



**Technical Documentation Version 6.2**

---

# **RPL User Interface**

---



**C A D S W E S**

**Center for Advanced Decision Support for Water and Environmental Systems**

These documents are copyrighted by the Regents of the University of Colorado. No part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means electronic, mechanical, recording or otherwise without the prior written consent of The University of Colorado. All rights are reserved by The University of Colorado.

The University of Colorado makes no warranty of any kind with respect to the completeness or accuracy of this document. The University of Colorado may make improvements and/or changes in the product(s) and/or programs described within this document at any time and without notice.

# RPL User Interface Table of Contents

<b>RPL Editor Dialogs .....</b>	<b>1</b>
Types of RPL Sets .....	1
Save Location .....	1
Managing RPL Sets .....	2
Actions specific to Rulesets, Optimization Goal Sets, and Global Function Sets ..	2
Actions specific to Accounting Method Set, Expression Slot Set, Initialization Rules, and Iterative MRM Sets.....	4
Tour of a RPL Set .....	5
Elements .....	5
RPL Set Editor View.....	7
Editing RPL Sets .....	9
Blocks and Groups .....	9
Validity .....	11
Working with RPL Dialogs .....	12
Execution Constraint / Execute Block Only When.....	13
Description .....	13
Comments .....	13
Executing DMIs from Blocks.....	14
Stop On NaN.....	14
Statements .....	14
Editing a RPL Expression .....	16
Using the palette .....	16
Entering Values .....	17
Undo and Redo .....	17
Using the History .....	18
Data types for looping variables.....	18
RPL Short Cuts .....	19
Disabling an Item in a List or a Statement.....	19
Open Slots and Objects from RPL dialogs.....	20
RPL Search and Replace Dialog .....	21
Accessing the dialog .....	21
Searching for occurrences of a string .....	21
Replacing matching strings with another string.....	22
Functions .....	22
Predefined Functions .....	22
Writing a Function .....	23
Constraints on Functions .....	25
Time Invariant Functions and Function Value Caching.....	25

---

Global RPL Functions .....	25
Creating a new Global RPL Function Set .....	26
Opening an Existing Global Function Set .....	28
Using Global RPL Functions .....	29
Example: Creating a new ruleset .....	29
<b>Developing Efficient RPL Expressions .....</b>	<b>32</b>
<b>RPL Printing and Formatting .....</b>	<b>33</b>
Printing .....	33
Formatting - The Display Settings .....	33
Font .....	34
Colors.....	34
Line Breaks.....	35
<b>Exporting and Importing RPL sets .....</b>	<b>39</b>
Export .....	40
Import .....	40
<b>RPL External Documentation .....</b>	<b>43</b>
Overview .....	43
File Association .....	44
Document Structure .....	45
Configuration and User Interface .....	47
Viewing and Editing .....	50
HTML .....	51
MS WORD .....	52
PDF .....	53
Text.....	53
Use Example .....	53

# RPL Editor Dialogs








## 1. RPL Editor Dialogs

This document describes basic rule, function, method, or constraint construction with the RPL Set Editor. The RPL Set Editor is the main window through which policy is managed. From this window:

- Rules, functions, or method, may be: added, deleted, opened, named, prioritized and turned off or on.
- Entire RPL sets may be: opened, closed, saved, loaded, exported, imported and unloaded.

### 1.1 Types of RPL Sets

This document refers to all of the RPL sets in RiverWare as follows. The table shows information about each set that will be referred to later

RPL Set	Set Color	Save Location	Link to More information
Expression Slot Set	 Orange	In model file	Click <a href="#">HERE (Slots.pdf, Section 3.5)</a>
Global Function Set	 Brown	Separate file	Click <a href="#">HERE (Section 1.11)</a>
Initialization Rules	 Teal	In model file	Click <a href="#">HERE (Simulation.pdf, Section 5.1.2)</a>
Iterative MRM Ruleset	 Navy Blue	In model file	Click <a href="#">HERE (MRM.pdf, Section 4.3.3)</a>
Object Level Accounting Method Set	 Green	In model file	Click <a href="#">HERE (Accounting.pdf, Section 4.2)</a>
Optimization Goal Set	 Purple	Separate file OR in model file	Click <a href="#">HERE (Optimization.pdf, Section 4.6)</a>
Rulebased Simulation (RBS) Ruleset	 Red	Separate file OR in model file	Click <a href="#">HERE (RulebasedSimulation.pdf, Section 1.3)</a>

When dealing with specific menus, this document will use the terminology “Set” in place of any type of set. Please note that each of the sets has a unique menu name and should be used accordingly.

### 1.2 Save Location

Based on the type, the set is saved in one of two locations as shown in the table above: in the model file or in an external file.

Four of the sets (expression slot set, initialization rules, iterative MRM rulesets, and accounting method set) are always saved in the model file. Global function sets are always saved in an external file. Rulesets and goal sets may be saved in either location as specified in the Run Parameter for that

controller. This is accessed from the **Run Control** dialog. When the Rulebased Simulation or Optimization controller is selected, then the **View → Optimization/Rulebased Simulation Run Parameters** menu opens the parameter dialog.

The toggle **Save Loaded RPL Set with Model** controls where the set is saved. By default, this toggle is **OFF** meaning the set is saved in a file external to the model. Checking it **ON** will save the **Loaded** set with the model file.

