run with an error. For GetSlotValsByCol, if one of the slot values within the desired time range is a NaN, the function exits the rule with an early termination. For GetSlotValsByColNaNToZero, it converts any NaNs into zero. For periodic slots and GetSlotValsByCol, the dates used are those within the range and falling on a run timestep.	

Syntax Example:

```
GetSlotValsByCol(WaterUser1.Periodic Diversion Request, @"t",
@"September 31,Current Year", 3)
GetSlotValsByColNaNToZero(WaterUser1.IrrigatedAreaByCrop, @"t",
@"September 31,Current Year", 3)
```

Return Example:

```
{ 1.43"cms", 2.12 "cms", 2.54 "cms", 2.2 "cms"}
```

86. GetTableColumnVals & GetTableColumnValsSkipNaN

This function evaluates to a list. Each item of the list is the value of the given table slot, in the given column, from the given start row, to the given end row.

Description	All of the values of a table slot column between two rows.	
Type	LIST{NUMERIC}	
Arguments		
1	SLOT	the table slot whose values to get
2	NUMERIC	column
3	NUMERIC	start row
4	NUMERIC	end row
Evaluation	A list is generated by looking up each value in the given column of the given table slot beginning with the start row, and ending with the end row (inclusive). Rows and columns are numbered beginning with zero.	
Mathematical Expression	$\{NUMERIC\} = \{slot_{(start\ row,\ column)}, \ldots, slot_{(end\ row,\ column)}\}$	
Comments	Units are not required for row and column indices and, if provided, will be ignored. If the column, start row, or end row does not exist in the slot, or if the start row is greater than the end row, this function aborts the run with an error. For the GetTableColumnVals function, if one of the slot values within the desired time range is a NaN, the function exits the rule with an early termination. For the GetTableColumnValsSkipNaN variation of this function, these values are just omitted from the return list.	

Syntax Example:

```
GetTableColumnVals(Chickamauga Data.Power Coeffs, 0, 0, 1)
GetTableColumnValsSkipNaN(Chickamauga Data.Power Coeffs, 0, 0, 1)
```

Return Example:

```
{ 1.43"cms", 2.12 "cms"}
```

87. GetTableRowVals & GetTableRowValsSkipNaN

This function evaluates to a list. Each item of the list is the value of the given table slot, in the given row, from the given start column, to the given end column.

Description	All of the values of a table slot column between two columns.	
Type	LIST{NUMERIC}	
Arguments		
1	SLOT	the table slot whose values to get
2	NUMERIC	row
3	NUMERIC	start column
4	NUMERIC	end column
Evaluation	A list is generated by looking up each value in the given row, of the given table slot beginning with the start column, and ending with the end column (inclusive). Rows and columns are numbered beginning with zero.	
Mathematical Expression	$\{NUMERIC\} = \{slot_{(row,\, start\ column)}, \ldots, slot_{(row,\, end\ column)}\}$	
Comments	Units are not required for row and column indices and, if provided, will be ignored. If the row, start column, or end column do not exist in the slot, or if the start column is greater than the end column, this function aborts the run with an error. For the GetTableRowVals function, if one of the slot values within the desired time range is a NaN, the function exits the rule with an early termination. For the GetTableRowValsSkipNaN variation of this function, these values are just omitted from the return list.	

Syntax Example:

```
GetTableRowVals(Chickamauga Data.Power Coeffs, 0, 0, 1)
GetTableRowValsSkipNaN(Chickamauga Data.Power Coeffs, 0, 0, 1)
```

Return Example:

```
{2.54 "cms", 2.2 "cms"}
```

88. GetTimestep

This function evaluates to the length of the timestep ending on the given datetime.

Description	The length of a timestep, in units of "sec".	
Type	NUMERIC	
Arguments		
1	DATETIME	the datetime of the end of the timestep
Evaluation	The datetime argument; which may be specified symbolically, is converted into an actual datetime. Then, the length of the timestep within which this time is, is determined. This function requires that the specified datetime resolve to a fully specified datetime or an error will occur.	
Comments	If the given datetime corresponds to the moment when one timestep ends and another begins, this function evaluates to the length of the timestep which is ending. As elsewhere in RiverWare, 24:00 hours is considered to be the day which is ending, and 00:00 hours is considered to be the day which is just beginning.	

Syntax Example:

```
GetTimestep(@"February 23, 1996")
```

Return Example:

```
21600 "sec" in a 6-hour model
86400 "sec" in a daily model
2505600.0 "sec" in a monthly model
```

89. GetUpperBound

Description	Returns the upper bound for the given series slot.	
Type	NUMERIC	
Arguments		
1	SLOT	the slot whose bound is to be returned.
Evaluation		
Comments	It is considered an error if the specified slot is not a series slot with a valid upper bound. The upper bound is specified in the slot configuration under the view menu.	

Syntax Example:

```
GetUpperBound("Res A.Power")
```

Return Example:

```
400.0 MW
```

90. GetUpperBoundByCol

Description	Returns the upper bound for the specified column of the given aggregate series slot.	
Type	NUMERIC	
Arguments		
1	SLOT	the agg slot whose bound is to be returned.
2	NUMERIC	the column index (0-based).
Evaluation		
Comments	It is considered an error if the input slot is not an Agg. Series Slot with a valid upper bound for the given column. The upper bound is specified in the slot configuration under the view menu.	

Syntax Example:

```
GetUpperBoundByCol("Res A.Hydro Block Use", 3)
```

Return Example:

```
200.0 MWH
```

91. GetYear

This function evaluates to the year of the given datetime.

Description	The year, expressed as a number in units of "NONE".	
Type	NUMERIC	
Arguments		
1	DATETIME	the datetime
Evaluation	The datetime argument; which may be specified symbolically, is converted into an actual datetime. Then, the year within which this time is, is determined. This function requires that the datetime be at least partially specified with a valid year, E.g. @"Year 2010" or @"Current Year" will work.	
Comments	As elsewhere in RiverWare, 24:00 hours is considered to be the day which is ending, and 00:00 hours is considered to be the day which is just beginning.	

Syntax Example:

```
GetYear(@"24:00 December 31, 1999")
```

Return Example:

```
1999.0
```

92. GetYearAsString

This function evaluates to the year of the given datetime as a string.

Description	The year as a string.	
Type	STRING	
Arguments		
1	DATETIME	the datetime
Evaluation	The datetime argument; which may be specified symbolically, is converted into an actual datetime and the year is returned as a string. This function requires that the datetime be at least partially specified with a valid year, E.g. @"Year 2010" or @"Current Year" will work.	
Comments	As elsewhere in RiverWare, 24:00 hours is considered to be the day which is ending, and 00:00 hours is considered to be the day which is just beginning.	

Syntax Example:

```
GetYearAsString(@"February 23, 1996")
```

Return Example:

```
"1996"
```

93. HasFlag

This function returns whether the specified slot on the given timestep has the specified flag.

Description	Use to test the flag status of a slot at a given datetime.	
Type	BOOLEAN	
Arguments		
1	SLOT	The slot to test
2	DATETIME	The datetime at which to test
3	STRING	The **flag** to test. The flag is specified by string, which can be the single letter version of the flag or the full flag name: O Output I Input R Rules B Best Efficiency M Maximum Capacity i Iterative MRM Z DMI Input U Unit Values T Target TB Target-Begin tb Target-Begin-RiverWare S Surcharge Release G Regulation Discharge D Drift m Method A Account P Propagated
Evaluation	The function returns whether or not the slot has the flag at that date. If the slot does not have a value at the date then the function returns that an invalid value was found (which will cause the calling block to terminate early).	

Comments	An error is returned if the slot is not a Series Slot or if the date is not a legal timestep for that slot. See also the IsInput function, **HERE (IsInput)** which is a more limited function used to test for the “I” or “Z” flag. Note, unlike this function, it returns FALSE when the slot does not have a value.

Syntax Example:

```
HasFlag($”Mead.Outflow”, @”24:00 December 31, 1999", “R”)
HasFlag($”Powell.Storage”, @”24:00 March 31, 2006", “Target”)
```

Return Example:

```
TRUE or FALSE
```

94. HasRuleFiredSuccessfully

Description	Returns whether or not the rule with a given name has successfully executed (with or without effect) during the current timestep of a rulebased simulation.	
Type	BOOLEAN	
Arguments		
1	STRING	The name of the rule. The special string "Current Rule" or “ThisRule” (not case sensitive) is interpreted as a reference to the currently executing rule.
Evaluation	This function returns TRUE if: • the rule finished successfully (i.e., at least one assignment is attempted and none fail), or • the rule finished ineffectively (i.e., the rule is evaluated but the logic within the execution constraint or within the body of the rule decides no assignment is necessary or the rule attempts assignment but priority is junior so no assignment is made). The function returns FALSE if • the rule has not yet fired, or • the rule has fired but terminated early (the rule encountered a NaN in a slot value). Note that, as mentioned above, if the input name is "Current Rule" or “ThisRule”, then this is taken to be a reference to the currently executing rule. Using the structure **NOT** HasRuleFiredSuccessfully("ThisRule") will cause the that rule to only execute successfully once.	
Comments	HasRuleFiredSuccessfully behave as follows for the various RPL Sets: • **Rulebased Simulation Rules**: has the rule fired in the current timestep as described above • **Initialization Rules**: has the rule fired in the current single run. • **Iterative MRM Rules**: has the rule fired in the current MRM iteration (single run). • **Global Functions set**: the behavior for the caller's application. • **Other**: not applicable, aborts with an error message.	

Syntax Example:

```
HasRuleFiredSuccessfully("Smith Flood Control")
```

```
HasRuleFiredSuccessfully("Current Rule")
HasRuleFiredSuccessfully("ThisRule")
```

Return Example:

```
TRUE or FALSE
```

95. HydropowerRelease

This function calculates the additional outflow necessary to satisfy an unmet load (energy requirement) while not causing additional downstream flooding.

Description	Calculates the additional outflow necessary to meet an unmet load, if any exists. The function limits the additional release to ensure that additional downstream flooding does not occur.	
Type	LIST { LIST { SLOT, NUMERIC, OBJECT } }	
Arguments		
1	STRING	the name of the computational subbasin

Evaluation	The function does the following: 1. Prioritizes the power reservoirs in the basin according to the relative Load shortage using the equation below. $Shortage = \frac{Load - Energy}{Load}$ If Load is unknown because the Seasonal Load Time method is selected, it is calculated. The calculated shortage then is a value less than one. The reservoirs with the highest values are first, the lowest reservoirs last. 2. Runs the selected Additional Hydropower Release method on the each power reservoir in the subbasin to calculate the proposed additional release required to meet the load within outflow constraints. 3. In order of priority, hypothetically makes the additional release, visits downstream control points until it reaches a tandem reservoir or the end of the subbasin, whichever comes first. If the release causes (additional) flooding at a control point, it reduces the release until flooding is not caused or the release becomes zero. Resulting releases are then hypothetically routed to downstream control points to make adjustments to their available space hydrographs. A control point's available space hydrograph (in units of flow projected into the future based on the routing coefficients on the control point) is calculated as: $\text{Available Space} = \text{Regulation Discharge} - (\text{Inflow} + \text{forecasted Local Inflow})$ Inflow includes the value of the Inflow slot (at the time of the last dispatch) and the additional inflow resulting from the hypothetical additional releases from upstream reservoirs. It also contains the proposed flood control release hydrograph from the last pass of the flood control method. **Note:** Note, if the "Releases Not Constrained by Flooding" method is selected in the "Hydropower Flooding Exception" category on the control point (**HERE (Objects.pdf, Section 9.1.16.2)**, the control point is ignored, i.e. flooding is allowed at that control point.

Evaluation (cont)	4. Once all power reservoirs have been visited (in priority order), the HydropowerRelease function returns to the calling rule. For each reservoir in the subbasin, three triplets may be returned: • Additional Power Release: This is the additional release to meet the load. If the proposed release is positive, but the additional power release was constrained to zero, the triplet {reservoir.Additional Power Release, 0.0, reservoir} will be returned. If the proposed release is zero, the Additional Power Release (of zero) triplet will not be returned. • Outflow: If the new Outflow is the same as the existing Outflow, no Outflow triplet is returned because the value of the Outflow slot will not change as a direct result of this rule. Otherwise the value for outflow is the existing Outflow plus the additional power release. • Load: the Load triplet is only returned if the “Seasonal Load Time” method (**HERE (Objects.pdf, Section 16.1.33.7)**) is selected on the reservoir. The calling rule is expected to make the assignments of the values to the slots
Comments	This function is dependent on having executed the predefined function FloodControl() on a computational subbasin using the Operating Level Balancing method. This flood control method operation sets up the network topology and necessary data. HydropowerRelease requires that the reservoirs in the subbasin have already dispatched and have valid values in the Regulation Discharge, Outflow, Energy, and Load slots. Use of this function for USACE-SWD: **HERE (USACE_SWD.pdf, Section 3.9.3)**.

Syntax Example:

```
HydropowerRelease("Flood Basin") where "Flood Basin" contains Res1 and Res2.
```

Return Example:

```
{ {"Res1.Additional Hydropower Release", 63.32 "cms", "res1"},
  {"Res1.Outflow", 63.32 "cms", "Res1"},
  {"Res2.Additional Hydropower Release", 23.20 "cms", "Res2"}
  {"Res2.Outflow", 32.02 "cms", "Res2"},
  {"Res2.Load", 2.1 "MWH", "Res2"} }
```

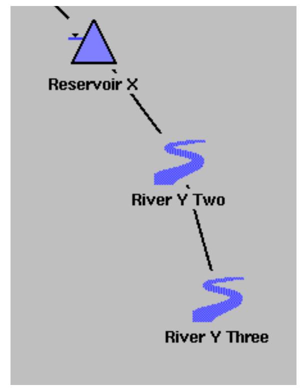
Use Examples:

```
FOREACH (LIST triplet IN HydropowerRelease( "Flood Basin" )) DO
       ( triplet<0> )[] = triplet<1>
ENDFOREACH
```

Hypothetical Simulation Overview

This section presents an overview of a set of functions that allow the user to run a "hypothetical simulation" where:

- a limited number of objects on the workspace are involved
- the simulation has no side-effects, i.e., after simulation the workspace is exactly as before
- the objects involved initially have their current values, except for those values that the user provides as "fixed values" to the hypothetical simulation
- at least one and possibly more values resulting from the hypothetical simulation are available for use within rulebased simulation



Why would you want to do this? Lets consider the following example. Suppose that you would like to maintain a minimum flow of 100 cfs at some point in the River Y. Many miles upstream from this point you can control the outflow from Reservoir X. One question you might ask is: what is the release from X which will lead to the 100 cfs flow at the point of concern? A related but simpler question is: If I release 200 cfs from reservoir X, what will be the flow at the point of concern?

Even the answer to the second question can't be easily predicted; you might have to take into consideration many hydrologic inflows and flow-dependent physical processes like lags, losses, and diversions through different sections of River Y. you might even require that you know the release over some extended period of time in order to be able to determine the flow in the Y at a single time. At any rate, this is exactly the sort of computation performed by the objects in a RiverWare simulation.

On the other hand, answering the first question requires not only knowing the physical consequences of outflow from X, but a search for the release which has the target consequence. The target consequence cannot just be set and allowed to solve upstream because there are routing algorithms can only solve in the downstream direction.

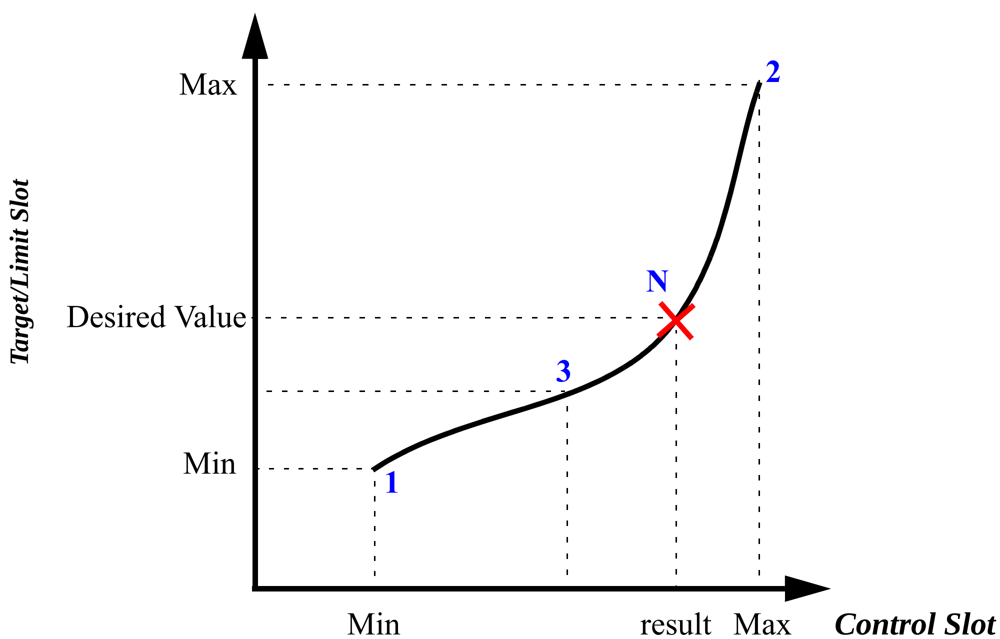
The following functions can be used to do hypothetical simulations:

- **HypSim** - perform a hypothetical simulation with user specified values and returns user-requested result values. Click **HERE (HypSim)** for more information on this function.
- **HypLimitSim** - perform hypothetical simulations iteratively to find a value which, when set on a given slot, will lead to another slot achieving but not exceeding a limiting value. Click **HERE (HypLimitSim)** for more information on this function.
- **HypLimitSimWithStatus** - Same as HypLimitSim but with information on whether a satisfying value was found or not. Click **HERE (HypLimitSimWithStatus)** for more information on this function.
- **HypTargetSim** - perform hypothetical simulations iteratively to find a value which, when set on a given slot, will lead to a desired value on another slot. Click **HERE (HypTargetSim)** for more information on this function.
- **HypTargetSimWithStatus** - Same as HypTargetSim but with information on whether a satisfying value was found or not. Click **HERE (HypTargetSimWithStatus)** for more information on this function.

Each hypothetical simulation function proceeds in the following manner. The first time the function is called, all of the objects and links in the specified subbasin are cloned. This copies each object including all of the slots and values. This cloning is necessary as all subsequent computations are performed on the cloned objects, thereby not affecting the real objects on the workspace. Once the objects are cloned, any values specified in the arguments as "fixed values" are set on the objects. Remember, the function also copied all its data at the time it was cloned, so the fixed values are values that are to be set on the object that are not already there.

Then, the hypothetical simulation performs the computations described for that function. For HypSim, the cloned objects will dispatch to simulate the effects of the fixed values. A list containing the values of the specified slots at the specified datetimes will be returned. The calling rule/function can then use these results in its computation.

HypLimitSim, HypLimitSimWithStatus, HypTargetSim, and HypTargetSimWithStatus perform an iterative solution. For each of these functions, you provide min and max values for a control slot and a target or limit of a downstream slot. The computations then boil down to univariate zero-finding, where each evaluation is a hypothetical simulation with different inputs. The solution is found using the bisection method as shown in the following figure; it simulates using the min control slot value (1), then the max control slot value (2), then bisects between the two(3). It continues bisecting until the tolerance or desired accuracy of the limit/target slot is met and a solution is found (N.) If the result of any try is outside of the range of previous tries, then a warning message is issued saying that the function is non-monotonic. There are either multiple solutions or the function is not well behaved. This typically leads to convergence issues.



Following are some notes to consider about hypothetical simulation and particularly the iterative hypothetical simulations:

- These are hypothetical simulations, not rulebased simulations. You can set fixed values in the arguments but they cannot change once the function is executed.
- Hypothetical simulation does not support accounting or optimization functionality.
- Each iteration within hypothetical simulation is setting the control value to a new value and re-simulating the system. Some object methods may have a dependence on previous solutions that will lead to different results. Also, slot convergence can be an issue. If setting a new value on a slot does not propagate down the system because convergence is too loose, then the target/limit slot value may never be achieved.
- Use the “WithStatus” version of the function if you suspect that the function may not work in all cases but you still need a result. If these functions do not find a result, the closest value is returned along with the status. But, make sure to check the status in the calling rule or function, don’t just use the result blindly. Use the “without status” version if you reasonably expect there to be a solution. Then, if there is a problem with the computation and no solution is found, the run is stopped and a message is posted.
- For the iterative functions, if there are multiple timesteps involved, the control slot is always set to the same value for all timesteps in the hypothetical simulation range.
- These hypothetical simulation functions are expensive in terms of run time. Following are some approaches to limit slow down from these functions:
    - Limit their use as much as possible.
    - If you need to call a function multiple times per timestep with the same arguments, create a helper function with no arguments; functions with no arguments are executed once per timestep and the results are cached for later use.
    - Only include the relevant objects in the subbasin; cloning objects and copying values is time and memory expensive.
    - Only include as many “fixed values” as necessary. If you are only solving for the target slot value at t+2, don’t include fixed values from t through t+7. This will lead to unnecessary dispatching of the cloned objects.

96. HypSim

This function performs a hypothetical simulation with user specified values and returns user-requested result values from the simulation.

Description	Hypothetically simulate a portion of the workspace with user input values and return requested result values.	
Type	LIST {NUMERIC}	
Arguments		

1	STRING	Subbasin name over which to perform the hypothetical simulation. If there is no Subbasin with the given name, the string is taken to be the name of an object and a temporary Subbasin is created containing only that object. An error will be issued if this subbasin contains a Data Object.
2	LIST { LIST { SLOT, NUMERIC, DATETIME } }	Fixed value(s) the user would like to set in each hypothetical simulation. Each item in the list is a list itself containing a slot, the value to set, and the timestep at which to set it.
3	LIST { LIST { SLOT, DATETIME } }	Output(s) to get back from the simulation - each output must specify the slot and the timestep from which to return a value
4	NUMERIC	The minimum number of timesteps before and after the current timestep which might be involved in the simulation. As part of hypothetical simulation RiverWare makes copies of the objects in the subbasin and this input is used to determine how much data should be copied from each object. One can usually set this value to 0 and RiverWare will use a heuristic to determine the range over which to copy data. If this function fails because there was not enough data on some object, then input a higher value.
Evaluation	When function is executed, a hypothetical simulation is initiated where the fixed values are set on the specified slots, the portion of the workspace specified by the Subbasin is simulated, and the values of the output slots are returned as a list.	
Comments	This simulation is hypothetical in that none of the actual values on any objects in the workspace will be modified by hypothetical simulation within a rule. HypSim does not support accounting or optimization functionality. HypSim was originally named HypotheticalSimulation.	

Syntax Example:

```
HypSim("upper basin", {{Navajo.Outflow, 1000 "cfs", @"t"},
{FlamingGorge.Outflow, 1000 "cfs", @"t"}}, {{GreenColorado.Outflow, @"t + 1
Timesteps"}, {SanJuanColorado.Outflow, @"t + 2 Timesteps"}})
```

Return Example:

```
{2.83 "cms", 2.86 "cms"}
```

97. HypLimitSim

This function finds a value which, when set on a given slot, will lead to another slot achieving but not exceeding a limiting value within a given time frame.

Description	Given a control slot and a limit slot, limit date/time, and limit value, this function uses hypothetical simulation (see description of the predefined function HypSim) to find a value *x* such that if the control slot were set to *x* at all timesteps in the range [current date, target date], then the limit slot's maximum value in this range would equal the target value. If the value *x* exceeds the physical constraint for that slot at a particular timestep (max outflow on a reservoir for example), then the constrained value is used instead of the *x* value for that timestep.	
Type	NUMERIC	
Arguments		
1	STRING	The name of the Subbasin over which to perform the hypothetical simulations. This should include the objects on which the control and limit slot exist as well as all other objects necessary to compute the limit slot's value. If there is no Subbasin with the given name, the string is taken to be the name of an object and a temporary Subbasin is created containing only that object. An error will be issued if this subbasin contains a Data Object.
2	SLOT	The control slot, the slot with which you desire to control the target slot's value.
3	NUMERIC	The minimum control slot value. A value less than this is not considered a legal return value.
4	NUMERIC	The maximum control slot value. A value greater than this is not considered a legal return value.
5	LIST { LIST { SLOT, NUMERIC, DATETIME } }	Fixed value(s) the user would like to set in each hypothetical simulation. Each item in the list is a list itself containing a slot, the value to set, and the timestep at which to set it.
6	SLOT	The limit slot, the slot whose value you would like to attain a certain value.
7	DATETIME	The end limit date/time, the end of the time range during which you are concerned with the limit slot's value.
8	NUMERIC	The limit value, the value which you would like the limit slot to achieve but not exceed during the limiting time range.

9	NUMERIC	The tolerance or desired accuracy of the returned value. If the function is successful, it indicates that setting the control slot to the returned value will lead to a maximum limit slot value which differs by no more than the tolerance from the desired limit value.
10	NUMERIC	The maximum number of iterations of hypothetical simulations allowed. If this number is reached without finding an return value within the tolerance, then the function fails.
11	NUMERIC	The minimum number of timesteps before and after the current timestep which might be involved in the simulation. As part of hypothetical simulation RiverWare makes copies of the objects in the subbasin and this input is used to determine how much data should be copied from each object. One can usually set this value to 0 and RiverWare will use a heuristic to determine the range over which to copy data. If this function fails because there was not enough data on some object, then input a higher value.
Evaluation	The implementation of this function uses an iterative algorithm (the bisection algorithm) which performs an hypothetical simulation of the subbasin at each iteration.	
Comments	RiverWare assumes that the target value range (computed using the minimum and maximum control slot values) includes the target value itself. For example, if the control slot minimum of 100 cfs leads to a simulated target value of 100 cfs. the control slot maximum of 1000 cfs leads to a simulated target slot value of 200 cfs, and the target value is 300 cfs, then the function would fail because the target value is not in the range implied by the input control slot minimum and maximum values (100-200 cfs). Mathematically, this is the assumption that limit slot's value is a monotonic function of the control slot's value. See also documentation **HERE**; all comments mentioned there apply here as well.	

Syntax Example:

```
HypLimitSim("upper basin", Navajo.Outflow, 10 "cfs", 1000 "cfs",
{{Navajo.Outflow, 1000 "cfs", @"t"}}, Powell.Inflow, @"t", 100 "cfs", 0.1 "cfs",
10)
```

Return Example:

```
18.34 "cms"
```

98. HypLimitSimWithStatus

This function finds a value which, when set on a given slot, will lead to another slot achieving but not exceeding a limiting within a given time frame. If a value satisfying this criterion is not found, then an attempt is made to find a value that comes close to doing so.

Description	Given a control slot and a limit slot, limit end date/time, and limit value, this function uses hypothetical simulation (see description of the predefined function HypSim) to find a value *x* such that if the control slot were set to *x* at all timesteps in the range [current date, end limit date], then the limit slot's maximum value in this range would equal the target value. If the value *x* exceeds the physical constraint for that slot at a particular timestep (max outflow on a reservoir for example), then the constrained value is used instead of the *x* value for that timestep. A four-item list is returned. The first item in the list is a boolean TRUE value if a satisfying control slot value was found, FALSE otherwise. If the first item is TRUE, then the second item is the satisfying control slot value, otherwise this value is as close as the function could get to finding such a value. The third item is a list of the control slot values used in the solution. These values will all be the same as the second item, except if some of the values were constrained due to physical limitations. The fourth item is a list of the limit slot values that correspond to the control slot values given in the previous list. Note: this function is very similar to HypLimitSim: this only difference is that HypLimitSim fails if it can not find a satisfying control slot value, whereas this function does not fail, rather it still returns a value, along with the indication that this value does not achieve the limit and the additional information discussed above.	
Type	LIST {BOOLEAN, NUMERIC, LIST, LIST}	
Arguments		
1	STRING	The name of the Subbasin over which to perform the hypothetical simulations. This should include the objects on which the control and limit slot exist as well as all other objects necessary to compute the limit slot's value. If there is no Subbasin with the given name, the string is taken to be the name of an object and a temporary Subbasin is created containing only that object. An error will be issued if this subbasin contains a Data Object.
2	SLOT	The control slot, the slot with which you desire to control the target slot's value.

3	NUMERIC	The minimum control slot value. A value less than this is not considered a legal return value.
4	NUMERIC	The maximum control slot value. A value greater than this is not considered a legal return value.
5	LIST { LIST { SLOT, NUMERIC, DATETIME } }	Fixed value(s) the user would like to set in each hypothetical simulation. Each item in the list is a list itself containing a slot, the value to set, and the timestep at which to set it.
6	SLOT	The limit slot, the slot whose value you would like to attain a certain value.
7	DATETIME	The end limit date/time, the end of the time range during which you are concerned with the limit slot's value.
8	NUMERIC	The limit value, the value which you would like the limit slot to achieve but not exceed during the limiting time range.
9	NUMERIC	The tolerance or desired accuracy of the returned value. If the function is successful, it indicates that setting the control slot to the returned value will lead to a maximum limit slot value which differs by no more than the tolerance from the desired limit value.
10	NUMERIC	The maximum number of iterations of hypothetical simulations allowed. If this number is reached without finding an return value within the tolerance, then the function fails.
11	NUMERIC	The minimum number of timesteps before and after the current timestep which might be involved in the simulation. As part of hypothetical simulation RiverWare makes copies of the objects in the subbasin and this input is used to determine how much data should be copied from each object. One can usually set this value to 0 and RiverWare will use a heuristic to determine the range over which to copy data. If this function fails because there was not enough data on some object, then input a higher value.
Evaluation	There are two conditions in which that function will fail but this function will return false as the first item in the return list: • The minimum and maximum control slot values lead to a range of limit values which does not include the input limit value. In this case the value returned is the minimum or maximum control slot value, which ever leads to a limit slot value closest to the input limit value. • The tolerance is not achieved within the iteration limit. In this case, the value returned is the current best guess. If "Hypothetical Simulation" diagnostics are turned on, then if HypLimitSimWithStatus can not find a satisfying control value, a diagnostic will be posted describing why it failed to do so.	

Comments	See also documentation for HypLimitSim for more details; the differences between these two functions are how problems are dealt with, this function is more flexible (as described in the Evaluation section). See also documentation **HERE**; all comments mentioned there apply here as well.

Syntax Example:

```
HypLimitSimWithStatus("upper basin", Navajo.Outflow, 10 "cfs", 1000 "cfs",
{{Navajo.Outflow, 1000 "cfs", @"t"}, {FlamingGorge.Outflow, 1000 "cfs", @"t"}},
Powell.Inflow, @"t", 100 "cfs", 0.1 "cfs", 10)
```

Return Example:

```
{TRUE, 2.3 "cms", {2.3 "cms", 2.28 "cms"}, {2.83 "cms", 2.83 "cms"}}
```

99. HypTargetSim

This function finds a value which, when set on a given slot, will lead to a desired value on another slot.

Description	Given a control slot and a target slot, target date/time, and target value, this function uses hypothetical simulation (see description of the predefined function HypSim) to find a value *x* such that if the control slot were set to *x* at all timesteps in the range [current date, target date], then the target slot's value would equal the target value. If the value *x* exceeds the physical constraint for that slot at a particular timestep (max outflow on a reservoir for example), then the constrained value is used instead of the *x* value for that timestep.	
Type	NUMERIC	
Arguments		
1	STRING	The name of the Subbasin over which to perform the hypothetical simulations. This should include the objects on which the control and target slot exist as well as all other objects necessary to compute the target slot's value. If there is no Subbasin with the given name, the string is taken to be the name of an object and a temporary Subbasin is created containing only that object. An error will be issued if this subbasin contains a Data Object.
2	SLOT	The control slot, the slot with which you desire to control the target slot's value.
3	NUMERIC	The minimum control slot value. A value less than this is not considered a legal return value.

4	NUMERIC	The maximum control slot value. A value greater than this is not considered a legal return value.
5	LIST { LIST { SLOT, NUMERIC, DATETIME } }	Fixed value(s) the user would like to set in each hypothetical simulation. Each item in the list is a list itself containing a slot, the value to set, and the timestep at which to set it.
6	SLOT	The target slot, the slot whose value you would like to attain a certain value.
7	DATETIME	The target date/time, the timestep at which you would like the target slot to attain the desired value.
8	NUMERIC	The target value, the value which you would like the target slot to achieve at the target date/time.
9	NUMERIC	The tolerance or desired accuracy of the returned value. If the function is successful, it indicates that setting the control slot to the returned value will lead to a value which differs by no more than the tolerance from the desired target value.
10	NUMERIC	The maximum number of iterations of hypothetical simulations allowed. If this number is reached without finding an return value within the tolerance, then the function fails.
11	NUMERIC	The minimum number of timesteps before and after the current timestep which might be involved in the simulation. As part of hypothetical simulation RiverWare makes copies of the objects in the subbasin and this input is used to determine how much data should be copied from each object. One can usually set this value to 0 and RiverWare will use a heuristic to determine the range over which to copy data. If this function fails because there was not enough data on some object, then input a higher value.
Evaluation	In a sense, HypTargetSim is the inverse of HypSim. In particular, *HypTargetSim(basin, control, min, max, targetSlot, targetDate, targetValue, tolerance, maxIterations) = x* implies that *HypSim(basin,{{targetSlot, t, x}, ..., {targetSlot, targetDate, x}}, {{controlSlot, targetDate}} = targetValue* The implementation of this function uses an iterative algorithm (the bisection algorithm) which performs an hypothetical simulation of the subbasin at each iteration.	

Comments	RiverWare assumes that the target value range (computed using the minimum and maximum control slot values) includes the target value itself. For example, if the control slot minimum of 100 cfs leads to a simulated target value of 100 cfs. the control slot maximum of 1000 cfs leads to a simulated target slot value of 200 cfs, and the target value is 300 cfs, then the function would fail because the target value is not in the range implied by the input control slot minimum and maximum values (100-200 cfs). Mathematically, this is the assumption that target slot's value is a monotonic function of the control slot's value. See also documentation **HERE**; all comments mentioned there apply here as well. HypTargetSim was originally named HypotheticalTargetSimulation.

Syntax Example:

```
HypTargetSim("upper basin", Navajo.Outflow, 10 "cfs", 1000 "cfs",
{{Navajo.Outflow, 1000 "cfs", @"t"}}, Powell.Inflow, @"t", 100 "cfs", 0.1 "cfs",
10)
```

Return Example:

```
23.4 "cms"
```

100. HypTargetSimWithStatus

This function finds a value which, when set on a given slot, will lead to a desired value on another slot. If a value satisfying this criterion is not found, then an attempt is made to find a value that comes close to doing so.

Description	Given a control slot and a target slot, target date/time, and target value, this function uses hypothetical simulation (see description of the predefined function HypSim) to find a value *x* such that if the control slot were set to *x* at all timesteps in the range [current date, target date], then the target slot's value would equal the target value. If the value *x* exceeds the physical constraint for that slot at a particular timestep (max outflow on a reservoir for example), then the constrained value is used instead of the *x* value for that timestep. A three-item list is returned. The first item in the list is a boolean TRUE value if a satisfying control slot value was found, FALSE otherwise. If the first item is TRUE, then the second item is the satisfying control slot value, otherwise this value is as close as the function could get to finding such a value. The third item is a list of the control slot values used in the solution. These values will all be the same as the second item, except if some of the values were constrained due to physical limitations. Note: this function is very similar to HypTargetSim: this only difference is that HypTargetSim fails if it can not find a satisfying control slot value, whereas this function does not fail, rather it still returns a value, along with the indication that this value does not achieve the target and the list of control slot values.	
Type	LIST {BOOLEAN, NUMERIC, LIST}	
Arguments		
1	STRING	The name of the Subbasin over which to perform the hypothetical simulations. This should include the objects on which the control and target slot exist as well as all other objects necessary to compute the target slot's value. If there is no Subbasin with the given name, the string is taken to be the name of an object and a temporary Subbasin is created containing only that object. An error will be issued if this subbasin contains a Data Object.
2	SLOT	The control slot, the slot with which you desire to control the target slot's value.
3	NUMERIC	The minimum control slot value. A value less than this is not considered a legal return value.
4	NUMERIC	The maximum control slot value. A value greater than this is not considered a legal return value.

5	LIST { LIST { SLOT, NUMERIC, DATETIME } }	Fixed value(s) the user would like to set in each hypothetical simulation. Each item in the list is a list itself containing a slot, the value to set, and the timestep at which to set it.
6	SLOT	The target slot, the slot whose value you would like to attain a certain value.
7	DATETIME	The target date/time, the timestep at which you would like the target slot to attain the desired value.
8	NUMERIC	The target value, the value which you would like the target slot to achieve at the target date/time.
9	NUMERIC	The tolerance or desired accuracy of the returned value. If the function is successful, it indicates that setting the control slot to the returned value will lead to a value which differs by no more than the tolerance from the desired target value.
10	NUMERIC	The maximum number of iterations of hypothetical simulations allowed. If this number is reached without finding an return value within the tolerance, then the function fails.
11	NUMERIC	The minimum number of timesteps before and after the current timestep which might be involved in the simulation. As part of hypothetical simulation RiverWare makes copies of the objects in the subbasin and this input is used to determine how much data should be copied from each object. One can usually set this value to 0 and RiverWare will use a heuristic to determine the range over which to copy data. If this function fails because there was not enough data on some object, then input a higher value.
Evaluation	There are two conditions in which that function will fail but this function will return false as the first item in the return list: • The minimum and maximum control slot values lead to a range of target values which does not include the input target value. In this case the value returned is the minimum or maximum control slot value, which ever leads to a target value closest to the input target value. • The tolerance is not achieved within the iteration limit. In this case, the value returned is the current best guess. If "Hypothetical Simulation" diagnostics are turned on, then if HypTargetSimWithStatus can not find a satisfying control value, a diagnostic will be posted describing why it failed to do so.	

Comments	See also documentation for HypTargetSim for more details; the differences between these two functions are how problems are dealt with, this function is more flexible (as described in the Evaluation section). See also documentation **HERE**; all comments mentioned there apply here as well.

Syntax Example:

```
HypTargetSimWithStatus("upper basin", Navajo.Outflow, 10 "cfs", 1000 "cfs",
{{Navajo.Outflow, 1000 "cfs", @"t"}, {FlamingGorge.Outflow, 1000 "cfs", @"t"}},
Powell.Inflow, @"t", 100 "cfs", 0.1 "cfs", 10)
```

Return Example:

```
{TRUE, 2.3 "cms", {2.3 "cms", 2.28 "cms"}}
```

101. IntegerToString

Description	Returns a string representation of a numeric value interpreted as an integer.	
Type	STRING	
Arguments		
1	NUMERIC	a value
Evaluation	Given a numeric value, IntegerToString returns a string representation of that value rounded to the nearest integer and with the units removed.	
Comments	Note, this function uses the RPL units of the specified NUMERIC. In the example, below, the RPL units are cfs as it is a literal value. But, if you reference a slot value, it will return the string using the relevant RPL units, which are often, but not always, internal units, cms, m, etc... For more flexibility with units, see “IntegerWithUnitsToString,” **HERE**.	

Syntax Example:

```
IntegerToString(123.456 “cfs”)
```

Return Example:

```
“123”
```

102. IntegerWithUnitsToString

Description	Returns a string representation of a numeric value interpreted as an integer after the specified value is converted to the given units.	
Type	STRING	
Arguments		
1	NUMERIC	a value to convert and truncate
2	NUMERIC	a value whose units are those to which the first value should be converted
Evaluation	Given a numeric value, IntegerWithUnitsToString converts that value to the display units of a second input value, and then returns a string representation of the converted value with the fractional part of the value and the units removed.	
Comments	It is an error if the two numeric input values do not have compatible units (i.e., are not of the same unit type). Note, for the second argument, only its display units are involved in the computation; the scalar portion of this value is ignored. This value should typically be specified as a literal value, not the result of computation or lookup on a slot, so that the units of this value will be known with certainty.	

Syntax Example:

```
IntegerWithUnitsToString(Res.Inflow[], 1.0 “cfs”)
```

Return Example:

```
“123”, assuming that the current value on the slot Res.Inflow is 123.9 cfs.
```

103. IsControllerRBS

Description	Returns true if and only if the current controller is Rulebased Simulation (RBS) or Inline Rulebased Simulation and Accounting.	
Type	BOOLEAN	
Arguments		
Evaluation		
Comments		

Syntax Example:

```
IsControllerRBS()
```

Return Example:

```
TRUE or FALSE
```

104. IsEven

This function returns whether or not a given number is even.

Description	Returns true if and only if the input value (rounded down) is even.	
Type	BOOLEAN	
Arguments		
1	NUMERIC	a value
2	NUMERIC	the units in which to determine evenness.
Evaluation	Converts the value into the desired units, rounds down to the nearest integer, then returns whether or not this value is even.	
Comments		

Syntax Example:

```
IsEven(Puddle.Inflow[], 1.0 [cfs])
```

Return Example:

```
TRUE or FALSE
```

105. IsInput

This function evaluates whether there is an input value on the given slot at the given datetime.

Description	The input status of a slot at a given datetime.	
Type	BOOLEAN	
Arguments		
1	SLOT	the series slot
2	DATETIME	the datetime
Evaluation	The datetime argument; which may be specified symbolically, is converted into an actual datetime. Then, the flag of the value in the series slot at that time is compared with the user input flags. If the flag is an “I” or “Z”, true is returned. An “i” is considered a user input within an iterative MRM run, but not a user input within a single run or outside of a run. The “R” flag is not considered an input. Also, IsInput returns false if the slot[datetime] is NaN. But, this function does NOT terminate the executing rule if the value at the given datetime is a NaN. See also the similar yet more general function: **HERE (HasFlag)**.	

Syntax Example:

```
IsInput(SanJuanData.Mexico Call, @"t")
```

Return Example:

```
TRUE or FALSE
```

106. IsOdd

This function returns whether or not a given number is odd.

Description	Returns true if and only if the input value (rounded down) is odd.	
Type	BOOLEAN	
Arguments		
1	NUMERIC	a value
2	NUMERIC	the units in which to determine oddness.
Evaluation	Converts the value into the desired units, rounds down to the nearest integer, then returns whether or not this value is odd.	

Comments	

Syntax Example:

```
IsOdd(Puddle.Inflow[], 1.0 [cfs])
```

Return Example:

```
TRUE or FALSE
```

107. LeapYear

This function evaluates whether a given date occurs in a leap year.

Description	Whether or not a given datetime is in a leap year (containing 366 days instead of 365).	
Type	BOOLEAN	
Arguments		
1	DATETIME	the datetime
Evaluation	The datetime argument; which may be specified symbolically, is converted into an actual datetime. If the date of this year is a leap year, true is returned, otherwise false.	
Comments		

Syntax Example:

```
LeapYear(@"t")
```

Return Example:

```
TRUE or FALSE
```

108. ListDownstreamObjects

Creates a list of linked downstream objects that are between two specified objects.

Description	This function evaluates to a list of linked downstream objects that are between two specified objects, inclusive.	
Type	LIST {OBJECT}	
Arguments		

1	OBJECT	The upstream object at which to start the search
2	OBJECT	The downstream object at which to finish the search
Evaluation	ListDownstreamObjects searches links downstream from the first object until it finds the second object. It returns a list of all objects that it finds, inclusive of the specified objects.	
Comments	Notes / Limitations to this function: • To find the next downstream object, the function traverses links from main channel outflow slots. These slots typically include Outflows from each object; the table below shows the main channel outflow slot for each object • If a Bifurcation or Pipe Junction has more than one outflow link, an error is issued. • Object 1 must be upstream of Object 2. That is, if Object 2 is not found in the search, an error is issued. • For Aggregate Reaches and Agg Distribution canals, only the elements are returned, not the aggregate (but each element includes the agg name, E.g. %"AggReach:Element1").	

Object Type		Main channel outflow slot(s)
Agg Diversion Site		Total Outflow
Agg Distribution Canal		Total Return Flow
Agg Reach, Confluence, Control Point, Distribution Canal, Groundwater,	Inline Power, Inline Pump, Pipeline, Reach, Reservoir	Outflow
Bifurcation		Outflow1, Outflow2
Diversion Object,		NONE
Pipe Junction		Flow 1, Flow 2
Stream Gage		Gage Outflow
Water User		NONE

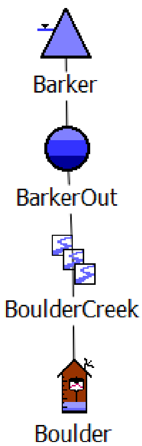
Syntax Example:

```
ListDownstreamObjects(%"Barker", %"Boulder")
```

Return Example:

```
{%"Barker",
 %"BarkerOut",
 %"BoulderCreek:Routing",
```

```
%”BoulderCreek:Locals”,
%”Boulder”}
```



109. ListSubbasin

This function evaluates to a list of the objects in a given subbasin.

Description	The objects in the given subbasin.	
Type	LIST {OBJECT}	
Arguments		
1	STRING	the name of the subbasin
Evaluation	The list of subbasins in the model is searched for a match to the given string. Then, a list is generated from the member objects of that subbasin.	
Comments	Both predefined and user defined subbasins may be listed. Member objects are listed in the order in which they appear in the **Edit Subbasins** dialog. If there is no subbasin with the given name in the model, this function aborts the run with an error.	

Syntax Example:

```
ListSubbasin("LevelPowerReservoir")
ListSubbasin("Colorado above GJ")
```

Return Example:

```
{%"Mead", %"Powell", %"Havasu"}
```

110. Ln

This function evaluates to the natural logarithm of the given number.

Description	The natural logarithm of a number.	
Type	NUMERIC	
Arguments		
1	NUMERIC	the value
2	NUMERIC	the units in which to compute the logarithm
Evaluation	Converts the value into the desired units, and then computes the natural logarithm of this value. The solution is this number in the units of the converted value.	
Comments	The natural logarithm may only be evaluated for values greater than zero. This function aborts the run with an error, if it is evaluated with a value less than or equal to zero. The two arguments must have compatible units, otherwise the run is aborted with an error. Note that this function does not use the scalar portion of the units argument.	

Syntax Example:

```
Ln(1.0 "cfs", 0.0 "cms")
```

Return Example:

```
-3.56429837 "cms"
```

111. Log

This function evaluates to the base10 logarithm of the given number.

Description	The base10 logarithm of a number.	
Type	NUMERIC	
Arguments		
1	NUMERIC	the value
2	NUMERIC	the units in which to compute the logarithm

Evaluation	Converts the value into the desired units, and then computes the base 10 logarithm of this value. The solution is this number in the units of the converted value.
Comments	The base10 logarithm may only be evaluated for values greater than zero. This function aborts the run with an error, if it is evaluated with a value less than or equal to zero. The two arguments must have compatible units, otherwise the run is aborted with an error. Note that this function does not use the scalar portion of the units argument.

Syntax Example:

```
Log(100.0 "cfs", 0.0 "cms")
```

Return Example:

```
0.45204489 "cms"
```

112. Max

This function evaluates to the greater of its two arguments.

Description	The greater value.	
Type	NUMERIC	
Arguments		
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.	

Syntax Example:

```
Max(100"cfs", 10"cms") = 10.000 "cms"
Max(Powell.Storage[], Mead.Storage[]) = 1233481837.55 "m3"
```

113. MaxItem

This function evaluates to the greatest number in a given list.

Description	The greatest value.	
Type	NUMERIC	
Arguments		
1	LIST	the list of values
Evaluation	The numbers in the input list are converted to a common unit and compared. The greatest number is returned.	
Comments	If the list is empty, one of the items is not numeric, or they do not all have compatible types, this function aborts the run with an error.	

Syntax Example:

```
MaxItem({100"cfs", 10"cms", 50 [cfs]})
```

Return Example:

```
10.00 "cms"
```

114. MaxObjectsAggregatedOverTime

This function returns a single numeric value which is the largest of several objects' aggregated slot values. The objects' slot values may be aggregated as a **SUM**, **AVG**, **MIN**, or **MAX** over a specified time range.

Description	The largest of several object's values, each of which is the result of aggregating a slot's values over time.	
Type	LIST	
Arguments		
1	STRING	subbasin name
2	STRING	slot name
3	STRING	aggregation function ("SUM", "AVG", "MIN", or "MAX")
4	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
5	BOOLEAN	time conversion option ("TRUE" or "FALSE")
6	DATETIME	start date

7	DATETIME	end date
Evaluation	A list of slots is generated by searching all of the objects in the subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values to be aggregated are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, each slot's values are aggregated according to the aggregation function argument over the time range of the datetime arguments. During each of these slot aggregations, any values which do not satisfy the aggregation filter argument are ignored. Finally, the largest of the objects' aggregated slot values is determined. This value is returned as the first value in a list. If there is a date/time associated with this value, it is returned as the second value in the list. This will be the case if the "MIN" or "MAX" aggregation function is specified for the fifth argument.	
Mathematical Expression	$Max(\forall_{(obj\text{ in }subbasin)}[\forall_{(t\text{ from }start\text{ to }end)}[AggFunction_{(obj)}(obj.slotname)]])$	
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, this function aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.	

Syntax Example:

```
MaxObjectsAggregatedOverTime("upper basin", "Inflow", "MAX","ALL", TRUE
@"October, Previous Year",
@"September, Current Year")
```

Return Example:

```
324.3 "cms"
```

115. MaxObjectsAtEachTimestep

This function evaluates to a list. Each item of the list is a list comprised of the datetime at which the largest value was determined, and the value itself.

Description	The largest of several object's slot values for each timestep in a range.	
Type	LIST{LIST{DATETIME, NUMERIC}}	
Arguments		
1	STRING	Subbasin name

2	STRING	slot name
3	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
4	BOOLEAN	time conversion option ("TRUE" or "FALSE")
5	DATETIME	start date
6	DATETIME	end date
Evaluation	A list of slots is generated by searching all of the objects in the Subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values whose maximum to find are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, all of the object's slot values are compared, yielding one maximum value for each timestep in the time range of the datetime arguments. The function returns a list of two items, where the first and second items of the inner lists are the datetime and the largest value, respectively.	
Mathematical Expression	$\forall_{(t \text{ from } start \text{ to } end)}[\{t, Max(\forall_{(obj \text{ in } subbasin)}[obj.slotname])\}]$	
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, RiverWare aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.	

Syntax Example:

```
MaxObjectsAtEachTimestep("upper basin", "Storage", "ALL", FALSE
@"October, Previous Year",
@"September, Current Year")
```

Return Example:
For a monthly model, the above function would return something like:

```
{ { 24:00 October 31, 1996, 1233232.2 "m3" },
  { 24:00 November 30, 1996, 1067478.3 "m3" },
  ....
  { 24:00 September 30, 1997, 1563456.7 "m3" } }
```

116. MaxTimestepsAggregatedOverObjects

This function evaluates to a single numeric value, which is the largest value resulting from aggregating several objects' slot values at each timestep.

Description	Largest over a timeseries of values, each of which is the result of aggregating several objects' slot values.	
Type	LIST	
Arguments		
1	STRING	Subbasin name
2	STRING	slot name
3	STRING	aggregation function ("SUM", "AVG", "MIN", or "MAX")
4	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
5	BOOLEAN	time conversion option ("TRUE" or "FALSE")
6	DATETIME	start datetime
7	DATETIME	end datetime
Evaluation	A list of slots is generated by searching all of the objects in the Subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values to be aggregated are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, all of the objects' slot values are aggregated according to the aggregation function argument for each timestep in the time range of the datetime arguments. During each of these slot aggregations, any values which do not satisfy the aggregation filter argument are ignored. Finally, the largest value in the timeseries of object aggregated slot values is determined. This value is returned as the second value in a list. The first item is the date/time associated with this value. If there is an object associated with the value, it is returned as the third value in the list. This will be the case if the "MIN" or "MAX" aggregation function is specified for the third argument.	
Mathematical Expression	$Max(\forall_{(t \text{ from } start \text{ to } end)}[\forall_{(obj \text{ in } subbasin)}[AggFunction_{(t)}(obj.slotname)]])$	

Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, RiverWare aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.

Syntax Example:

```
MaxTimestepsAggregatedOverObjects("upper basin","Storage","MAX","ALL",
FALSE, @"October, Previous Year",
@"September, Current Year")
```

Return Example:

```
{@"March 31, 2004", 2342343232.32"m3", %"Res1"}
```

117. MaxTimestepsForEachObject

This function evaluates to a list. Each item of the list is a list comprised of the object name, and the largest value of the slot on that object for the time range specified.

Description	Largest value in a slot over a time range, for each object in a subbasin.	
Type	LIST {LIST {OBJECT, NUMERIC}}	
Arguments		
1	STRING	Subbasin name
2	STRING	slot name
3	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
4	BOOLEAN	time conversion option ("TRUE" or "FALSE")
5	DATETIME	start datetime
6	DATETIME	end datetime
Evaluation	A list of slots is generated by searching all of the objects in the Subbasin argument for slots which match the slot name argument. For each object, the largest slot value over every timestep in the range of the datetime arguments is determined. Any values which do not satisfy the aggregation filter argument are ignored during the calculation. If the time conversion option argument is TRUE, and the values to be aggregated are of the FLOW unit type, the values are first multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME.	

Mathematical Expression	$\forall_{(obj \text{ in } subbasin)}[\{obj, Max(\forall_{(t \text{ from } start \text{ to } end)}[obj.slotname])\}]$
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, this function aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, this function also aborts RiverWare with an error.

Syntax Example:

```
MaxTimestepsForEachObject("upper basin", "Inflow", "ALL", TRUE,
@"October, Previous Year",
@"September, Current Year")
```

Return Example:

```
{ {%"Res1", 12.23"cms"}, {%"Reach2", 4.92 "cms"}, {%"Res2", 23.2 "cms"} }
```

118. MeetLowFlowRequirement

This function computes the necessary Low Flow Releases from contributing reservoirs to meet a low flow requirement at a specified control point.

Description	Computes the Low Flow Release from each reservoir so that the low flow requirement at the specified control point is met.	
Type	LIST{ LIST{Slot, Value, Object}}	
Arguments		
1	STRING	The subbasin used to perform the calculations
2	OBJECT	The control point whose low flow requirement needs to be met

Evaluation	Returns a LIST of LISTs with the inner list containing a triplet. Each triplet is a slot (at index zero), the value to set on that slot (at index one), and the object of the slot (at index two). The function returns each Low Flow Release slot and the value to set on that slot. Also, the Outflow slot for each reservoir is returned with the new outflow value (Outflow plus Low Flow Release). The function computes the release for each contributing reservoir. The contributing reservoirs are specified in a slot called Low Flow Reservoirs on the Control Point. The rule executes as follows: First, the specified low flow reservoirs are sorted in descending order according to Operating Level. Reservoirs that are below the bottom of the conservation pool are excluded. Next, each reservoir (beginning with the most full reservoir) makes a release until the requirement (in the Computed Low Flow Requirement slot on the Control Point) is met, the Maximum Low Flow Delivery Rate (on the reservoir) is met, or the reservoir reaches the bottom of the conservation pool (whichever value is lowest). In addition, as each reservoir is making releases, the function calls the getMaxOutflowGivenInflow function to calculate the maximum flow that can be released from the reservoir. If the calculated low flow release is greater than this max, the release is reduced to the max. The Low Flow Release and updated Outflow value (limited by max constraints) for each reservoir is returned by the RPL function to the calling rule. The rule sets these slots using the syntax given below. After the rule executes, the system solves and routes the values downstream. NOTE: Each time this rule function is evaluated, it adds to the existing value in the Low Flow Release slot on reservoir objects. This is because each reservoir may contribute to the low flow requirement of more than one control point. So if the user wants to recompute all the low flow releases, they must all be reset to zero. In other words, this function is designed to execute once for each control point, adding to the Low Flow Releases made for previous control points.

Comments	A rule needs to be created for each Control Point that has a low flow requirement. Each rule will call the MeetLowFlowRequirement function for the specified Control Point. After simulating the new releases, the next rule will be executed for the next low flow Control Point. The specified subbasin needs to include all the relevant objects (reservoirs, control points). Within the function execution, no routing of low flow releases is considered between the reservoir and the control point. But, during the simulation after the rule finishes, routing is considered. As a result, the water released may not make it to the control point on a single timestep. Each reservoir must have the Conservation and Flood Pools method selected in the Operating Levels category. The reservoirs specified in the Low Flow Reservoirs slot on each control point MUST be upstream of the control point. RiverWare does not check this and the results will be incorrect if the user does not enforce this. Also, no tandem operations are considered by this RPL function, i.e. the function assumes that reservoir releases can travel directly to a control point without passing through another reservoir. Use of this function for USACE-SWD: **HERE (USACE_SWD.pdf, Section 3.7)**.

Syntax Example:

```
MeetLowFlowRequirement("Basin") where "Basin" contains Res1, Res2, and CP1.
```

Return Example:

```
{ {"Res1.Low Flow Release", 9.75 "cms", "Res1"},
  {"Res1.Outflow", 9.75 "cms", "Res1"},
  {"Res2.Low Flow Release", 2.35 "cms", "Res2"}
  {"Res2.Outflow", 3.25 "cms", "Res2"} }
```

Use Examples:

```
FOR EACH ( LIST result IN MeetLowFlowRequirement("Basin", %"CP1")) DO
     ( result<0> )[] = result<1>
END FOR EACH
```

119. Min

This function evaluates to the smaller of its two arguments.

Description	The lesser value.
Type	NUMERIC

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

Syntax Example:

```
Min(100"cfs", 10"cms") returns 100 "cfs"
Min(Powell.Storage[], Mead.Storage[]) returns 12236343.55 "m3"
```

120. MinItem

This function evaluates to the least number in a given list.

Description	The smallest value.	
Type	NUMERIC	
Arguments		
1	LIST	the list of values
Evaluation	The numbers in the input list are converted to a common unit and compared. The smallest number is returned.	
Comments	If the list is empty, one of the items is not numeric, or they do not all have compatible types, this function aborts the run with an error.	

Syntax Example:

```
MinItem({100"cfs", 10"cms", 50 [cfs]})
```

Return Example:

```
50.0 [cfs]
```

121. MinObjectsAggregatedOverTime

This function returns a single numeric value which is the smallest of several objects' aggregated slot values. The objects' slot values may be aggregated as a **SUM**, **AVG**, **MIN**, or **MAX** over a specified time range.

Description	The smallest of several object's values, each of which is the result of aggregating a slot's values over time.	
Type	LIST{NUMERIC, DATETIME}	
Arguments		
1	STRING	subbasin name
2	STRING	slot name
3	STRING	aggregation function ("SUM", "AVG", "MIN", or "MAX")
4	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
5	BOOLEAN	time conversion option ("TRUE" or "FALSE")
6	DATETIME	start date
7	DATETIME	end date
Evaluation	A list of slots is generated by searching all of the objects in the subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values to be aggregated are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, each slot's values are aggregated according to the aggregation function argument over the time range of the datetime arguments. During each of these slot aggregations, any values which do not satisfy the aggregation filter argument are ignored. Finally, the smallest of the objects' aggregated slot values is determined. This value is returned as the first value in a list. If there is a date/time associated with this value, it is returned as the second value in the list. This will be the case if the "MIN" or "MAX" aggregation function is specified for the third argument.	
Mathematical Expression	$Min(\forall_{(obj \text{ in } subbasin)}[\forall_{(t \text{ from } start \text{ to } end)}[AggFunction_{(obj)}(obj.slotname)]])$	

Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, this function aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.

Syntax Example:

```
MinObjectsAggregatedOverTime("upper basin", "Inflow", "MAX","ALL", TRUE
@"October, Previous Year",
@"September, Current Year")
```

Return Example:

```
{0.24 "cms", @"February 3, 2003"}
```

122. MinObjectsAtEachTimestep

This function evaluates to a list. Each item of the list is a list comprised of the datetime at which the smallest value was determined and the value itself.

Description	The smallest of several object's slot values, for each timestep in a range.	
Type	LIST{LIST{DATETIME, NUMERIC}}	
Arguments		
1	STRING	Subbasin name
2	STRING	slot name
3	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
4	BOOLEAN	time conversion option ("TRUE" or "FALSE")
5	DATETIME	start date
6	DATETIME	end date

Evaluation	A list of slots is generated by searching all of the objects in the Subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values whose maximum to find are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, all of the object's slot values are compared, yielding one minimum value for each timestep in the time range of the datetime arguments. The function returns a list of two items, where the first and second items of the inner lists are the datetime and the smallest value, respectively.
Mathematical Expression	$\forall_{(t \text{ from } start \text{ to } end)}[\{t, Min(\forall_{(obj \text{ in } subbasin)}[obj.slotname])\}]$
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, RiverWare aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.

Syntax Example:

```
MinObjectsAtEachTimestep("upper basin", "Storage", "ALL", FALSE
@"October, Previous Year",
@"September, Current Year")
```

Return Example:
For a monthly model, the above function would return something like:

```
{ { 24:00 October 31, 1996, 1232.2 "m3" },
  { 24:00 November 30, 1996, 1074.3 "m3" },
  ....
  { 24:00 September 30, 1997, 1564.0 "m3" } }
```

123. MinTimestepsAggregatedOverObjects

This function evaluates to a single numeric value which is the smallest value resulting from aggregating several objects' slot values at each timestep.

Description	Smallest over a timeseries of values, each of which is the result of aggregating several objects' slot values.	
Type	LIST	
Arguments		
1	STRING	Subbasin name

2	STRING	slot name
3	STRING	aggregation function ("SUM", "AVG", "MIN", or "MAX")
4	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
5	BOOLEAN	time conversion option ("TRUE" or "FALSE")
6	DATETIME	start datetime
7	DATETIME	end datetime
Evaluation	A list of slots is generated by searching all of the objects in the Subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values to be aggregated are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, all of the objects' slot values are aggregated according to the aggregation function argument for each timestep in the time range of the datetime arguments. During each of these slot aggregations, any values which do not satisfy the aggregation filter argument are ignored. Finally, the smallest value in the timeseries of object aggregated slot values is determined. This value is returned as the second value in a list. The first item is the date/time associated with this value. If there is an object associated with the value, it is returned as the third value in the list. This will be the case if the "MIN" or "MAX" aggregation function is specified for the third argument.	
Mathematical Expression	$Min(\forall_{(t \text{ from } start \text{ to } end)}[\forall_{(obj \text{ in } subbasin)}[AggFunction_{(t)}(obj.slotname)]])$	
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, RiverWare aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.	

Syntax Example:

```
MinTimestepsAggregatedOverObjects("upper basin","Storage","MAX","ALL",
 FALSE, @"October, Previous Year",
 @"September, Current Year")
```

Return Example:

```
{ @"March 31, 2001", "0.23"cms", %"Res1"}
```

124. MinTimestepsForEachObject

This function evaluates to a list. Each item of the list is a list comprised of the object name and the smallest value of the slot on that object for the time range specified.

Description	Smallest value in a slot over a time range, for each object in a subbasin.	
Type	LIST {LIST {OBJECT, NUMERIC}}	
Arguments		
1	STRING	Subbasin name
2	STRING	slot name
3	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
4	BOOLEAN	time conversion option ("TRUE" or "FALSE")
5	DATETIME	start datetime
6	DATETIME	end datetime
Evaluation	A list of slots is generated by searching all of the objects in the Subbasin argument for slots which match the slot name argument. For each object, the smallest slot value over every timestep in the range of the datetime arguments is determined. Any values which do not satisfy the aggregation filter argument are ignored during the calculation. If the time conversion option argument is TRUE, and the values to be aggregated are of the FLOW unit type, the values are first multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME.	
Mathematical Expression	$\forall_{(obj\ \mathrm{in}\ subbasin)}[\{obj, Min(\forall_{(t\ \mathrm{from}\ start\ \mathrm{to}\ end)}[obj.slotname])\}]$	
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, this function aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, this function also aborts RiverWare with an error.	

Syntax Example:

```
MinTimestepsForEachObject("upper basin", "Inflow", "ALL", TRUE,
@"October, Previous Year",
@"September, Current Year")
```

Return Example:

```
{ {%"Res1", 0.0"cms"}, {%"Reach2", 0.02 "cms"}, {%"Res2", 3.2 "cms"} }
```

125. Mod

This function computes the integer modulus of two numbers.

Description	Integer modulus of two numbers.	
Type	NUMERIC	
Arguments		
1	NUMERIC	the numerator
2	NUMERIC	the units to which to convert the numerator
3	NUMERIC	the denominator
4	NUMERIC	the units to which to convert the denominator
Evaluation	Converts numerator and denominator into the specified units, then returns the integral modulus of the converted values, where integral modulus of x and y returns the integral remainder after integral division of x and y, which can be defined as: $\mathrm{Mod}(x, y) \equiv y \cdot \left(\frac{\lfloor x \rfloor}{\lfloor y \rfloor} - \left\lfloor \frac{\lfloor x \rfloor}{\lfloor y \rfloor} \right\rfloor \right)$	
Comments	If the denominator is equal to zero, the run is aborted with an error. Each of the units arguments must have units which are compatible with the value they are associated with, otherwise the run is aborted with an error. Note that this function does not use the scalar portion of either of the units arguments.	

Syntax Example:

```
Mod(3.9 "m", 0.0 "ft", 5.0 "sec", 0.0 "sec")
```

Return Example:

```
2.0
```

126. NetNonShortDiversionRequirement

This function computes the diversion required to satisfy all of Water Users’ requests in an Aggregate Diversion Site.

Description	Minimum diversion required to meet all Water Users’ requests.

Type	NUMERIC	
Arguments		
1	OBJECT	the aggregate diversion site or diversion object
2	DATETIME	the timestep
Evaluation	If the object is an Aggregate Diversion Site and is linked with the **No Structure** or **Lumped Structure**, this function evaluates to the current value of the **Total Diversion Requested** slot. If the object is an Aggregate Diversion Site and if the Aggregate Diversion Site is linked with the **Sequential Structure**, the function sets a total diversion requirement equal to the topmost Water User's **Diversion Requested** and then loops over the remaining Water Users. The water available at each element is calculated based on the upstream elements' diversions and their return flows. If this water is enough to satisfy the Water User's **Diversion Requested**, the total diversion requirement is not modified. If this water is not enough to satisfy the Water User's **Diversion Requested**, the total diversion requirement is increased to satisfy this Water User. If the object is a diversion object, this function evaluates to the current value of the **Diversion Request** slot. Note, the diversion object cannot use the Percent of Available method.	
Mathematical Expression	For sequential agg diversion sites: Max $(\forall_{(WU \text{ in } Agg)} \left[DivReq_{(WU)} + \sum_{Upstream\ WU} DivReq - \sum_{Upstream\ WU} ReturnFlow \right]$ $)$	
Comments	This function exits with an early termination if any of the required data used to solve the diversion is unknown. The required data is the same as that needed for the objects to fully dispatch, except **Total/Incoming Available Water**, which need not be known. For sequentially linked Agg Diversion Sites, the function takes into account whether **Return Flow,** or **Surface Return Flow** are linked to another object, or unlinked and available to the next Water User in the diversion.	

Syntax Example:

```
NetNonShortDiversionRequirement(%"CAP Diversion", @"t")
```

Return Example:

```
9.25 "cms"
```

127. NetSubbasinDiversionRequirement

This function computes the inflow to a subbasin required to satisfy all diversions in the subbasin while meeting minimum flow requirements below all diversion points.

Description	Minimum Inflow required to meet all diversion requests and minimum flows.	
Type	NUMERIC	
Arguments		
1	LIST	the subbasin's Reach and Confluence objects in downstream order
2	DATETIME	the timestep at which to calculate
Evaluation	The subbasin diversion requirement is originally set to zero. Each of the objects in the subbasin list is processed in the downstream order they are provided in. If the object is a Reach, the following calculations are performed: • If an Aggregate Diversion Site or Diversion Object is linked to the Reach's **Diversion** slot, the NetNonShortDiversionRequirement function is executed on the diversion. If a water user is linked, then the Water User's Diversion Request is used. This step determines the diversion request from the Reach. • The Reach's minimum flow just below the diversion point is determined from the Reach's **Minimum Diversion Bypass** slot. If this slot does not exist because of the selected User Method in the **Min Diversion Bypass** category, the minimum flow requirement is zero. • If nothing is linked to the **Diversion** slot, but a value exists in the slot, this value is assumed to be the diversion requirement for this reach. In this case, there is no minimum flow requirement below the diversion point. • The subbasin diversion requirement is recalculated as the greater of the previous subbasin diversion requirement or the Reach diversion requirement plus the minimum flow requirement plus any cumulative upstream diversions minus any cumulative upstream return flows minus any cumulative upstream tributary inflows. • Any **Local Inflow** to the Reach is added to the cumulative tributary inflows.	

	• If the **Return Flow** slot has a valid value, it is added to the cumulative return flows. If the **Return Flow** slot does not have a valid value, but a Water User or an Aggregate Diversion Site object is linked to it, the return flow is estimated. Return flow is estimated by subtracting the object's (**Total**) **Depletion Requested** from its (**Total**) **Diversion Requested**. The estimated return flow is then added to the cumulative return flows. If the object is a Confluence, the **Inflow1** and **Inflow2** slots are checked to determine which is the main subbasin flow, and which is the tributary inflow. The objects linked to the inflow slots are checked against the last Reach object to be processed. When a match is found, the other **Inflow**, if valid, is added to the cumulative tributary inflows. The loop continues until all objects in the list have been processed. The largest subbasin diversion requirement calculated at any diversion point is the total subbasin diversion requirement.
Mathematical Expression	$Max(\forall Reach \text{ in } subbasin)$ $\left[NetNonShortDiversionRequirement(AggDivSite) + minimum\ flow_{(Reach)} + \sum_{Upstream\ Reach} Diversion - \sum_{Upstream\ Reach} Return\ Flow - \sum_{Upstream\ Reach,\ Confluence} Tributary\ Inflow \right]$
Comments	This function exits its calling rule with an early termination if any of the required data used to solve the diversions are unknown. The required data is the same as that needed for the NetNonShortDiversionRequirement predefined function for each Aggregate Diversion Site along the subbasin. This function aborts the run with an error if an object other than a Reach or Confluence is in the subbasin list. One of the Confluence **Inflows** must be linked to the previous Reach object upstream, or an Aggregate Reach which contains the previous Reach object upstream as its last element. If this condition is not met, the Confluence cannot determine which slot is the tributary inflow and the function aborts the run with an error. All subbasin diversion requirement calculations are performed at the given timestep. Subbasin diversion requirement will not be correct if there are lags in Reaches. This predefined function is recommended for use in long timestep models or for subbasins where there is no lag between top and bottom.

Syntax Example:

```
NetDiversionRequirement({%"La Plata",%"Hesperus", %"Pine Ridge"},
                        @"t")
```

Return Example:

```
9.25 "cms"
```

128. NextDate

Returns the next date which matches a partially specified date.

Description	Resolves a partially specified date/time into the next (with respect to a reference date) date/time which matches the specified fields.	
Type	DATETIME	
Arguments		
1	DATETIME	a reference date/time.
2	DATETIME	a partially specified date/time.
Evaluation	The unspecified fields with a coarser resolution are resolved into the future with respect to the reference date. If there are finer resolution fields, they are filled in with default values (e.g., time with 24:00, day of the month with the last day of the month). Note that if the partial date can be resolved into the current date, it is. See the "Syntax Examples" section below for some examples.	
Comments	If the reference date/time is not fully specified or if the partial date/time is, then the run is aborted with an error.	

Syntax Example:

```
NextDate(@"t", @"March")
returns: 24:00 March 31, 1995
(assuming the current timestep is any date between March, 1994 and March, 1995)
NextDate(@"24:00 February 28, 1995", @"March")
returns: 24:00 March 31, 1995
NextDate(@"24:00 May 10, 1995", @"March 20")
returns: 24:00 March 20, 1996
NextDate(@"24:00 February 28, 1994", @"MAX DayOfMonth")
returns: February 28, 1994
NextDate(@"24:00 February 28, 1994", @"Tuesday")
returns: 24:00 March 1, 1994
NextDate(@"24:00 February 28, 1994", @"6:00 MAX DayOfYear")
returns: 6:00 December 31, 1994
NextDate(@"24:00 February 28, 1994", @"6:00")
returns: 6:00 March 1, 1994
NextDate(@"", @"") (returns: )
```

129. NumberToDate

Description	Given a numeric encoding of a date/time, returns the corresponding date/time value.	
Type	DATETIME	
Arguments		
1	NUMERIC	The numeric encoding of a date/time value.
Comments	Slots representing date/time values have unit type DateTime. Internally these values are represented as numbers although the interface displays them as date/times. Looking up a value on such a slot will retrieve the numeric encoding, this function converts that number to a date/time value as required to treat it as a date within policy. If the unit for the slot corresponds to a partially specified date/time format, then the result will a partially specified date/time value.	

Syntax Example:

```
NumberToDate(Data.PriorityDate[])
```

Return Example:

The above call might return @"January 12" if the Data.PriorityDate slot has units "MonthAndDate".

Use Examples:

This function should be used in conjunction with Dates on Series slots **HERE (Slots.pdf, Section 4)** and the DateToNumber function **HERE (DateToNumber)**. A specific use example is shown **HERE (Slots.pdf, Section 4.3)**.

130. NumberToYear

Description	Given a numeric value, NumberToYear returns a DATETIME with only the year.	
Type	DATETIME	
Arguments		
1	NUMERIC	A number.
Evaluation	NumberToYear truncates the specified numeric value and returns the value as a year DATETIME.	

Syntax Example:

```
NumberToYear(2013.987 “s”)
```

Return Example:

```
@“Year 2013”
```

131. NumColumns/NumRows

Description	Returns the number of columns/rows in a table slot or periodic slot.	
Type	NUMERIC	
Arguments		
1	SLOT	a table or periodic slot.
Comments	If the slot is not a table or periodic slot, the run is aborted with an error.	

Syntax Example:

```
NumRows($"Data.MyTable") = 3
```

132. ObjAcctSupplyByWaterTypeRelTypeDestType

Description	This function finds a list of objects, accounts and supplies that match the given arguments. It returns a list of triplets{ OBJECT object, STRING account, STRING supply }, where the object^account is served by the supply, and the object is in the given subbasin (argument 1), the supply has the given release type and destination type (arguments 3 and 4), and the *supplying account* (upstream end of the supply in the returned triplet) has the given water type (argument 2).	
Type	LIST (LIST { OBJECT, STRING, STRING })	
Arguments		
1	STRING	The name of the subbasin in which to search.
2	STRING	The water type of the upstream end of the supplies returned. The string "ALL" means that any water type will satisfy this filter. The string "NONE" means that only supplying accounts lacking a water type will satisfy this filter.
3	STRING	The release type of the supply returned. The string "ALL" means that any release type will satisfy this filter. The string "NONE" means that only supplies lacking a release type will satisfy this filter.
4	STRING	The destination type of the supplies returned. The string "ALL" means that any destination type will satisfy this filter. The string "NONE" means that only supplies lacking a destination type will satisfy this filter.
Comments	This function is meant to be used in conjunction with the water rights solvers (SolveWaterRights()). It looks for supplies that are "appropriation points" for legal water accounts as defined for the water rights solver. In the solver, these supplies are identified by the water type of the account at the point of appropriation. Usually these supplies directly supply the object^account in the returned triplets. The one exception to this is when the supply serves an offstream reservoir. In this case, the offstream reservoir is supplied through a diversion object, and so a passthrough account on the diversion object sits between the point of diversion and the receiving object^account. This is the only case in which any indirection is detected, and the function looks two hops upstream to check the supplying account's water type. In all other cases, the function looks only one hop upstream.	

Syntax Example:

```
ObjAcctSupplyByWaterTypeRelTypeDestType("WRA", "MyWT", "MyRel", "MyDest")
```

Return Example:

```
{ {"Res1", "Farmer1", "Res1 Farmer1 Diversion to Farmer1 Diversion"},
  {"Res1", "Farmer2", "Res1 Farmer2 Diversion to Farmer2 Diversion"}  }
```

133. ObjectAttributeValue

Description	For the specified Object and a string representing an attribute, return the value for that particular Object, as a string	
Type	STRING	
Arguments		
1	OBJECT	The Object
2	STRING	The name of the Attribute
Evaluation		
Comments	If the attribute is not found on the object, an error is issued. If the object or attribute is not found in the **model**, an error will be issued.	

Syntax Example:

```
ObjectAttributeValue(%"Dolores", "State")
```

Return Example:

```
"Colorado"
```

134. ObjectHasAttributeValue

Description	Return whether the particular Object has the specified Attribute Value.	
Type	BOOLEAN	
Arguments		
1	OBJECT	The Object
2	STRING	The name of the Attribute
3	STRING	The Value of the Attribute.
Evaluation	When evaluated, the function looks at the particular object and checks to see if it has the given Attribute and Value.	
Comments	If the attribute or value is not found on the **object**, FALSE is returned. If the object, attribute or value is not found in the **model**, an error will be issued.	

Syntax Example:

```
ObjectHasAttributeValue(%"Dolores", "State", "Colorado")
```

Return Example:

```
TRUE
```

135. ObjectiveValue

Description	Returns the objective value from the last successful solution to the Optimization problem.	
Type	NUMERIC	
Arguments		
Comments	If there is no solution information available (e.g., if an Optimization run has not occurred), the run is aborted with an error diagnostic.	

Syntax Example:

```
ObjectiveValue()
```

Return Example:

```
12.23
```

136. ObjectsFromAccountName

Description	Returns a list of the objects that contain an account with the given name and account type.	
Type	LIST{OBJECT}	
Arguments		
1	STRING	The name of the account.
2	STRING	The name of the account type, e.g, "Storage". The string "ALL" for account type designates all account types.
Comments		

Syntax Example:

```
ObjectsFromAccountName("Municipal", "Storage")
```

Return Example:

```
{%"Reservoir1", %"Reservoir2"}
```

137. ObjectsFromAttributeValue

Description	Given a string representing an attribute and a string representing a value, return a list of all of the objects that have that value for the attribute	
Type	LIST {OBJECT, OJBECT, ...}	
Arguments		
1	STRING	The name of the Attribute
2	STRING	The Value of the that Attribute.
Evaluation	When evaluated, the function looks throughout the model and finds all objects that have the Attribute Value pair.The set of matching objects is returned in a list.	
Comments	If the attribute or value is not found in the model, an error will be issued.	

```
ObjectsFromAttributeValue("State", "Colorado")
```

Return Example:

```
{%"Arkansas", %"RioGrande", %"SanJuan", %"Dolores", %"Gunnison"}
```

138. ObjectsFromWaterType

Description	Returns a list of the objects that have an account with given water type and account type.	
Type	LIST{OBJECTS}	
Arguments		
1	STRING	The water type. The string "ALL" for water type designates all water types, and the string "NONE" designates the default water type.
2	STRING	The name of the account type, e.g, "Storage". The string "ALL" for account type designates all account types.
Comments		

Syntax Example:

```
ObjectsFromWaterType("ALL", "Storage")
```

Return Example:

```
{%"Reservoir1", %"Reservoir2"}
```

139. OffsetDate

This function adds some number of timesteps to a given date/time and returns the result.

Description	Returns the date/time which is some number of timesteps added to or subtracted from an input date/time.	
Type	DATETIME	
Arguments		
1	DATETIME	a date/time.
2	NUMERIC	the number of timesteps to add (a negative number will subtract). Should have units of "NONE".
3	STRING	a timestep specification. This specification includes an integer which can be positive or negative. Examples include "1 MONTH", "2 hours". Case is not important in this string.
Evaluation	Adds the given timestep, to the given date, to the given number of times, then returns the result.	

Comments	If the second argument has units other than "NONE", or if the third argument is not recognized as a timestep, then the run is aborted with an error. **Note:** Although one can put any integral amount within the timestep specification, if one makes this integer 1, then the second argument allows (1) one to vary the increment at the time of rule execution. Additional information on datetime math can be found **HERE (RPLTypesPalette.pdf, Section 1.3.4)**

Syntax Example:

```
OffsetDate(@"t", 1, "1 Months")
OffsetDate(@"January 1, 2000", getIncr(), "1 Hours")
```

Return Example:

```
@"July 31, 2007"
```

140. OperatingHeadToMaxRelease

This function performs a lookup in a Power Reservoir object's **Max Turbine Q table** based on a given operating head, and then evaluates to the corresponding maximum turbine release.

Description	Find the maximum turbine release at a given reservoir operating head.	
Type	NUMERIC	
Arguments		
1	OBJECT	power reservoir object
2	NUMERIC	operating head
3	DATETIME	datetime context for unit conversions
Evaluation	The operating head argument is looked up in the **Operating Head** column, of the **Max Turbine Q** table, of the power reservoir object argument, to determine the **Turbine Capacity**. If the exact operating head is not in the table, the lookup performs a linear interpolation between the two nearest bounding operating heads and their corresponding turbine capacities.	

Mathematical Expression	$turbine\ capacity = turbine\ capacity_{(lesser)} + \frac{turbine\ capacity_{(greater)} - turbine\ capacity_{(lesser)}}{operating\ head_{(greater)} - operating\ head_{(lesser)}} \times (operating\ head - operating\ head_{(lesser)})$
Comments	If the object is not a power reservoir, the function aborts the run with an error. If the Power Reservoir does not have a **Max Turbine Q** table, **Plant Power Coefficient** must be selected as the **Power** selected method, or this function exits the rule with an early termination.

Syntax Example:

```
OperatingHeadToMaxRelease(%"Hoover Dam", 508.63 "ft",
@"t")
```

Return Example:

```
152.23 "cms"
```

141. OptValue

Description	Returns a slot variable's optimal value as calculated during the last Optimization run.	
Type	NUMERIC	
Arguments		
1	SLOT	Desired Slot
2	DATETIME	Date at which to return the value
Evaluation	This function returns the optimal value for the given slot at the given timestep as part of the most recent optimization problem solution of the last run using the Optimization controller. For more information see the Optimization section **HERE (Optimization.pdf, Section 6.9)**.	
Comments	If there is no optimum value for the slot at that timestep, an invalid value is returned. This function supports the return of slot values which correspond to decision variables, that is to variables which are contained in the problem and are not replaced with linear combinations of other variables. If called for slots whose variables are not decision variables, an error message will be posted and the run will be aborted. Note that the OptValuePiecewise function allows access to the piecewise approximation of non-decision variables for which the piecewise approximation technique is applicable.	

Syntax Example:

```
OptValue(%"Norris.Outflow", @”18:00 Jan 23, 2009”)
```

Return Example:

```
152.23 "cms"
```

142. OptValueByCol

Description	Returns a slot variable’s optimal value as calculated during the last Optimization run for a variable associated with a particular column of an agg series slot.	
Type	NUMERIC	
Arguments		
1	SLOT	Specified Agg Series Slot
2	NUMERIC	Column index (0-based)
3	DATETIME	Date at which to return the value
Evaluation	This function returns the optimal value for the given slot and column at the given timestep as part of the most recent optimization problem solution of the last run using the Optimization controller. For more information see the Optimization section **HERE (Optimization.pdf, Section 6.9)**.	
Comments	If there is no optimum value for the slot and column at that timestep, an invalid value is returned. This function supports the return of slot values which correspond to decision variables, that is, to variables which are contained in the problem and are not replaced with linear combinations of other variables. If called for slots whose variables are not decision variables, an error message will be posted and the run will be aborted.	

Syntax Example:

```
OptValueByCol(%"Thermal.Hydro Block Use", 7, @”18:00 Jan 23, 2009”)
```

Return Example:

```
152.23 "MWH"
```

143. OptValuePiecewise

Description	Returns a slot variable's piecewise approximation value as calculated during the last Optimization run.	
Type	NUMERIC	
Arguments		
1	SLOT	Specified Slot
2	DATETIME	Date at which to return the value
Evaluation	Given a Series Slot and a date, this function returns the piecewise approximation of the slot at the given date, as computed during the most recent problem solution of the last optimization run. For more information see the Optimization section **HERE (Optimization.pdf, Section 6.9)**.	
Comments	The function will fail if the variables associated with the slot are not approximated (i.e., are decision variables or are replaced by a linear combination of other variables). If there is no approximation value available, perhaps because the quantity was not referenced in the optimization policy or there has been not been a successful optimization run, the function returns an invalid value.	

Syntax Example:

```
OptValuePiecewise(%"Kumquat Reservoir.Power", @”18:00 Jan 23, 2009”)
```

Return Example:

```
6.1 "MW"
```

144. Percentile

Returns the pth percentile from a list of values.

Description	This function returns the pth percentile from a list of values. In other words, it returns the value with a given percentile from a distribution.	
Type	NUMERIC	
Arguments		
1	LIST	A list of NUMERIC values representing the distribution.

2	NUMERIC	The probability p for which you wish the corresponding value. Note that p should be between 0 and 1 inclusive.
Evaluation	The list in argument one describes the distribution by providing independently sampled values from that distribution. The function returns an estimate of the value which has the given probability p of being greater than a value taken from the distribution. Consequently, for an input probability p at most (100*p)% of the values in the data set will be less than the return value (and at most 100*(1-p)% will be greater than this value). Several methods exist for computing the percentile; the following is the technical definition used by the Percentile function: for the pth percentile, Percentile(list, p): Compute p x (N + 1) where N is the number of items in the data set. Then set k and d, where k + d = p x (N + 1), k is an integer, and d is a fraction greater than or equal to 0 and less than 1. Essentially k is the integer part and d is the decimal part of p x (N + 1). Then, sort the numeric values in the list in increasing order. The function $Y_{[i]}$ denotes the i'th sorted value of the numeric list, where i is between 1 and N, inclusive. Then: If k = 0, Percentile(list, p) = $Y_{[1]}$ If 0 < k < N, Percentile(list, p) = $Y_{[k]} + (d)(Y_{[k+1]} - Y_{[k]})$ If k = N, Percentile(list, p) = $Y_{[N]}$ For more details, refer to “NIST/SEMATECH e-Handbook of Statistical Methods” (http://www.itl.nist.gov/div898/handbook/prc/section2/prc252.htm).	
Comments	Note that this function is sometimes called the “quantile” function. Excel’s, PERCENTILE function sets 1+p(N-1) equal to k + d, then proceeds as above. The two methods give similar results.	

Syntax Example:

```
Percentile({1”cfs”, 7”cfs”, 3”cfs”, 4”cfs”}, 0.3)
```

Return Example:

```
2.0”cfs”
```

145. PercentRank

Description	Find the rank of a given value within a list of values as a percentage of the number of values in the list.	
Type	NUMERIC	
Arguments		
1	LIST	a list of numeric values
2	NUMERIC	the value for which to determine the rank
Evaluation	This function provides a measure of the relative standing of a value within a data set. If the value whose percent rank is being determined, x, is one of the input values, then the return value is computed by: $\frac{\text{\# of values less than x}}{\text{\# of values} - \text{\# of values equal to x}}$ Otherwise, interpolation is used to combine the percent ranks for the closest data points on either size of x. If the input values are viewed as a sample from some distribution, then PERCENTRANK can be viewed as a smooth estimate of the empirical cumulative distribution function.	
Comments	Note, this function produces the same results as Excel's PERCENTRANK function.	

Syntax Example:

```
PercentRank({4"cfs", 4"cfs", 3"cfs", 1"cfs"}, 2.3"cfs")
```

Return Example:

```
0.21666667
```

146. PreviousDate

Returns the previous date which matches a partially specified date.

Description	Resolves a partially specified date/time into the previous (with respect to a reference date) date/time which matches the specified fields.
Type	DATETIME

Arguments		
1	DATETIME	a reference date/time.
2	DATETIME	a partially specified date/time.
Evaluation	The unspecified fields with a coarser resolution are resolved into the past with respect to the reference date. If there are finer resolution fields, they are filled in with default values (e.g., time with 24:00, day of the month with the last day of the month). Note that if the partial date can be resolved into the current date, it is. See the "Syntax Examples" section below for some examples.	
Comments	If the reference date/time is not fully specified or if the partial date/time is, then the run is aborted with an error.	

Syntax Example:

```
PreviousDate(@"t", @"March")
returns: 24:00 March 31, 1994
(assuming the t is any date between March, 1994 and March, 1995)
PreviousDate(@"24:00 May 10, 1995", @"March 20")
returns: 24:00 March 20, 1995
PreviousDate(@"24:00 February 28, 1994", @"MAX DayOfMonth")
returns: February 28, 1994
PreviousDate(@"24:00 February 28, 1994", @"Tuesday")
returns: 24:00 February 22, 1994
PreviousDate(@"24:00 February 28, 1994", @"6:00 MAX DayOfYear")
returns: 6:00 December 31, 1993
PreviousDate(@"24:00 February 28, 1994", @"6:00")
returns: 6:00 February 28, 1994
```

147. RanDev

Description	Returns the next number in a pseudo-random sequence.	
Type	NUMERIC	
Arguments		
1	NUMERIC	a number which is ignored except that the units are taken as the units to be returned.
Evaluation	Returns the next number in the pseudo-random series, given a seed.	
Comments	This function should not be called within a user-defined function which has no arguments, if that user-defined function might be called multiple times within a single block (rule). This is because functions with no arguments are actually evaluated only once per rule and return this same result on each function call during the execution of that block. This is not a very good random number generator, but is implemented in this way for historical reasons. If ResetRanDev() has not been called before this function, then the results are unpredictable.	

Syntax Example:

```
RanDev(1)
```

Return Example:

```
0.34105
```

148. Random, RandomNormal

Description	Returns a given number in a pseudo-random sequence.	
Type	NUMERIC	
Arguments		
1	NUMERIC	a numeric value which is rounded down to an integer and used to identify a unique sequence of numbers -- calls with the same integral seed refer to the same random sequence of numbers.
2	NUMERIC	a numeric value which is rounded down to an integer and denotes the one-based index into the random sequence of the value to be returned.
3	NUMERIC	a number which is ignored except that the units are taken as the units to be returned.
Evaluation	**Random** returns a number from a random sequence of numbers uniformly distributed in the range [0, 1.0]. **RandomNormal** returns a number from a random sequence of numbers whose distribution is normal with a mean of 0 and a standard deviation of 1. The unique sequence of numbers associated with each integral seed is the same on all platforms supported by RiverWare, allowing for repeatable results. The sequences are generated using the linear congruential method described in Park and Miller (1988) Communication of the ACM, vol 31, pages 1192-1201. Note: random number generators such as this are often referred to as "pseudo-random" because they are not the result of an intrinsically random process, are in fact predictably determined by the seed.	
Comments	The time to evaluate a call to either of these functions is proportional to the magnitude of the index argument (because the entire sequence must be generated at least once per RiverWare execution). Thus, if performance is an important issue, one should choose to get numbers from the earlier portion of a sequence.	

Syntax Example:

```
RandomNormal(1.0, 1.0, 1.0)
RandomNormal(1.0, 3.0, 0.0)
```

Return Example:

Refer to the sequence: 0.09151046 0.33494915 -1.421276 -1.24931121 ...

Thus, the first call returns the first number in the sequence (0.09151046) and the second call returns the third number in the sequence (0.33494915).

149. ReleaseTypes

This function evaluates to the list of user-defined ReleaseTypes

Description	This function returns a list of the names of all ReleaseTypes defined in the Water Accounting System Configuration.	
Type	LIST {STRING}	
Arguments		
Evaluation		
Comments	ReleaseTypes are properties of Supplies. The returned list does not include the default ("NONE") ReleaseType.	

Syntax Example:

```
ReleaseTypes()
```

Return Example:

```
{"MinimumFlows", "ProjectWater", "Flood"}
```

150. ReleaseTypesFromObject

This function evaluates to the list of ReleaseTypes which represent outflows from an Object

Description	This function returns a list of unique names of ReleaseTypes of Supplies which represent outflows from a specified Object.	
Type	LIST {STRING}	
Arguments		
1	OBJECT	The Object.
Evaluation	The set of Accounts on the Object are examined. The outflow Supplies on those Accounts which link a different downstream Object are considered. The names of the ReleaseTypes of those Supplies are added to the returned list -- but any given ReleaseType name will appear on the list only once.	

Comments	ReleaseTypes are properties of Supplies. The returned list can include the default ("NONE") ReleaseType. Supplies which represent "internal flows" between two Accounts on the Object are not considered.

Syntax Example:

```
ReleaseTypesFromObject(%"Reservior1")
```

Return Example:

```
{"MinimumFlows", "Flood"}
```

151. ResetRanDev

Description	Initialize internal data structures to permit RanDev() to return a pseudo-random sequence of numbers. This involves reading a file, each line of which has a date associated with it. Basically, this is a "seeding" function.	
Type	BOOLEAN	
Arguments		
1	BOOLEAN	True if some lines in the initialization file should be skipped.
2	DATETIME	The date of the line to be skipped.
Evaluation	Returns true if initialization was successful.	
Comments	The recommendation is that this function be called within a block that contains only the following statement: obj.slot[] = IF (NOT ResetRanDev(...)) STOP_RUN "ResetRanDev failed" ENDIF This will never assign any values but will always evaluate the function call. An alternative is to embed the call within a Print statement, but if diagnostics are turned off then this statement will not get executed. This, and the RanDev() function are scheduled to be removed in the future and replaced with a more convenient and effective means of generating a sequence of pseudo-random numbers.	

Syntax Example:

```
ResetRanDev(TRUE, @"24:00:00 October Max DayOfMonth, 1983")
```

Return Example:

```
TRUE or FALSE
```

152. Reverse

Description	Reverses the order of items in a list.	
Type	LIST	
Arguments		
1	LIST	a list of values
Evaluation	Returns a list with the same values as the input list, in reverse order.	
Comments		

Syntax Example:

```
Reverse({ 1.0, {res1, 10}, "hello", 0.0, "bob"})
```

Return Example:

```
{ "bob", 0.0, "hello", {res1, 10}, 1.0}
```

153. RowLabel

Description	Returns the label associated with a given row of a table slot.	
Type	STRING	
Arguments		
1	SLOT	A table slot
2	NUMERIC	The row index (0-based).
Evaluation	Returns the label of the row of the table slot which has the given index.	
Comments	It is an error to provide an illegal index (e.g., an index of 4 with a table which has only 4 rows). If the row index is legal but there is no label for that row, then the empty string is returned: "".	

Syntax Example:

```
RowLabel(DataObjA.CoeffTable, 2)
```

Return Example:

```
"Coefficient 1"
```

154. RowLabels

Description	Returns a list containing the labels of the rows of a given table slot, in order.	
Type	LIST of STRING values	
Arguments		
1	SLOT	A table slot or agg. series slot
Evaluation	Returns the label of the column of the table slot which has the given index.	
Comments	It is an error if the input slot has a type other than table slot. For each column, if no label exists the empty string is returned.	

Syntax Example:

```
RowLabels(DataObjA.CoeffTable)
```

Return Example:

```
{"Coefficient 1", "Coefficient 2", "Coefficient 3"}
```

155. RunStartDate and RunEndDate

Description	RunStartDate and RunEndDate return the start or end date of the currently active controller, respectively.	
Type	DATETIME	
Arguments		
Comments	When evaluated from a Rule, Goal, or Method set, these functions are equivalent to @"Start Timestep" or @"Finish Timestep". But, for Expression Series Slots, the symbolic datetime specifications @"Start Timestep" and @"Finish Timestep" refer to the expression slot's evaluation range, not the controller's start or end dates. Thus, RunStartDate() may not be equivalent to @"Start Timestep" and RunEndDate() may not be equivalent to @"Finish Timestep". But, regardless of the set from which they are called, RunStartDate and RunEndDate functions provide a fixed references to the controller's start and end dates, respectively.	

Syntax Example:

```
RunStartDate()
```

Return Example:

```
@"January 1, 2003"
```

156. RunTime

Description	Returns the number of seconds which have elapsed since the current run began, or if called from outside a run, the total number of seconds within the last run.
Type	NUMERIC

Syntax Example:

```
RunTime()
```

Return Example:

```
22.000 "s"
```

157. SlotCacheValue

Description	Returns a series slot's value from the slot cache.	
Type	NUMERIC	
Arguments		
1	SLOT	Series slot whose cache value is desired
2	DATETIME	Date for which the value is desired
Evaluation	Returns the slot cache value for the given series slot at the given date. If there is no such value, an invalid value is returned.	
Comments	The slot cache is a repository of series slot values which can be created from workspace series slots allowing access within one run to values computed by a previous run.The cache is described **HERE (Workspace.pdf, Section 5.10)**. **Note, the slot cache is under development. Please contact riverware-support@colorado.edu for more information and the current status of this feature.**	

Syntax Example:

```
SlotCacheValue(%"Berkley.Outflow", @”18:00 Jan 23, 2009”)
```

Return Example:

```
152.23 "cms"
```

158. SlotCacheValueByCol

Description	Returns an aggregate series slot's value from the slot cache.	
Type	NUMERIC	
Arguments		
1	SLOT	Aggregate series slot for which a slot cache value is desired.
2	NUMERIC	Index of the column for which a value is desired.
3	DATETIME	Date for which a value is desired.
Evaluation	Returns the slot cache value for the given series slot at the given date. If there is no such value, an invalid value is returned.	
Comments	The slot cache is a repository of series slot values which can be created from workspace series slots, allowing access within one run to values computed by a previous run. The cache is described **HERE (Workspace.pdf, Section 5.10)**. **Note, the slot cache is under development. Please contact riverware-support@colorado.edu for more information and the current status of this feature.**	

Syntax Example:

```
SlotCacheValueByCol(%"Klamath Data.Precip", 2, @”18:00 Jan 23, 2009”)
```

Return Example:

```
1.23 "m"
```

159. SlotWeightedAverageOverTime

Description	Computes a Series Slot's weighted average over a given time period, using another Series Slot's values in that same time range as the weights.	
Type	NUMERIC	
Arguments		
1	SLOT	Slot 1: the weighting slot
2	SLOT	Slot 2: the slot being averaged
3	DATETIME	Begin timestep for period of average - partially specified
4	DATETIME	End timestep for period of average - partially specified
	DATETIME	Reference timestep
Evaluation	The partially specified begin and end timesteps are converted to fully specified timesteps using the reference timestep to complete the missing information. The weighted average is then computed as the quotient of two summations over the averaging time period (begin timestep to end timestep): $\dfrac{\sum_{Begin}^{End} \text{Slot } 1[i] \times \text{Slot } 2[i]}{\sum_{Begin}^{End} \text{Slot } 1[i]}$	
Comments	For a calendar year weighting of monthly timesteps, the time arguments would be @“January”, @“December”, and @“t” For a water year weighting of monthly timesteps, the time arguments would be @“October 31, Previous Year”, @“September 30”, @“t”. For a monthly weighting, the time arguments would be @“DayOfMonth 1”, @“Max DayOfMonth”, @“t”.	

Syntax Example:

```
SlotWeightedAverageOverTime(Mead.Outflow, Mead.Salt Concentration, @"January",
@"December", @"t")
```

160. SolveInflow

This performs a mass balance and evaluates to the inflow of a reservoir given its outflow, previous storage, and end of timestep storage at the specified timestep.

Description	The inflow to a reservoir given outflow and storage.	
Type	NUMERIC	
Arguments		
1	OBJECT	the reservoir object for which to calculate
2	NUMERIC	the average outflow over the timestep
3	NUMERIC	the end of timestep storage
4	NUMERIC	the previous (beginning) storage
5	DATETIME	the timestep at which to calculate
Evaluation	This function calls the massBalanceSolveInflow() function on the given reservoir object at the given timestep, and provides it with the average outflow over the timestep, beginning storage, and ending storage. The function computes the end of timestep pool elevation, and then determines the average inflow over the timestep, taking into account the following sources and sinks. • The **Evaporation and Precipitation** category selected Method. • The **Bank Storage** category selected Method. • The **Seepage** category selected Method. • Side inflows like Hydrologic Inflow, Return Flow, and Diversion are NOT included. The total inflow is then calculated as the difference between the ending and beginning storage over the timestep, plus the outflow, evaporation, bank storage, and seepage, minus precipitation.	
Mathematical Expression	$Total\ Inflow\ = \frac{storage_{previous} - storage_{ending}}{\Delta t_{timestep}} + outflow$ $+ evaporation_{flow} + bank\ storage_{flow} + seepage$ $- precipitation_{flow}$	
Comments	The given outflow is a total outflow and should include any spills. The calculated inflow is a total inflow.	

Syntax Example:

```
SolveInflow(%"Hoover Dam", 13651 "cfs", 19853486 "acrefeet",
            19787262 "acrefeet", @"June, 1984"}
returns 12.5 "cms"
```

161. SolveOutflow

This performs a mass balance and evaluates to the outflow from a reservoir given its inflow, previous storage, and end of timestep storage at the specified timestep.

Description	The outflow from a reservoir.	
Type	NUMERIC	
Arguments		
1	OBJECT	the reservoir object for which to calculate
2	NUMERIC	the average inflow over the timestep
3	NUMERIC	the end of timestep storage
4	NUMERIC	the previous (beginning) storage
5	DATETIME	the timestep at which to calculate
Evaluation	This function calls the massBalanceSolveOutflow() function on the given reservoir object at the given timestep and provides it with the average inflow over the timestep, beginning storage, and ending storage. The function computes the end of timestep pool elevation, and then determines the average outflow over the timestep, taking into account the following sources and sinks, and thus they should not be included in the inflow value for Argument 2. • The **Evaporation and Precipitation** category selected Method. • The **Bank Storage** category selected Method. • The **Seepage** category selected Method. • Side inflows including: **Inflow 2** (Slope Power Reservoir only), **Hydrologic Inflow Net**, **Diversion**, **Return Flow**, **Canal Flow**, **Flow FROM Pumped Storage**, and **Flow TO Pumped Storage.** These slots are automatically added as dependencies to the calling rule. The outflow is then calculated as the difference between the ending and beginning storage over the timestep, plus the inflow, side inflows, and precipitation, minus evaporation, bank storage, and seepage.	

Mathematical Expression	$Outflow = \frac{storage_{previous} - storage_{ending}}{\Delta t_{timestep}} + inflow + side\ inflows - evaporation_{flow} - bank\ storage_{flow} - seepage + precipitation_{flow}$
Comments	The given inflow in argument 2 represents the main inflow only and should not include any side inflows. This is the same value which would be in the **Inflow** slot. The calculated outflow is a total outflow. It includes both **Release/Turbine Release** and **Spill**. The given timestep's **Inflow 2** (Slope Power Reservoir only), **Hydrologic Inflow Net, Diversion, Return Flow, Canal Flow, Flow FROM Pumped Storage,** and **Flow TO Pumped Storage.** are automatic dependencies of this function. Since the function evaluation depends on these slots, any change to their values at the indicated timestep, may impact the function result.

Syntax Example:

```
SolveOutflow(%"Hoover Dam", 11651 "cfs", 19853486 "acrefeet",
             19787262 "acrefeet", @"June, 1984"}
```

Return Example:

```
21.32 "cms"
```

162. SolveOutflowGivenEnergyInflow

This function evaluates to Outflow from a LevelPowerReservoir with the given Energy and Inflow at the specific timestep.

Description	The outflow from a LevelPowerReservoir.	
Type	NUMERIC	
Arguments		
1	OBJECT	the reservoir object for which to calculate (must be a LevelPowerReservoir)
2	NUMERIC	the energy value
3	NUMERIC	the inflow value
4	DATETIME	the timestep at which to calculate

Evaluation	This function behaves identically to the solution of the LevelPowerReservoir in simulation.
Comments	This function assumes that the LevelPowerReservoir has solved for all the timesteps prior to the date specified in argument 4. This is necessary because the solution requires previous storage, inflow, and energy. This information is retrieved from slots on the object at timesteps prior to the date specified in argument 4. If any of this information is missing, an error is posted and the rule fails. If this function is called on the first timestep, the initial input data are used. These data are already required for the LevelPowerReservoir to dispatch in simulation mode. This function takes into account the following sources and sinks automatically, and thus they should not be included in the inflow value for Argument 2. • The **Evaporation and Precipitation** category selected Method. • The **Bank Storage** category selected Method. • The **Seepage** category selected Method. • Side inflows including: **Hydrologic Inflow Net**, **Diversion**, **Return Flow**, **Canal Flow**, **Flow FROM Pumped Storage**, and **Flow TO Pumped Storage.** These slots are automatically added as dependencies to the calling rule.

Syntax Example:

```
SolveOutflowGivenEnergyInflow(%"HooverDam",HooverDam.Energy[],
                              HooverDam.Inflow[], @"t")
SolveOutflowGivenEnergyInflow(%"HooverDam",20.0"MWH",1000.0"cfs",
                              @"t")
```

Return Example:

```
16.342 "cms"
```

163. SolveShortage

Given some total available water, this method solves for the Diversion Shortage and Depletion Shortage on a Water User, or the Total Diversion Shortage and Total Depletion Shortage on an AggDiversionSite. It evaluates to a list which contains the two values.

Description	List containing the (Total) Diversion Shortage and (Total) Depletion Shortage	
Type	LIST {NUMERIC, NUMERIC}	
Arguments		

1	OBJECT	the object on which to perform the calculations (either an AggDiversionSite or a Water User)
2	NUMERIC	the total water available for diversion
3	DATETIME	the timestep at which to calculate
Evaluation	This function behaves identically to the solution of the object in simulation. It mimics the dispatch method of the given object. However, instead of setting slots, the method just returns the values for (Total) Diversion Shortage and (Total) Depletion Shortage.	
Comments	This function exits its calling rule with an early termination if any of the required data used to solve the diversions are unknown. Note: Depletion Requested is not required, if not specified it will be set equal to Diversion Requested. This function aborts the run with an error if an object other than a Water User or an AggDiversionSite is given as the first argument.	

Syntax Example:

```
SolveShortage(%"San Juan Diversion", 100 "cfs", @"t"}
```

Return Example:

```
{1.25 "cms", 1.02 "cms"}
```

164. SolveSlopeStorageGivenInflowHW

This function is used to solve a Slope Power Reservoir object when inflow and pool elevation are known. A LIST is returned which contains the resulting outflow as the first argument and the resulting storage as the second argument.

Description	List containing the resulting outflow and the resulting storage value	
Type	LIST {NUMERIC, NUMERIC}	
Arguments		
1	OBJECT	the object on which to perform the calculations (must be a Slope Power Reservoir)
2	NUMERIC	the inflow value
3	NUMERIC	the pool elevation value
4	DATETIME	the timestep at which to calculate
Evaluation	This function behaves identically to the solution of the object in simulation.	

Comments	This function assumes that the Slope Power Reservoir has solved (through simulation) for all timesteps prior to the date specified in argument 4. This is necessary because the solution requires previous inflow, outflow, storage and pool elevation data. This information is retrieved from slots on the object at timesteps prior to the date specified in argument 4. If any of this information is missing, an error is posted and the rule fails. If this function is called on the first timestep, the initial input data is used. This data is already required for the Slope Power Reservoir to dispatch in simulation mode. This function takes into account the following sources and sinks automatically, and thus they should not be included in the inflow value for Argument 2. • The **Evaporation and Precipitation** category selected Method. • The **Bank Storage** category selected Method. • The **Seepage** category selected Method. • Side inflows including: **Inflow 2**, **Hydrologic Inflow Net**, **Diversion**, **Return Flow**, **Canal Flow**, **Flow FROM Pumped Storage**, and **Flow TO Pumped Storage.** These slots are automatically added as dependencies to the calling rule.

Syntax Example:

```
SolveSlopeStorageGivenInflowHW(%"FtLoudoun", FtLoudoun.Inflow[], FtLoudoun.Pool
                               Elevation[], @"t")
SolveSlopeStorageGivenInflowHW(%"FtLoudoun", 100.0 "cfs", 240.45 "ft",
                               @"t")
```

Return Example:

```
{16.342 "cms", 123348183.75 "m3"}
```

165. SolveSlopeStorageGivenInflowOutflow

This function is used to solve a Slope Power Reservoir object when inflow and outflow are known. A LIST is returned which contains the resulting pool elevation as the first argument and the resulting storage as the second argument.

Description	List containing the resulting pool elevation and the resulting storage value	
Type	LIST {NUMERIC, NUMERIC}	
Arguments		
1	OBJECT	the object on which to perform the calculations (must be a Slope Power Reservoir)
2	NUMERIC	the inflow value

3	NUMERIC	the outflow value
4	DATETIME	the timestep at which to calculate
Evaluation	This function behaves identically to the solution of the object in simulation.	
Comments	This function assumes that the Slope Power Reservoir has solved (through simulation) for all timesteps prior to the date specified in argument 4. This is necessary because the solution requires previous inflow, outflow, storage and pool elevation data. This information is retrieved from slots on the object at timesteps prior to the date specified in argument 4. If any of this information is missing, an error is posted and the rule fails. If this function is called on the first timestep, the initial input data is used. This data is already required for the Slope Power Reservoir to dispatch in simulation mode. This function takes into account the following sources and sinks automatically, and thus they should not be included in the inflow value for Argument 2. • The **Evaporation and Precipitation** category selected Method. • The **Bank Storage** category selected Method. • The **Seepage** category selected Method. • Side inflows including: **Inflow 2**, **Hydrologic Inflow Net**, **Diversion**, **Return Flow**, **Canal Flow**, **Flow FROM Pumped Storage**, and **Flow TO Pumped Storage.** These slots are automatically added as dependencies to the calling rule.	

Syntax Example:

```
SolveSlopeStorageGivenInflowOutflow(%"FtLoudoun", FtLoudoun.Inflow[],
                                    FtLoudoun.Outflow[], @"t")
SolveSlopeStorageGivenInflowOutflow(%"FtLoudoun", 100.0 "cfs", 110.45 "cfs",
                                    @"t")
```

Return Example:

```
{1253.2 "m", 123348183.75 "m3"}
```

166. SolveStorage

This performs a mass balance and evaluates to the end of timestep storage of a reservoir, given its previous storage and average inflow and outflow at the specified timestep.

Description	The storage of a reservoir.	
Type	NUMERIC	
Arguments		

1	OBJECT	the reservoir object for which to calculate
2	NUMERIC	the average inflow over the timestep
3	NUMERIC	the average outflow over the timestep
4	NUMERIC	the previous (beginning) storage
5	DATETIME	the timestep at which to calculate
Evaluation	This function calls the massBalanceSolveStorage() function on the given reservoir object at the given timestep and provides it with the average inflow and outflow over the timestep, and beginning storage. The function must iterate to convergence due to the storage and pool elevation dependence of the following sources and sinks, which are included automatically: • The **Evaporation and Precipitation** Category selected Method. • The **Bank Storage** category selected Method. • The **Seepage** category selected Method. • Side inflows including: **Inflow 2** (Slope Power Reservoir only), **Hydrologic Inflow Net**, **Diversion**, **Return Flow**, **Canal Flow**, **Flow FROM Pumped Storage**, and **Flow TO Pumped Storage**. These slots are automatically added as dependencies to the calling rule. At each iteration, the ending storage is calculated as the previous storage plus the inflow, side inflows, and precipitation over the timestep, minus the evaporation, bank storage, and seepage over the timestep	
Mathematical Expression	$Storage = storage_{previous} + ((inflow + side\ inflows)\Delta t_{timestep}) - ((outflow)\Delta t_{timestep}) - evaporation_{volume} - bank\ storage_{volume} - ((seepage)\Delta t_{timestep}) + precipitation_{volume}$	
Comments	The given inflow in Argument 2 represents the main inflow only and should not include any side inflows. This is the same value which would be in the **Inflow** slot. The given outflow represents the total outflow. It should include both **Release/ Turbine Release** and **Spill**. The given timestep's **Inflow 2** (Slope Power Reservoir only), **Hydrologic Inflow Net**, **Diversion**, **Return Flow**, **Canal Flow**, **Flow FROM Pumped Storage**, and **Flow TO Pumped Storage** are automatic dependencies of this function. Since the function evaluation depends on these slots, any change to their values at the indicated timestep, may impact the function result.	

Syntax Example:

```
SolveStorage(%"Hoover Dam", 11651 "cfs", 13672 "cfs",
             19787262 "acrefeet", @"June, 1984")
```

Return Example:

```
123348183.75 "m3"
```

167. SolveSubbasinDiversions

This function evaluates to a list of two values. The first value, is the minimum inflow to a subbasin required to satisfy all of its diversions. The second value, is the outflow from the subbasin when this minimum flow is available.

Description	Minimum Inflow required to meet all diversion requests and resulting Outflow.	
Type	LIST {NUMERIC, NUMERIC}	
Arguments		
1	LIST	the subbasin’s Reach and Confluence objects in downstream order (can be included in a subbasin)
2	DATETIME	the timestep at which to calculate

<table>
<tr>
<td rowspan="2">Evaluation</td>
<td>The subbasin diversion requirement is originally set to zero. Each of the objects in the subbasin list is processed in the downstream order in which they are provided. If the object is a Reach, the following calculations are performed:

If an Aggregate Diversion Site is linked to the Reach’s Diversion slot, the NetNonShortDiversionRequirement function is executed on the diversion. This determines the diversion requirement from the Reach.
If an Aggregate Diversion Site is linked to the Reach’s Diversion slot, the minimum flow just below the diversion point is determined from the Reach’s Minimum Diversion Bypass slot. If this slot does not exist because of the selected User Method in the Min Diversion Bypass category, the minimum flow requirement is zero.
If nothing is linked to the Diversion slot, but a value exists in the slot, this value is assumed to be the diversion requirement for this reach. In this case, there is no minimum flow requirement below the diversion point.
The subbasin diversion requirement is recalculated as the greater of the previous subbasin diversion requirement or the Reach diversion requirement plus the minimum flow requirement plus any cumulative upstream diversions minus any cumulative upstream return flows minus any cumulative upstream tributary inflows.
Any Local Inflow to the Reach is added to the cumulative tributary inflows.
If the Return Flow slot has a valid value, it is added to the cumulative return flows. If the Return Flow slot does not have a valid value, but a Water User or an Aggregate Diversion Site object is linked to it, the return flow is estimated. Return flow is estimated by subtracting the object’s (Total) Depletion Requested from its (Total) Diversion Requested. The estimated return flow is then added to the cumulative return flows. If Depletion Requested is not specified, it will be set equal to Diversion Requested.

</td>
</tr>
<tr>
<td>If the object is a Confluence, the Inflow1 and Inflow2 slots are checked to determine which is the main subbasin flow and which is the tributary inflow. The objects linked to the inflow slots are checked against the last Reach object to be processed. When a match is found, the other Inflow, if valid, is added to the cumulative tributary inflows.

The loop continues until all objects in the list have been processed. The largest subbasin diversion requirement to have been calculated at any diversion point is the total subbasin diversion requirement.</td>
</tr>
</table>

Mathematical Expression	$Max(\forall Reach \text{ in } subbasin)\left[NetNonShortDiversionRequirement(AggDivSite) + minimum\ flow_{(Reach)} + \sum_{Upstream\ Reach} Diversion - \sum_{Upstream\ Reach} Return\ Flow - \sum_{Upstream\ Reach,\ Confluence} Tributary\ Inflow\right]$
Comments	This function exits its calling rule with an early termination if any of the required data used to solve the diversions are unknown. The required data is the same as that needed for the NetNonShortDiversionRequirement predefined function for each Aggregate Diversion Site along the subbasin. This function aborts the run with an error if an object other than a Reach or Confluence is in the subbasin list. One of the Confluence **Inflows** must be linked to the previous Reach object upstream, or an Aggregate Reach which contains the previous Reach object upstream as its last element. If this condition is not met, the Confluence cannot determine which slot is the tributary inflow and the function aborts the run with an error. All subbasin diversion requirement calculations are performed at the given timestep. Subbasin diversion requirement will not be correct if there are lags in Reaches. This predefined function is recommended for use in long timestep models or for subbasins where there is no lag between top and bottom.

Syntax Example:

```
SolveSubbasinDiversions(ListSubbasin("AnimasBasin"), @"t")
```

Return Example:

```
{ 0.954 "cms", 0.00 "cms" }
```

168. SolveTurbineRelGivenEnergyInflow

Description	This function computes the Turbine Release necessary to meet the specified Energy. If that energy cannot be met, the maximum turbine release is returned.	
Type	LIST {NUMERIC, BOOLEAN}	
Arguments		
1	OBJECT	the reservoir object for which to calculate (must be a LevelPowerReservoir)
2	NUMERIC	the energy value
3	NUMERIC	the inflow value
4	DATETIME	the timestep at which to calculate
Evaluation	This function behaves like the solution of the LevelPowerReservoir in simulation. If the given energy can be met by the turbine release, TRUE is returned in the list boolean. If the given energy cannot be met, the turbine release is calculated to be the maximum release as computed by GetMaxReleaseGivenInflow and FALSE is returned in the list boolean.The maximum turbine release may be reduced by specified Plant Power Limits and Plant Power Cap Fractions, as appropriate for the selected power method.	

Comments	This function assumes that the LevelPowerReservoir has solved for all the timesteps prior to the date specified in argument 4. This is necessary because the solution requires previous Storage. This information is retrieved from slots on the object at timesteps prior to the date specified in argument 4. If any of this information is missing, an error is posted and the rule fails. If this function is called on the first timestep, the initial input data are used. These data are already required for the LevelPowerReservoir to dispatch in simulation mode. This function takes into account the following sources and sinks automatically, and thus they should not be included in the inflow value for Argument 3. • The **Evaporation and Precipitation** category selected Method. • The **Bank Storage** category selected Method. • The **Seepage** category selected Method. • Side inflows including: **Hydrologic Inflow Net**, **Diversion**, **Return Flow**, **Canal Flow**, **Flow FROM Pumped Storage**, and **Flow TO Pumped Storage.** These slots are automatically added as dependencies to the calling rule. Also, if there are Unregulated Spills, the unregulated spill is limited to be no greater than the max unregulated spill. This is described in the Unregulated Spill documentation on the Level Power Reservoir. Note, for the **Plant Power Coefficient** and **Plant Efficiency Curve** power methods, if you have an input value on the **Power Coefficient** slot, the result of solveTurbineRelGivenEnergyInflow is non-unique. A value will be found, but there may be multiple solutions that meet the specified energy.

Syntax Example:

```
SolveTurbineRelGivenEnergyInflow(%"HooverDam",HooverDam.Energy[],
                                  HooverDam.Inflow[], @"t")
SolveTurbineRelGivenEnergyInflow(%"HooverDam", 20.0 "MWH", 1000.0  "cfs, @"t")
```

Return Example:

```
{16.342 "cms", TRUE}
```

169. SolveWaterRights and SolveWaterRightsWithLags

This water accounting function invokes the Water Rights Allocation method on a computational subbasin **HERE (Accounting.pdf, Section 10)**. The subbasin identifies a set of accounts for which to solve; the Water Type identifies the supply chain that models the allocatable flow of water in the subbasin. The date controls the behavior of instream flow rights during the solution.

Description	Invokes computational subbasin's Water Rights Allocation method.	
Type	SolveWaterRights: LIST { LIST { SLOT, NUMERIC } } SolveWaterRightsWithLags: LIST{ LIST{SLOT, DATETIME, NUMERIC} }	
Arguments		
1	STRING	the name of the computational subbasin
2	STRING	the name of the Water Type that identifies the allocatable flow supply chain
3	DATETIME	"controlling date" for instream flow rights. Rights at or senior to (i.e., with priority date earlier or equal to) this date can make calls; instream flow rights junior to this date compute their Available Allocatable Flow.
Evaluation	Runs the selected Water Rights Allocation method on the subbasin. For each water right account (has priority) in the subbasin, the function returns {slot, value} pairs for the following slots: • **Appropriation Request** on all rights, • **Available Allocatable Flow** on Instream Flow Accounts whose priority date is later than the controlling date (3rd argument), • **Supply** slot values representing appropriations to the water right accounts. For storage rights on in-line reservoirs, this is a Transfer In supply; for off-stream storage rights, this is a Diversion supply to the passthrough account on a Diversion object that supplies the off-stream storage right account. For diversion rights, this is a Diversion supply. A changing set of temporary slots (whose names begin with **Temp**) on the rights is also returned, for use by RiverWare developers. If no appropriation is to be made, a value of zero is returned so that old appropriations that are no longer valid will be invalidated by this rule. The SolveWaterRightsWithLags() predefined rule function works much like SolveWaterRights(), but is used when the subbasin passthrough accounts contain lags. It returns a list of {slot name, date-time, value} triplets, which the rule uses to place the value in the appropriate slot at the appropriate timestep. The timestep given will reflect the Local Timestep Offset of the account on which the slot resides. It is some number of timesteps after the current rules-controller timestep, and reflects the relationships of the account to other accounts in the subbasin based on their respective cumulative lag times to the end of the subbasin. For detailed descriptions of the solution methods, see the Accounting Water Rights documentation **HERE (Accounting.pdf, Section 10.2.5.1)**.	

Comments	The calling rule is expected to make the assignments of the values to the slots.

Use Examples:
To cause all instream flow rights to compute their Available Allocatable Flow values:

```
FOREACH (LIST pair IN SolveWaterRights( "Network", "Allocatable Flow",
                                        @"20:00:00 January 1, 1800" )) DO
       ( pair<0> )[] = pair<1>
ENDFOREACH
```

Use Examples:
To cause all instream flow rights to make calls, using their already-computed Available Allocatable Flow slot values:

```
FOREACH (LIST pair IN SolveWaterRights( "Network", "Allocatable Flow",
                                        @"20:00:00 December 31, 2030")) DO
       ( pair<0> )[] = pair<1>
ENDFOREACH
```

Use Examples:
Or if lags are to be considered:

```
FOREACH (LIST triplet IN SolveWaterRightsWithLags( "Network",
                                              "Allocatable Flow",
                                              @"20:00:00 December 31, 2030")) DO
       ( triplet<0> )[triplet<1>] = triplet<2>
ENDFOREACH
```

170. Sort

Description	Sort the items in a list.	
Type	LIST	
Arguments		
1	LIST	a list of values to be sorted
Evaluation	Returns a list with the same values as the input list, in increasing order.	
Comments	Comparisons across type are defined by the following arbitrary ordering, on which users are advised against relying: BOOLEAN < NUMERIC < STRING < OBJECT < SLOT < DATETIME < LIST Within each type, ordering is as: • BOOLEAN: TRUE < FALSE • NUMERIC: values involving different dimensions are sorted by lexicographic ordering on the names of the units; within values of the same dimensionality, the sorting is based on standard numeric comparisons. • STRING: Lexicographic ordering • OBJECT: Lexicographic ordering on the object's name • SLOT: Lexicographic ordering on the slot's name • DATETIME: Same as RPL operator • LIST: Based on comparison of items within the list (left to right).	

Syntax Example:

```
Sort({ 1.0, {res1, 10}, "hello", 0.0, "bob"})
```

Syntax Example:

```
{ 0.0, 1.0, "bob", "hello", {res1, 10}}
```

171. SortPairsAscending, SortPairsDescending

Description	Sort a list of two-item lists.	
Type	LIST	
Arguments		
1	LIST {LIST}	the list of lists to be sorted.
Evaluation	The input list must be a list of lists, each member list must contain at least two items. The pairs are sorted into ascending/descending order by the second item's value, and a list containing the first items of this sorted list of pairs is returned. Duplicates are not removed.	
Comments	Comparisons across type are defined by the following arbitrary ordering, on which users are advised against relying: BOOLEAN < NUMERIC < STRING < OBJECT < SLOT < DATETIME < LIST Within each type, ordering is as: • BOOLEAN: TRUE < FALSE • NUMERIC: values involving different dimensions are sorted by lexicographic ordering on the names of the units; within values of the same dimensionality, the sorting is based on standard numeric comparisons. • STRING: Lexicographic ordering • OBJECT: Lexicographic ordering on the object's name • SLOT: Lexicographic ordering on the slot's name • DATETIME: Same as RPL operator • LIST: Based on comparison of items within the list (left to right).	

Syntax Example:

```
SortPairsAscending({{"a", 10.0} , {"b", 2.0}, {"c", 5.0}, {"d", 10.0}})
```

Return Example:

```
{"b", "c", "a", "d"}
```

172. SourceAccountAndObject

Description	Given a supply (specified by name), returns a list containing the source (upstream) account and object.	
Type	LIST {STRING, OBJECT}	
Arguments		
1	STRING	The name of the supply.
Evaluation		
Comments		

Syntax Example:

```
SourceAccountAndObject("ResA Fish to ReachB Fish")
```

Return Example:

```
{"Fish", %"ResA"}
```

173. Split

Description	Split a string up into component pieces.	
Type	LIST {STRING}	
Arguments		
1	STRING	the primary string
2	STRING	the separator string
Evaluation	This function returns a list of the strings which are contained within the primary string and separated by the separator string. Where ambiguity exists the left most occurrence of the separator string is used.	
Comments	It is an error for the separator string to be the empty string.	

Syntax Example:

```
Split("ABabcdefABcdABcdef", "AB") = { "", "abcdef", "cd", "cdef" }
Split("ResA^MyAccount.Inflow", "^") = { "ResA", "MyAccount.Inflow" }
```

174. StorageToArea

This function performs a lookup in a Reservoir object's **Elevation Volume Table** based on a given storage to determine the corresponding pool elevation. The function then uses this pool elevation for a lookup in the Reservoir's **Elevation Area Table** and evaluates to the corresponding area.

Description	Find the area of a given reservoir with a given storage.	
Type	NUMERIC	
Arguments		
1	OBJECT	reservoir object
2	NUMERIC	storage
Evaluation	The storage argument is looked up in the **Storage** column of the **Elevation Volume Table** of the reservoir object argument to determine the **Pool Elevation**. If the exact storage is not in the table, the lookup performs a linear interpolation between the two nearest bounding storages and their corresponding pool elevations. The pool elevation is then looked up in the **Pool Elevation** column of the **Elevation Area Table** to determine the **Surface Area**. If the exact elevation is not in the table, another linear interpolation is performed. The function evaluates to the computed surface area.	
Mathematical Expression	$pool\ elevation = elevation_{(lesser)} + \left(\frac{elevation_{(greater)} - elevation_{(lesser)}}{storage_{(greater)} - storage_{(lesser)}} \times (storage - storage_{(lesser)})\right)$ $area = area_{(lesser)} + \left(\frac{area_{(greater)} - area_{(lesser)}}{pool\ elevation_{(greater)} - pool\ elevation_{(lesser)}} \times (pool\ elevation - pool\ elevation_{(lesser)})\right)$	
Comments	If the object is not a reservoir or the reservoir does not have an **Elevation Area Table**, the function aborts the run with an error (**CRSSEvaporationCalc**, **DailyEvaporationCalc**, **PanAndIceEvaporation**, **heatBudgetEvaporation**, or **InputEvaporation** must be selected as the **Evaporation and Precipitation** Category selected Method). If the reservoir is a Slope Power Reservoir, the calculation is based only on level storage and does not include any wedge storage effects. This function will issue an error if the "Time Varying Elevation Volume" method, **HERE (Objects.pdf, Section 22.1.23.3)**, or "Time Varying Elevation Area" method, **HERE (Objects.pdf, Section 22.1.24.2)**,is selected. Instead, use the StorageToAreaAtDate function described next.	

Syntax Example:

```
StorageToArea(%"WattsBar", 442.39 "1000 cfsday")
```

Return Example:

```
12203.231 "m2"
```

175. StorageToAreaAtDate

This function performs a lookup in the Reservoir object's **Elevation Volume Table** or **Elevation Volume Table Time Varying** based on a given storage and datetime and computes the corresponding pool elevation. The function then uses this pool elevation for a lookup in the Reservoir's **Elevation Area Table** or **Elevation Area Table Time Varying** and evaluates to the corresponding surface area.

This function must be used when the **Time Varying Elevation Volume** method or **Time Varying Elevation Area** is selected. Otherwise, the StorageToArea function can be used and no DATETIME argument is required.

Description	Find the surface area corresponding to a reservoir's storage.	
Type	NUMERIC	
Arguments		
1	OBJECT	reservoir object
2	NUMERIC	storage
3	DATETIME	the datetime at which to do the conversion

<table>
<tr><td>Evaluation</td><td>
On the specified reservoir object argument, if the “Time Varying Elevation Volume” method is selected, HERE (Objects.pdf, Section 22.1.23.3), the function will reference the Elevation Volume Table Time Varying table. The function will select the appropriate column to use based on the datetime argument. On timesteps that exactly match a modification date, the previous column is used. The relationship changes at the end of that timestep and is taken into account when the reservoir dispatches. For this algorithm the previous timestep’s relationship is used.
Otherwise, the Elevation Volume Table is used.
Then, the storage argument is looked up in the appropriate storage column to determine the elevation from the Pool Elevation column. If the exact elevation is not in the table, the lookup performs a linear interpolation between the two nearest bounding elevations and their corresponding surface areas.On the specified reservoir object argument, if the “Time Varying Elevation Area” method is selected, HERE (Objects.pdf, Section 22.1.24.2), the function will reference the Elevation Area Table Time Varying table. The function will select the appropriate column to use based on the datetime argument. On timesteps that exactly match a modification date, the previous column is used. The relationship changes at the end of that timestep and is taken into account when the reservoir dispatches. For this algorithm the previous timestep’s relationship is used.
Otherwise, the Elevation Area Table is used.
Then, the computed pool elevation is looked up in the Pool Elevation column to determine the surface area from the appropriate column. If the exact elevation is not in the table, the lookup performs a linear interpolation between the two nearest bounding elevations and their corresponding surface areas.
</td></tr>
<tr><td>Mathematical Expression</td><td>

$$pool\ elevation = elevation_{(lesser)} + \left(\frac{elevation_{(greater)} - elevation_{(lesser)}}{storage_{(greater)} - storage_{(lesser)}} \times (storage - storage_{(lesser)})\right)$$

$$area = area_{(lesser)} + \left(\frac{area_{(greater)} - area_{(lesser)}}{pool\ elevation_{(greater)} - pool\ elevation_{(lesser)}} \times (pool\ elevation - pool\ elevation_{(lesser)})\right)$$

</td></tr>
<tr><td>Comments</td><td>If the object is not a reservoir, or the reservoir does not have an Elevation Volume Table or Elevation Volume Table Time Varying AND Elevation Area Table or Elevation Area Table Time Varying, the function aborts the run with an error.</td></tr>
</table>

Syntax Example:

```
StorageToAreaAtDate(%"Lake Mead", 10520217087.2 [m3], @”t”)
```

Return Example:

```
634547087.2 [m2]
```

176. StorageToElevation

This function performs a lookup in a Reservoir object’s **Elevation Volume Table** based on a given storage and evaluates to the corresponding pool elevation.

Description	Find the reservoir elevation at a given storage.	
Type	NUMERIC	
Arguments		
1	OBJECT	reservoir object
2	NUMERIC	storage
Evaluation	The storage argument is looked up in the **Storage** column of the **Elevation Volume Table** of the reservoir object argument to determine the **Pool Elevation**. If the exact storage is not in the table, the lookup performs a linear interpolation between the two nearest bounding storages and their corresponding elevation values.	
Mathematical Expression	$pool\ elevation = elevation_{(lesser)} + \frac{elevation_{(greater)} - elevation_{(lesser)}}{storage_{(greater)} - storage_{(lesser)}} \times (storage - storage_{(lesser)})$	
Comments	If the object is not a reservoir, the function aborts the run with an error. If the reservoir is a Slope Power Reservoir, the calculation is based only on level storage and does not include any wedge storage effects. This function will issue an error if the “Time Varying Elevation Volume” method, **HERE (Objects.pdf, Section 22.1.24.2)**, is selected. Instead, use the StorageToElevationAtDate function described next.	

Syntax Example:

```
StorageToElevation(%"WattsBar", 442.39 "1000 cfsday")
```

Return Example:

```
1792.25 "m"
```

177. StorageToElevationAtDate

This function performs a lookup in the Reservoir object's **Elevation Volume Table** or **Elevation Volume Table Time Varying** based on a given elevation and datetime and evaluates to the corresponding volume. This function must be used when the "Time Varying Elevation Volume" method is selected. Otherwise, the StorageToElevation function can be used and no DATETIME argument is required.

Description	Finds the elevation corresponding to a reservoir's storage.	
Type	NUMERIC	
Arguments		
1	OBJECT	reservoir object
2	NUMERIC	storage
3	DATETIME	the datetime at which to do the conversion
Evaluation	On the specified reservoir object argument, if the "Time Varying Elevation Volume" method is selected, HERE (Objects.pdf, Section 22.1.23.3), the function will reference the **Elevation Volume Table Time Varying** table. The function will select the appropriate column to use based on the datetime argument. On timesteps that exactly match a modification date, the previous column is used. The relationship changes at the end of that timestep and is taken into account when the reservoir dispatches. For this algorithm, the previous timestep's relationship is used. Otherwise, the **Elevation Volume Table** is used and the datetime is not used. Then, the storage argument is looked up in the appropriate **Storage** column to determine the **Pool Elevation**. If the exact elevation is not in the table, the lookup performs a linear interpolation between the two nearest bounding elevations and their corresponding storages.	
Mathematical Expression	$pool\ elevation = elevation_{(lesser)} + \frac{elevation_{(greater)} - elevation_{(lesser)}}{storage_{(greater)} - storage_{(lesser)}} \times (storage - storage_{(lesser)})$	
Comments	If the object is not a reservoir, or the reservoir does not have an **Elevation Volume Table** or **Elevation Volume Table Time Varying**, the function aborts the run with an error.	

Syntax Example:

```
StorageToElevationAtDate(%"Lake Mead", 634547087.2 [m3], @”t”)
```

Return Example:

```
1210.03 "ft"
```

178. Sum

This function sums a list of numbers.

Description	Sum a non-empty list of numbers.	
Type	NUMERIC	
Arguments		
1	LIST{NUMERIC}	a list of numeric values.
Evaluation	The numbers in the input list are added up, the total is returned.	
Mathematical Expression	$\text{total} = \sum_{x \in \text{input list}} x$	
Comments	If the input list is empty, one of the items in the list is not NUMERIC, or the unit types of items in the list are incompatible, this function aborts the run with an error.	

Syntax Example:

```
Sum({1.0 [cfs], 2.0 [cms]})
```

Return Example:

```
71.629333443 "cfs"
```

179. SumAccountSlotsByWaterType

This function sums the values of all accounting slots of a given name on accounts of a given water type.

Description	The sum of slots of a given name and water type.	
Type	NUMERIC	
Arguments		
1	OBJECT	the object on which to sum

2	STRING	the water type of accounts to sum
3	STRING	the name of the slots to sum
4	DATETIME	the date at which to sum
Evaluation	The function contains two nested loops. The outer loop iterates over all of the account types which may exist on the given object (Storage Account and/or Passthrough Account or Diversion Account). For each account type, a list is made of all of the accounts which are of the given water type. The inner loop iterates over all of these accounts and sums the values of the slots with the given name at the given time.	
Mathematical Expression	$total = \sum_{(account\ type,(account\ =\ water\ type))} object.slotname_{timestep}$	
Comments	If the object cannot accept accounts, has no accounts, or has no accounts of the given water type, this function aborts the run with an error. If none of the accounts of the given water type has a slot with the given name, or the slot is not a series slot, this function aborts the run with an error. Any slots which contain a NaN at the given datetime are ignored for the purpose of summation. If all of the slots contain NaNs at the given datetime, the function forces an early termination of the calling rule. If the given datetime argument does not land on an interval of the series slot, this function aborts the run with an error.	

Syntax Example:

```
SumAccountSlotsByWaterType(%"Heron Reservoir","RioGrande","storage",
@"t")
```

Return Example:

```
71629333.443 "m3"
```

180. SumByIndex

This function sums numbers at a given location within lists contained in a list.

Description	Given a list of lists and an index, sum the values at the given index in each list.	
Type	NUMERIC	
Arguments		

1	LIST{LIST}	a list of lists.
2	NUMERIC	an index.
Evaluation	All values located at the given index in each list contained within the input list are summed, the total is returned.	
Mathematical Expression	$\text{total} = \sum_{x \in \text{input list}} x_{\text{index}}$	
Comments	The input list must be non-empty. The index must be positive and a legal index for each of the lists contained within the values list. For example, if the index value is 3, the sublists must each contain at least 4 items. All items being summed must be numeric and have compatible dimensions. If any of these conditions is not met, this function aborts the run with an error.	

Syntax Example:

```
SumByIndex({true, 2.0 [cms]}, {false, 1.0 [cms]}, 1.0)
```

Return Example:

```
3.0 "cms"
```

181. SumFlowsToVolume and SumFlowsToVolumeSkipNaN

This function sums a series slot's FLOW values between a starting timestep and ending timestep and evaluates to the corresponding volume of water.

Description	The volume equivalent of flows over time.	
Type	NUMERIC	
Arguments		
1	SLOT	the series or periodic slot to sum
2	DATETIME	the start date
3	DATETIME	the end date
Evaluation	The function loops through all of the slot values between the start and end datetime arguments. For each value, the flow is multiplied by the corresponding timestep's length to convert it to a volume before adding it to the previous result. The function evaluates to the final result in units of "m3".	

Mathematical Expression	$volume = \sum_{start\ datetime - end\ datetime} (flow_{(slot)} \times \Delta t_{timestep})$
Comments	If the unit type of the slot argument is not FLOW or the starting or ending datetime argument is not defined in the series slot, this function aborts the run with an error. For the SumFlowsToVolume function, if one of the slot values in the time range is a NaN, the function forces an early termination of the calling rule. The "SkipNaN" variation treats an invalid value (NaN) as 0.0. For periodic slots, the dates used are those within the range and falling on a run timestep and the column used is the first column.

Syntax Example:

```
SumFlowsToVolume(Crystal.Inflow, @"January, 1999", @"September, 1999")
```

Return Example:

```
12.3023 "m3"
```

182. SumFlowsToVolumeByCol and SumFlowsToVolumeByColSkipNaN

Description	This function sums a a column of a slot's FLOW values between a starting timestep and ending timestep and evaluates to the corresponding volume of water	
Type	NUMERIC	
Arguments		
1	SLOT	the Agg Series Slot or periodic slot to sum
2	DATETIME	the start date
3	DATETIME	the end date
4	NUMERIC	the column (interpreted as a 0-based integral index)
Evaluation	The function loops through all of the slot values of the given column between the start and end datetime arguments. For each value, the flow is multiplied by the corresponding timestep's length to convert it to a volume before adding it to the previous result. The function evaluates to the final result in units of "m3".	
Mathematical Expression	$volume = \sum_{start\ datetime - end\ datetime} (flow_{(slot)} \times \Delta t_{timestep})$	
Comments	If the unit type of the given column of the slot is not FLOW or if the slot is an Agg Series Slot and the starting or ending datetime argument is not defined in the slot, this function aborts the run with an error. For the SumFlowsToVolumeByCol function, if one of the slot values in the time range is a NaN, the function forces an early termination of the calling rule. The "SkipNaN" variation treats an invalid value (NaN) as 0.0. For periodic slots, the dates used are those within the range and falling on a run timestep.	

Syntax Example:

```
SumFlowsToVolumeByCol(Data.Coeff, @"January, 1999", @"September, 1999", 1)
```

Return Example:

```
12.231 "m3"
```

183. SumObjectsAggregatedOverTime

This function returns a single numeric value obtained by summing several objects' aggregated slot values. The objects' slot values may be aggregated as a SUM, AVG, MIN, or MAX over a specified time range.

Description	Sum several object's values, each of which is the result of aggregating a slot's values over time.	
Type	NUMERIC	
Arguments		
1	STRING	subbasin name
2	STRING	slot name
3	STRING	aggregation function ("SUM", "AVG", "MIN", or "MAX")
4	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
5	BOOLEAN	time conversion option ("TRUE" or "FALSE")
6	DATETIME	start date
7	DATETIME	end date
Evaluation	A list of slots is generated by searching all of the objects in the subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values to be aggregated are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, each slot's values are aggregated according to the aggregation function argument over the time range of the datetime arguments. During each of these slot aggregations, any values which do not satisfy the aggregation filter argument are ignored. Finally, all of the object's aggregated slot values are summed	
Mathematical Expression	$\sum_{obj \text{ in } subbasin} (\forall_{(t \text{ from } start \text{ to } end)}[AggFunction_{(obj)}(obj.slotname)])$	
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, RiverWare aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.	

Syntax Example:

```
SumObjectsAggregatedOverTime("upper basin", "Inflow", "MAX","ALL", TRUE
@"October, Previous Year",
@"September, Current Year")
```

Return Example:

```
234.3 "cms"
```

184. SumObjectsAtEachTimestep

This function evaluates to a list. Each item of the list is a list comprised of the datetime at which the summation was performed and the value of the sum.

Description	Sum several object's slot values, for each timestep in a range.	
Type	LIST{LIST{DATETIME, NUMERIC}}	
Arguments		
1	STRING	subbasin name
2	STRING	slot name
3	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
4	BOOLEAN	time conversion option ("TRUE" or "FALSE")
5	DATETIME	start date
6	DATETIME	end date
Evaluation	A list of slots is generated by searching all of the objects in the subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values to be summed are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, all of the object's slot values are summed, yielding one value for each timestep in the time range of the datetime arguments. The function returns a list of two items, where the first and second items of the inner lists are the datetime and the summation value, respectively.	
Mathematical Expression	$\forall_{(t \text{ from } start \text{ to } end)} \left[\{t, \sum_{obj \text{ in } subbasin} obj.slotname\} \right]$	

Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, RiverWare aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.

Syntax Example:

```
SumObjectsAtEachTimestep("upper basin", "Storage", "ALL", FALSE
@"October, Previous Year",
@"November, Previous Year")
```

Return Example:

```
{ {@"October 31, 2003", 32303.2"m3"} , {@"November 30, 2003", 43232.2"m3"}}
```

185. SumSlot and SumSlotSkipNaN

This function sums a series slot’s values between a starting timestep and ending timestep.

Description	The sum of a slot’s values over time.	
Type	NUMERIC	
Arguments		
1	SLOT	the series or periodic slot to sum
2	DATETIME	the start date
3	DATETIME	the end date
Evaluation	The function loops through all of the slot values between the start and end datetime arguments. Each value is added to the previous result.	
Mathematical Expression	$volume = \sum_{start\ datetime - end\ datetime} slot\ value_t$	
Comments	If the starting or ending datetime argument is not defined in the series slot, this function aborts the run with an error. For the SumSlot function, if one of the slot values in the time range is a NaN, the function forces an early termination of the calling rule. The "SkipNaN" variation treats an invalid value (NaN) as 0.0. For periodic slots, the dates used are those in the first column within the range and falling on a run timestep.	

Syntax Example:

```
SumSlot(Crystal.Inflow, @"January 1, 1999", @"September 30, 1999")
```

Return Example:

```
32.47 "cms"
```

186. SumSlotByCol and SumSlotByColSkipNaN

Description	The sum the values in a column of a slot over a range of time.	
Type	NUMERIC	
Arguments		
1	SLOT	the Agg Series Slot or periodic slot to sum
2	DATETIME	the start date
3	DATETIME	the end date
4	NUMERIC	the column (interpreted as a 0-based integral index)
Evaluation	The function loops through all of the slot values in the given column between the start and end datetime arguments. Each value is added to the previous result.	
Mathematical Expression	$volume = \sum_{start\ datetime - end\ datetime} slot\ value_t$	
Comments	If the slot is an Agg Series Slot and the starting or ending datetime argument is not defined in the slot, this function aborts the run with an error. For the SumSlotByCol function, if one of the slot values in the time range is a NaN, the function forces an early termination of the calling rule. The "SkipNaN" variation treats an invalid value (NaN) as 0.0. For periodic slots, the dates used are those within the range and falling on a run timestep.	

Syntax Example:

```
SumSlotByCol(Data.Coeff, @"January 1, 1999", @"September 30, 1999", 2)
```

Return Example:

```
25.323
```

187. SumTableColumn

This function evaluates to the sum of a table slot's values, in the given column, from the given start row to the given end row.

Description	Sum of the values of a table slot column between two rows.	
Type	NUMERIC	
Arguments		
1	SLOT	the table slot whose values to sum
2	NUMERIC	column
3	NUMERIC	start row
4	NUMERIC	end row
Evaluation	The function loops over each value in the given column of the given table slot, beginning with the start row and ending with the end row (inclusive). Each value is added to the previous sum. The function evaluates to this sum. Rows and columns are numbered beginning with zero.	
Mathematical Expression	$NUMERIC = \sum_{start\ row - end\ row} slot\ value_{(row,\ column)}$	
Comments	Units are not required for row and column indices and, if provided, will be ignored. If the column, start row, or end row do not exist in the slot or if the start row is greater than the end row, this function aborts the run with an error. If one of the slot values within the desired time range is a NaN, the function exits the rule with an early termination.	

Syntax Example:

```
SumTableColumn(Chickamauga Data.Flow, 0, 0, 1)
```

Return Example:

```
13.95
```

188. SumTableRow

This function evaluates to the sum of a table slot's values, in the given row, from the given start column to the given end column.

Description	Sum of the values of a table slot row between two columns.

Type	NUMERIC	
Arguments		
1	SLOT	the table slot whose values to sum
2	NUMERIC	row
3	NUMERIC	start column
4	NUMERIC	end column
Evaluation	The function loops over each value in the given row of the given table slot, beginning with the start column and ending with the end column (inclusive). Each value is added to the previous sum. The function evaluates to this sum. Rows and columns are numbered beginning with zero.	
Mathematical Expression	$NUMERIC = \sum_{start\ column-end\ column} slot\ value_{(row,\ column)}$	
Comments	Units are not required for row and column indices and, if provided, will be ignored. If the row, start column, or end column do not exist in the slot or if the start column is greater than the end column, this function aborts the run with an error. If one of the slot values within the desired time range is a NaN, the function exits the rule with an early termination.	

Syntax Example:

```
SumTableRow(Chickamauga Data.units, 0, 0, 1)
```

Return Example:

```
13.95
```

189. SumTimestepsAggregatedOverObjects

This function evaluates to a single numeric value, which is the sum over time of values resulting from aggregating several objects slot values at each timestep.

Description	Sum of a timeseries of values, each of which is the result of aggregating several objects’ slot values.	
Type	NUMERIC	
Arguments		
1	STRING	Subbasin name
2	STRING	slot name
3	STRING	aggregation function ("SUM", "AVG", "MIN", or "MAX")

4	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
5	BOOLEAN	time conversion option ("TRUE" or "FALSE")
6	DATETIME	start datetime
7	DATETIME	end datetime
Evaluation	A list of slots is generated by searching all of the objects in the subbasin argument for slots which match the slot name argument. If the time conversion option argument is TRUE, and the values to be aggregated are of the FLOW unit type, the values are multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME. Next, all of the objects' slot values are aggregated according to the aggregation function argument for each timestep in the time range of the datetime arguments. During each of these slot aggregations, any values which do not satisfy the aggregation filter argument are ignored. Finally, the timeseries of object aggregated slot values are summed.	
Mathematical Expression	$\sum_{t \text{ from } start \text{ to } end} \forall_{(obj \text{ in } subbasin)}[AggFunction_{(t)}(obj.slotname)]$	
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, RiverWare aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, the "SUM" aggregation function yields an aggregated value of 0.0 for that slot, while the "AVG", "MIN", and "MAX" aggregation functions abort RiverWare with an error.	

Syntax Example:

```
SumTimestepsAggregatedOverObjects("upper basin","Inflow","SUM","ALL",
FALSE, @"October, Previous Year",
@"September, Current Year")
```

Return Example:

```
133.43 "cms"
```

190. SumTimestepsForEachObject

This function evaluates to a list. Each item of the list is a list comprised of the object name and the sum of the slot values on that object for the time range specified.

Description	Sum a slot's values over a time range, for each object in a subbasin.
Type	LIST {LIST {OBJECT, NUMERIC}}

Arguments		
1	STRING	Subbasin name
2	STRING	slot name
3	STRING	aggregation filter ("INPUT", "OUTPUT", or "ALL")
4	BOOLEAN	time conversion option ("TRUE" or "FALSE")
5	DATETIME	start datetime
6	DATETIME	end datetime
Evaluation	A list of slots is generated by searching all of the objects in the Subbasin argument for slots which match the slot name argument. For each object, the slot's values over every timestep in the range of the datetime arguments are summed. Any values which do not satisfy the aggregation filter argument are ignored during the calculation. If the time conversion option argument is TRUE, and the values to be summed are of the FLOW unit type, the values are first multiplied by their corresponding timestep length to convert them to values of the unit type VOLUME.	
Mathematical Expression	$\forall_{(obj \text{ in } subbasin)} \left[\{obj, \sum_{(t \text{ from } start \text{ to } end)} obj.slotname\} \right]$	
Comments	If the time conversion option argument is TRUE, but the unit of the slot values is not FLOW, this function aborts the run with an error. If none of the values for a slot satisfy the aggregation filter argument, this function also aborts RiverWare with an error.	

Syntax Example:

```
SumTimestepsForEachObject("upper basin", "Inflow", "ALL", TRUE,
@"October, Previous Year",
@"September, Current Year")
```

Return Example:

```
{ {%"Res1", 12.23"cms"}, {%"Reach2", 4.92 "cms"}, {%"Res2", 23.2 "cms"} }
```

191. SupplyAttributes

Description	Given a supply (specified by name), returns a list containing the supply's attributes, i.e., the supply's release type and destination.	
Type	LIST {STRING, STRING}	
Arguments		
1	STRING	The name of the supply.
Evaluation		
Comments		

Syntax Example:

```
SupplyAttributes("ResA One to ResB Two")
```

Return Example:

```
{"IrrigationWater", "FarmerB"}
```

192. SupplyNamesFrom, SupplyNamesFrom1to1

Description	This function returns a list of names of Supplies which represent outflows from given Accounts and which have the indicated ReleaseType and Destination.	
Type	LIST {STRING}	
Arguments		
1	LIST { LIST { OBJECT, STRING } }	Source List: A List of pairs (represented as Lists) containing an Object and an Account Name of an Account on that Object.
2	STRING	ReleaseType name or "NONE" or "ALL"
3	STRING	Destination name or "NONE" or "ALL"
Evaluation	A temporary list of Accounts is created from the Source List. For each of those Accounts, we examine the outflow Supplies which (1) link an Account on a different downstream Object, and (2) have the indicated ReleaseType, and (3) have the indicated Destination In the case of SupplyNamesFrom, for each of these Accounts being considered, the names of all related Supplies matching the criteria are added to the returned List. In the case of SupplyNamesFrom1to1, there should be zero or one matching Supplies: (1) If there are no Supplies matching the criteria, then an empty string ("") is added to the returned List, or (2) If there is exactly ONE Supply matching the criteria, then the name of that Supply is added to the returned List, or (3) If there is more than one Supply matching the criteria, then an error is generated. In this way, the list returned by SupplyNamesFrom is guaranteed to contain exactly one string for each Account in the Source List. If the ReleaseType argument or the Destination argument is "NONE," then only Supplies having the default (unassigned) attribute of that type are considered. If the ReleaseType or the Destination argument is "ALL," then that Supply attribute is ignored.	

Syntax Example:

```
SupplyNamesFrom({{%"ResA","One"}, {%"ResB","One"}}, "Account Fill", "Abiquiu")
```

Return Example:

```
SupplyNamesFrom: {"ResA One to ResB One", "ResB One to ResB Three"}
SupplyNamesFrom1to1: {"ResA One to ResB One"}
```

193. SupplySlotsFrom, SupplySlotsFrom1to1

Description	This function returns a list of Supply slots of Supplies which represent outflows from given Accounts and which have the indicated ReleaseType and Destination.	
Type	LIST {SLOT}	
Arguments		
1	LIST { LIST { OBJECT, STRING } }	Source List: A List of pairs (represented as Lists) containing an Object and an Account Name of an Account on that Object.
2	STRING	ReleaseType name or "NONE" or "ALL"
3	STRING	Destination name or "NONE" or "ALL"
Evaluation	A temporary list of Accounts is created from the Source List. For each of those Accounts, we examine the outflow Supplies which (1) link an Account on a different downstream Object, and (2) have the indicated ReleaseType, and (3) have the indicated Destination In the case of SupplySlotsFrom, for each of these Accounts being considered, the Supply slots of all related Supplies matching the criteria are added to the returned List. In the case of SupplySlotsFrom1to1, there should be one matching Supply: (1) If there are no Supplies matching the criteria, or more than one, then an error is generated. (2) If there is exactly ONE Supply matching the criteria, then the Supply slot of that Supply is added to the returned List. In this way, the list returned by SupplySlotsFrom1to1 is guaranteed to contain exactly one slot for each Account in the Source List. If the ReleaseType argument or the Destination argument is "NONE," then only Supplies having the default (unassigned) attribute of that type are considered. If the ReleaseType argument or the Destination argument is "ALL," then that Supply attribute is ignored.	

Syntax Example:

```
SupplySlotsFrom({{%"ResA","One"},
                 {%"ResB","Two"}}, "Account Fill", "Abiquiu")
```

Return Example:

```
SupplySlotsFrom: {$"ResA One to ResB One.Supply", $"ResA One to ResB Two.Supply"}
SupplySlotsFrom1to1: {$"ResA One to ResB One.Supply"}
```

194. SupplyNamesFromIntra, SupplyNamesFromIntra1to1

<table>
<tr><th>Description</th><td colspan="2">This function returns a list of names of Supplies which represent internal flows (i.e. Transfer supplies) from given Accounts and which have the indicated ReleaseType and Destination.</td></tr>
<tr><th>Type</th><td colspan="2">LIST {STRING}</td></tr>
<tr><th>Arguments</th><td colspan="2"></td></tr>
<tr><th>1</th><td>LIST { LIST { OBJECT, STRING } }</td><td>Source List: A List of pairs (represented as Lists) containing an Object and an Account Name of an Account on that Object.</td></tr>
<tr><th>2</th><td>STRING</td><td>ReleaseType name or "NONE" or "ALL"</td></tr>
<tr><th>3</th><td>STRING</td><td>Destination name or "NONE" or "ALL"</td></tr>
<tr><th>Evaluation</th><td colspan="2">A temporary list of Accounts is created from the Source List. For each of those Accounts, we examine the outflow Supplies which:
(1) link an Account on the SAME Object, and
(2) have the indicated ReleaseType, and
(3) have the indicated Destination
In the case of SupplyNamesIntra, for each of these Accounts being considered, the names of all related Supplies matching the criteria are added to the returned List. In the case of SupplyNamesIntra1to1, there should be zero or one matching Supplies:
(1) If there are no Supplies matching the criteria, then an empty string ("") is added to the returned List, or
(2) If there is exactly ONE Supply matching the criteria, then the name of that Supply is added to the returned List, or
(3) If there is more than one Supply matching the criteria, then an error is generated.
In this way, the list returned by SupplyNamesIntra is guaranteed to contain exactly one string for each Account specified in the Source List.
If the ReleaseType argument or the Destination argument is "NONE," then only Supplies having the default (unassigned) attribute of that type are considered.
If the ReleaseType or the Destination argument is "ALL," then that Supply attribute is ignored.</td></tr>
<tr><th>Comments</th><td colspan="2">ReleaseTypes and Destinations are properties of Supplies.</td></tr>
</table>

Syntax Example:

```
SupplyNamesFromIntra({{%"ResA","One"}, {%"ResA","Two"}}, "Account Fill",
```

```
"Abiquiu")
```

Return Example:

```
SupplyNamesFromIntra: {"ResA One to ResA Two", "ResA Two to ResA Three"}
SupplyNamesFromIntra1to1: {"ResA One to ResA Two"}
```

195. SupplySlotsFromIntra, SupplySlotsFromIntra1to1

Description	This function returns a list of Supply slots of Supplies which represent internal flows (i.e. Transfer supplies) from given Accounts and which have the indicated ReleaseType and Destination.	
Type	LIST {SLOT}	
Arguments		
1	LIST { LIST { OBJECT, STRING } }	Source List: A List of pairs (represented as Lists) containing an Object and an Account Name of an Account on that Object.
2	STRING	ReleaseType name or "NONE" or "ALL"
3	STRING	Destination name or "NONE" or "ALL"
Evaluation	A temporary list of Accounts is created from the Source List. For each of those Accounts, we examine the outflow Supplies which: (1) link an Account on the SAME Object, and (2) have the indicated ReleaseType, and (3) have the indicated Destination In the case of SupplySlotsIntra, for each of these Accounts being considered, the Supply slots of all related Supplies matching the criteria are added to the returned List. In the case of SupplySlotsIntra1to1, there should be one matching Supply: (1) If there are no Supplies matching the criteria, or more than one, then an error is posted. (2) If there is exactly ONE Supply matching the criteria, then the Supply slot of that Supply is added to the returned List. In this way, the list returned by SupplySlotsIntra1to1 is guaranteed to contain exactly one slot for each Account in the input list. If the ReleaseType argument or the Destination argument is "NONE," then only Supplies having the default (unassigned) attribute of that type are considered. If the ReleaseType argument or the Destination argument is "ALL," then that Supply attribute is ignored.	
Comments		

Syntax Example:

```
SupplySlotsFromIntra({{%"ResA","One"}, {%"ResA","Two"}}, "Account Fill",
"Abiquiu")
```

Return Example:

```
SupplySlotsFromIntra: {$"ResA One to ResA Two.Supply", $"ResA Two to ResA
                        Three.Supply"}
SupplySlotsFromIntra1to1: {$"ResA One to ResA Two.Supply"}
```

196. SupplyNamesTo, SupplyNamesTo1to1

Description	This function returns a list of names of Supplies which represent inflows to given Accounts and which have the indicated ReleaseType and Destination.	
Type	LIST {STRING}	
Arguments		
1	LIST { LIST { OBJECT, STRING } }	Source List: A List of pairs (represented as Lists) containing an Object and an Account Name of an Account on that Object.
2	STRING	ReleaseType name or "NONE" or "ALL"
3	STRING	Destination name or "NONE" or "ALL"
Evaluation	A temporary list of Accounts is created from the Source List. For each of those Accounts, we examine the inflow Supplies which (1) link an Account on a different upstream Object, and (2) have the indicated ReleaseType, and (3) have the indicated Destination In the case of SupplyNamesTo, for each of these Accounts being considered, the names of all related Supplies matching the criteria are added to the returned List. In the case of SupplyNamesTo1to1, there should be zero or one matching Supplies: (1) If there are no Supplies matching the criteria, then an empty string ("") is added to the returned List, or (2) If there is exactly ONE Supply matching the criteria, then the name of that Supply is added to the returned List, or (3) If there is more than one Supply matching the criteria, then an error is generated. In this way, the list returned by SupplyNamesTo is guaranteed to contain exactly one string for each Account in the Source List. If the ReleaseType argument or the Destination argument is "NONE," then only Supplies having the default (unassigned) attribute of that type are considered. If the ReleaseType argument or the Destination argument is "ALL," then that Supply attribute is ignored.	
Comments		

Syntax Example:

```
SupplyNamesTo({{%"ResA","One"}, {%"ResB","Two"}}, "Account Fill", "Abiquiu")
```

Return Example:

```
SupplyNamesTo: {"ReachA One to ResA One", "Reach A Two to ResB Two"}
```

197. SupplySlotsTo, SupplySlotsTo1to1

Description	This function returns a list of Supply slots of Supplies which represent inflows to given Accounts and which have the indicated ReleaseType and Destination.	
Type	LIST {SLOT}	
Arguments		
1	LIST { LIST { OBJECT, STRING } }	Source List: A List of pairs (represented as Lists) containing an Object and an Account Name of an Account on that Object.
2	STRING	ReleaseType name or "NONE" or "ALL"
3	STRING	Destination name or "NONE" or "ALL"
Evaluation	A temporary list of Accounts is created from the Source List. For each of those Accounts, we examine the inflow Supplies which (1) link an Account on a different upstream Object, and (2) have the indicated ReleaseType, and (3) have the indicated Destination In the case of SupplySlotsTo, for each of these Accounts being considered, the Supply slots of all related Supplies matching the criteria are added to the returned List. In the case of SupplySlotsTo1to1, there should be one matching Supply: (1) If there are no Supplies matching the criteria, or more than one, then an error is generated. (2) If there is exactly ONE Supply matching the criteria, then the Supply slot of that Supply is added to the returned List. In this way, the list returned by SupplySlotsTo1to1 is guaranteed to contain exactly one slot for each Account in the Source List. If the ReleaseType argument or the Destination argument is "NONE," then only Supplies having the default (unassigned) attribute of that type are considered. If the ReleaseType argument or the Destination argument is "ALL," then that Supply attribute is ignored.	
Comments		

Syntax Example:

```
SupplySlotsTo({{%"ResA","One"}, {%"ResB","Two"}}, "Account Fill", "Abiquiu")
```

Return Example:

```
{$"ReachA One to ResA One.Supply", $"Reach A Two to ResB Two.Supply"}
```

198. SupplyNamesToIntra, SupplyNamesToIntra1to1

Description	This function returns a list of names of Supplies which represent internal flows (i.e. Transfer supplies) to given Accounts and which have the indicated ReleaseType and Destination.	
Type	LIST {STRING}	
Arguments		
1	LIST { LIST { OBJECT, STRING } }	Source List: A List of pairs (represented as Lists) containing an Object and an Account Name of an Account on that Object.
2	STRING	ReleaseType name or "NONE" or "ALL"
3	STRING	Destination name or "NONE" or "ALL"
Evaluation	A temporary list of Accounts is created from the Source List. For each of those Accounts, we examine the inflow Supplies which: (1) link an Account on the SAME Object, and (2) have the indicated ReleaseType, and (3) have the indicated Destination In the case of SupplyNamesToIntra, for each of these Accounts being considered, the names of all related Supplies matching the criteria are added to the returned List. In the case of SupplyNamesToIntra1to1, there should be zero or one matching Supplies: (1) If there are no Supplies matching the criteria, then an empty string ("") is added to the returned List, or (2) If there is exactly ONE Supply matching the criteria, then the name of that Supply is added to the returned List, or (3) If there is more than one Supply matching the criteria, then an error is generated. In this way, the list returned by SupplyNamesToIntra is guaranteed to contain exactly one string for each Account specified in the Source List. If the ReleaseType argument or the Destination argument is "NONE," then only Supplies having the default (unassigned) attribute of that type are considered. If the ReleaseType or the Destination argument is "ALL," then that Supply attribute is ignored.	
Comments	ReleaseTypes and Destinations are properties of Supplies.	

Syntax Example:

```
SupplyNamesToIntra({{%"ResA","One"}, {%"ResA","Two"}}, "Account Fill",
"Abiquiu")
```

Return Example:

```
SupplyNamesToIntra: {"ResA One to ResA Two", "ResA Three to ResA Two"}
```

199. SupplySlotsToIntra, SupplySlotsToIntra1to1

<table>
<tr><th>Description</th><td colspan="2">This function returns a list of Supply slots of Supplies which represent internal flows (i.e. Transfer supplies) to given Accounts and which have the indicated ReleaseType and Destination.</td></tr>
<tr><th>Type</th><td colspan="2">LIST {SLOT}</td></tr>
<tr><th>Arguments</th><td colspan="2"></td></tr>
<tr><th>1</th><td>LIST { LIST { OBJECT, STRING } }</td><td>Source List: A List of pairs (represented as Lists) containing an Object and an Account Name of an Account on that Object.</td></tr>
<tr><th>2</th><td>STRING</td><td>ReleaseType name or "NONE" or "ALL"</td></tr>
<tr><th>3</th><td>STRING</td><td>Destination name or "NONE" or "ALL"</td></tr>
<tr><th>Evaluation</th><td colspan="2">A temporary list of Accounts is created from the Source List. For each of those Accounts, we examine the inflow Supplies which:
(1) link an Account on the SAME Object, and
(2) have the indicated ReleaseType, and
(3) have the indicated Destination
In the case of SupplySlotsToIntra, for each of these Accounts being considered, the Supply slots of all related Supplies matching the criteria are added to the returned List. In the case of SupplySlotsToIntra1to1, there should be one matching Supply:
(1) If there are no Supplies matching the criteria, or more than one, then an error is generated.
(2) If there is exactly ONE Supply matching the criteria, then the Supply slot of that Supply is added to the returned List.
In this way, the list returned by SupplySlotsToIntra1to1 is guaranteed to contain exactly one slot for each Account in the input list.
If the ReleaseType argument or the Destination argument is "NONE," then only Supplies having the default (unassigned) attribute of that type are considered. If the ReleaseType argument or the Destination argument is "ALL," then that Supply attribute is ignored.</td></tr>
</table>

Syntax Example:

```
SupplySlotsToIntra({{%"ResA","One"}, {%"ResA","Two"}}, "Account Fill",
"Abiquiu")
```

Return Example:

```
SupplySlotsToIntra: {$"ResA One to ResA Two.Supply", $"ResA Three to ResA
                        Two.Supply"}
```

200. TableInterpolation

This function performs a lookup in a given table slot, based on a given value in a given column, and evaluates to the corresponding value in the other given column.

Description	Table lookup with linear interpolation.	
Type	NUMERIC	
Arguments		
1	SLOT	table slot in which to do lookup
2	NUMERIC	"from" column
3	NUMERIC	"to" column
4	NUMERIC	value in "from" column
5	DATETIME	datetime context for unit conversions
Evaluation	The value argument is looked up in the "from" column of the given table slot to determine the corresponding value in the "to" column. If the exact value is not in the table, the lookup performs a linear interpolation between the two nearest bounding values in the "from" column and their corresponding values in the "to" column.	
Mathematical Expression	$to\ value = to\ value_{(lesser)} + \frac{to\ value_{(greater)} - to\ value_{(lesser)}}{from\ value_{(greater)} - from\ value_{(lesser)}}(from\ value - from\ value_{(lesser)})$	
Comments	If the given slot is not a table slot or if the "from" column or "to" column does not exist in the table, the function aborts the run with an error. Column numbers are zero based with a unit type of NONE. If the "from" value is not the same unit type as the "from" column, or the "from" value is not between the first and last value of the "from" column, the function aborts the run with an error.	

Syntax Example:

```
TableInterpolation(Lake Mead.Elevation Volume Table, 0, 1,
1210.03 "ft", @"t")
TableInterpolation(Mead.Evaporation Table,
GetColumnIndex(Mead.Evaporation Table, "Julian Day"),
1, 1210.03 "ft", @"t")
```

Return Example:

```
234342422.32 "m"
```

201. TableInterpolation3D

This function performs a double interpolation using two columns of data, and a value for each column, to compute the corresponding value in a third column of data. The data in the two columns used for the double interpolation must be in ascending order.

Description	Table lookup with double linear interpolation.	
Type	NUMERIC	
Arguments		
1	SLOT	Table slot on which to do the lookup
2	NUMERIC	This is the column number (zero based) corresponding to the first column of data in the table - the data to use for the outer/first interpolation
3	NUMERIC	The value to use for the first column
4	NUMERIC	This is the column number (zero based) corresponding to the second column of data in the table - the data to use for the inner/second interpolation
5	NUMERIC	The value to use for the second column
6	NUMERIC	This is the column number of the third column of data in the table - where the answer will be computed
7	DATETIME	datetime context for unit conversions
Evaluation		
Mathematical Expression		

Comments	If the given slot is not a table slot or if the columns do not exist in the table, the function aborts the run with an error. Column numbers are zero based with a unit type of NONE. If a value is not the same unit type as the values in the corresponding column, the function aborts the run with an error. If the values in the table do not encompass the values passed into the function, the function aborts the run with an error. Also, the values in the columns used for both the inner and outer interpolations MUST BE IN ASCENDING ORDER.

Syntax Example:

```
TableInterpolation3D(LowerReach.Interpolated GainLoss Table, 0,
GetDayOfYear(@"t"), 1, LowerReach.Inflow[], 2, @"t")
```

Return Example:

```
1.344
```

202. TableLookup

Description	Table lookup to nearest value.	
Type	NUMERIC	
Arguments		
1	SLOT	table slot in which to do lookup
2	NUMERIC	"from" column
3	NUMERIC	"to" column
4	NUMERIC	value in "from" column
5	DATETIME	datetime context for unit conversions
5	BOOLEAN	whether to round up (or down)
Evaluation	The value argument is looked up in the "from" column of the given table slot to determine the corresponding value in the "to" column. If the round up argument is true and exact value is not in the table, the lookup finds the row whose value is the smallest value larger than the lookup value and returns the value in that row's "to" column. If the round up argument is false and the exact value is not in the table, the lookup finds the row whose value is the largest value smaller than the lookup value and returns the value in that row's "to" column.	
Comments	Error conditions which will lead to an error diagnostic and cause the run to halt include: the given slot is not a table slot, one of the two column indices are not valid for the table, the lookup value has units which are inconsistent with the from column of the table, or the search value is not contained within the table. Column numbers are zero based with a unit type of NONE.	

Syntax Example:

```
TableLookup(Lake Mead.Elevation Volume Table, 0, 1,
            1210.0 "ft", @"t", TRUE)
```

Return Example:

```
1323342 "acre-feet"
```

203. TargetHWGivenInflow

This function computes the outflow required to meet a specified Pool Elevation at a specified future timestep. It performs a lumped mass balance across several timesteps specified as a target range. This function only works with Storage Reservoirs and Level Power Reservoirs.

Description	Computes the outflow required to meet a target pool elevation	
Type	NUMERIC	
Arguments		
1	OBJECT	the reservoir on which to perform the calculation
2	DATETIME	the target begin date
3	DATETIME	the target date (target end date)
4	NUMERIC	the target pool elevation value
5	NUMERIC	the total inflow volume over the target range
6	NUMERIC	the previous storage value (before the target begin date)
Evaluation	This function takes the target pool elevation value and converts it to a storage. This is the storage value desired at the target date. The difference between the target storage and the previous storage is the change in storage over the target range. Since the total inflow volume over the target range is given as an argument, the total outflow volume can be computed. Side flows are automatically included in the mass balance computation and thus should not be included in the inflow value provided in Argument 5. These include **Hydrologic Inflow Net, Diversion, Return Flow, Canal Flow, Flow FROM Pumped Storage,** and **Flow TO Pumped Storage**. If any of the side flow slots contain NaN on one of the target period timesteps, the value will be assumed to be zero on that timestep. Other sources and sinks, such as **Evaporation**, **Precipitation**, **Bank Storage** and **Seepage** are NOT included in the mass balance computation. The total outflow volume is then converted to a flow rate and divided by the number of timesteps in the target range. The result is a single timestep outflow value. This value needs to be set on the outflow slot for every timestep in the target range in order to meet the target pool elevation.	
Mathematical Expression	$\text{Outflow Volume} = \text{Previous Storage} - \text{Target Storage} + \text{Inflow Volume} + \text{Side Flows}$	

Comments	This function is intended to be used with Storage Reservoirs and Level Power Reservoirs. If using a Slope Power Reservoir, the TargetSlopeHWGivenInflow function should be used. Target operations cannot be done on a Pump Storage Reservoir. This function will issue an error if the “Time Varying Elevation Volume” method, **HERE (Objects.pdf, Section 22.1.24.2)**, is selected and the target begin and end dates bound a table modification date.

Syntax Example:

```
TargetHWGivenInflow(Lake Mead, @"24:00 January 1, 2002",
@"24:00 January 5, 2002", 1200 "ft", 70,000 "acre-ft",
Lake Mead.Storage[@"24:00 December 31, 2002")
```

Return Example:

```
23.43 "cms"
```

204. TargetSlopeHWGivenInflow

This function computes the outflow required to meet a specified Pool Elevation at a specified future timestep. It performs a lumped mass balance across several timesteps specified as a target range. This function should be used with Slope Power Reservoirs.

Description	Computes the outflow required to meet a target pool elevation	
Type	NUMERIC	
Arguments		
1	OBJECT	the reservoir on which to perform the calculation
2	DATETIME	the target begin date
3	DATETIME	the target date (target end date)
4	LIST	the inflow values over the target range (there should be one value for each date in the target range)
5	NUMERIC	the target pool elevation value
6	NUMERIC	the previous pool elevation value (before the target begin date)
7	NUMERIC	the previous storage value (before the target begin date)

Evaluation	This function takes the target pool elevation value and converts it to a storage. This is an iterative procedure because there is not a one to one relationship between pool elevation and storage on a slope power reservoir. Once the target storage value has been computed, the change in storage over the target range is determined. Since the total inflow volume over the target range is given as an argument, the total outflow volume can be computed. Side flows are automatically included in the mass balance computation and thus should not be included in the inflow value provided in Argument 4. These include **Inflow 2**, **Hydrologic Inflow Net, Diversion, Return Flow, Canal Flow, Flow FROM Pumped Storage,** and **Flow TO Pumped Storage**. If any of the side flow slots contain NaN on one of the target period timesteps, the value will be assumed to be zero on that timestep. Other sources and sinks, such as **Evaporation**, **Precipitation**, **Bank Storage** and **Seepage** are NOT included in the mass balance computation. The total outflow volume is then converted to a flow rate and divided by the number of timesteps in the target range. The result is a single timestep outflow value. This value needs to be set on the outflow slot for every timestep in the target range in order to meet the target pool elevation.
Mathematical Expression	$\text{Outflow Volume} = \text{Previous Storage} - \text{Target Storage} + \text{Inflow Volume} + \text{Side Flows}$
Comments	

Syntax Example:

```
TargetSlopeHWGivenInflow(Lake Mead, @"24:00 January 1, 2002",
@"24:00 January 5, 2002",
{7500 "cfs", 7750 "cfs", 8125 "cfs", 8200 "cfs", 7900 "cfs"}, 1200 "ft", Lake
Mead.Pool Elevation[@"24:00 December 31, 2002",
Lake Mead.Storage[@"24:00 December 31, 2002")
```

Return Example:

```
23.43 "cms"
```

205. ToCelsius, ToFahrenheit, ToKelvin

Description	Convert a temperature to a given temperature scale.	
Type	NUMERIC	
Arguments		
1	NUMERIC	the temperature to convert
Evaluation	These functions take as input a numeric value representing a temperature in some scale and return the equivalent value in another scale.	
Comments	It is an error to try to convert a value that is not a temperature (i.e., is not in units of degrees Celsius, degrees Fahrenheit, or Kelvin.	

Syntax Example:

```
ToCelsius( 63.23 "F") = 17.35 "C"
ToFahrenheit( 290.5 "K") = 63.23 "F"
```

206. VolumeToFlow

This function evaluates to the average flow of water over a timestep, which corresponds to a given volume of water.

Description	The steady flow over a timestep corresponding to a volume.	
Type	NUMERIC	
Arguments		
1	NUMERIC	volume to be converted
2	DATETIME	timestep over which to convert
Evaluation	The number of seconds in the timestep of the datetime argument is determined. Then, the volume argument is divided by this number of seconds.	
Mathematical Expression	$flow = \frac{volume}{\Delta t_{(current\ timestep)}}$	
Comments	None	

Syntax Example:

```
VolumeToFlow($"Jemez Reservoir.Storage"[], @"t")
```

Return Example:

```
23203.231 "cms"
```

207. WaterOwners

This function evaluates to a list of all user-defined WaterOwners.

Description	This function returns a list of the names of all WaterOwners defined in the Water Accounting System Configuration.	
Type	LIST {STRING}	
Arguments		
Evaluation		
Comments	WaterOwners are properties of Accounts. The returned list does not include the default ("NONE") WaterOwner.	

Syntax Example:

```
WaterOwners()
```

Return Example:

```
{"IrrigationDistA","IrrigationDistB"}
```

208. WaterTypes

This function evaluates to the list of user-defined WaterTypes

Description	This function returns a list of the names of all WaterTypes defined in the Water Accounting System Configuration.	
Type	LIST {STRING}	
Arguments		
Evaluation		
Comments	WaterTypes are properties of Accounts. The returned list does not include the default ("NONE") WaterType.	

Syntax Example:

```
WaterTypes()
```

Return Example:

```
{"CityWater", "Farmer1", "Farmer2"}
```

209. WeightedSum

This function computes the normalized weighted sum of a list of numbers.

Description	The normalized weighted sum of a list of numbers.	
Type	NUMERIC	
Arguments		
1	LIST {NUMERIC}	the values to be summed
2	LIST {NUMERIC}	the weights of the values
Evaluation	The following mathematical expression is computed and returned.	
Mathematical Expression	$\dfrac{\sum_i \text{weight}_i \cdot \text{value}_i}{\sum_i \text{weight}_i}$	
Comments	All values must have the same dimensionality but may have different units (e.g., all values could be flows, but some in units of cms and others in cfs). Similarly, all weights must have the same dimensionality. Currently, if the dimensionality of the values or weights involves temperature, then all items in that list must have identical units (e.g., it would not be permitted for some values to have units of Celsius/meter and others to have units of Fahrenheit/ meter).	

Syntax Example:

```
WeightedSum({2.0 [m], 13.12 [ft]}, {0.5, 0.5})
```

Return Example:

```
= 2.99 [m]
```