Save Loaded RPL Set with Model

---

**Note:** This toggle only applies to the **Loaded** set. Any other opened sets will not be saved with the model file.

---

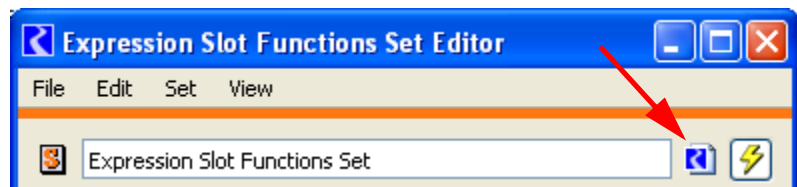
Once a set is saved with the model file, there is some risk that it may be lost if:

- the toggle above is unchecked and the model is saved,
- the set is unloaded and the model is saved,
- another set is loaded and the model is saved
- closing the set and the model is saved.

In each case, a warning message is presented to allow you to confirm that the action is intended. A set saved with the model is automatically loaded when the model is opened. The set is then minimized. You can bring it to the front at anytime using the workspace **Policy** menu or the buttons on the bottom of the workspace.



Further, for any set that is saved with the model, a model file icon is displayed at the top right of the dialog.



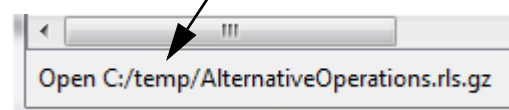
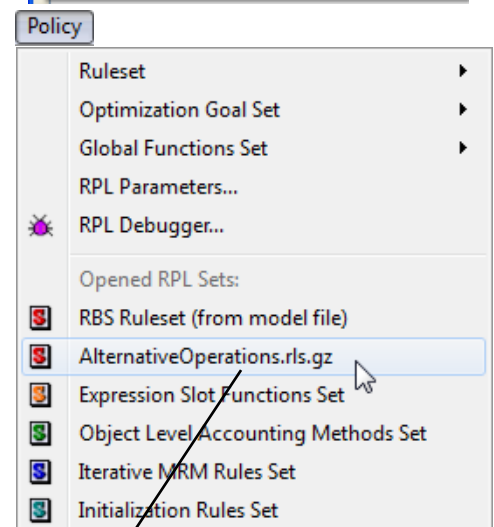
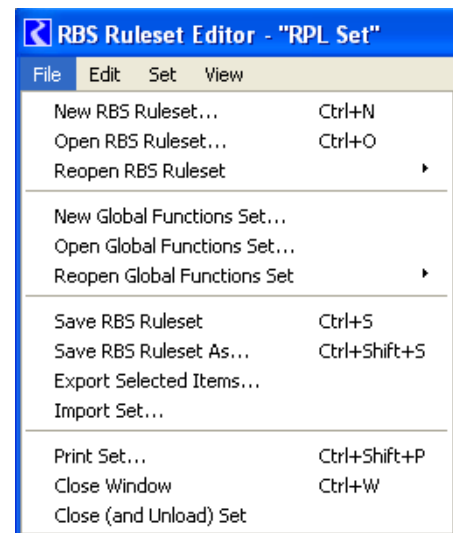
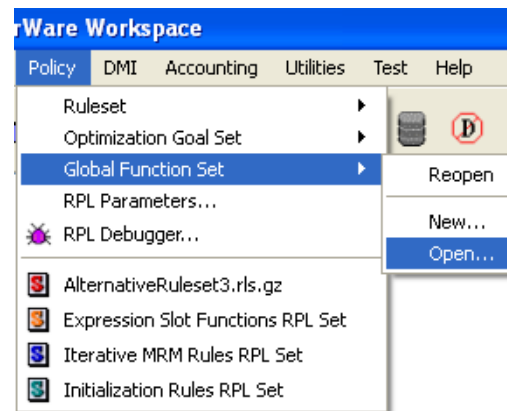
## 1.3 Managing RPL Sets



Following is a description of managing RPL sets. Much of this information is specific to the RPL set in question.

### 1.3.1 Actions specific to Rulesets, Optimization Goal Sets, and Global Function Sets

The following actions are specific to managing a set (either Ruleset, Optimization Goal Set, or Global Function Set) and are performed on an entire set:

- **New:** A new RPL set is created by selecting **Policy** ➤ **Set** ➤ **New...** from the main RiverWare workspace or **File** ➤ **New Set...** from an already opened set.
- **Open:** An existing RPL set (saved as a file) is opened by selecting **Policy** ➤ **Set** ➤ **Open...** from the main RiverWare workspace or **File** ➤ **Open Set...** from an already opened set. Either of these commands invokes a file chooser in which the desired set is selected. Each set is saved in a separate file. Several sets may be opened at once. Each opened set and any windows opened from these sets are identified by a unique color band across the top of their windows.
- **Reopen:** An existing RPL set (saved as a file) can be reopened by selecting **Policy** ➤ **Set** ➤ **Reopen...** from the main RiverWare workspace or **File** ➤ **Reopen Set...** from an already opened set. Either option then presents a cascading menu of recently opened sets and folders.
- **Reopen and Load:** An existing RPL set (saved as a file) can be reopened and loaded by selecting **Policy** ➤ **Set** ➤ **Reopen and Load...** from the main RiverWare workspace. Then, a cascading menu of recently opened sets and folders is presented.
- **Show RPL set:** When a set is opened, the name of the set is added to the workspace **Policy** menu as shown in the screenshot to the right. Only the short name is shown in the menu, but the full path is shown in the workspace status bar. Use this menu to raise/show the desired set. When the set is saved with the model file, the menu will show it as "RPL Set". Note that the expression Slot, initialization rules, MRM and accounting methods sets (saved with the model) are always shown too. Also, the workspace has buttons to show the **loaded** RBS ruleset



 or optimization goal set . The bar is colored when there is a loaded set, grey when there isn't. Click on one of the buttons (when a set is loaded) to raise that set to the top. The workspace also shows color coded buttons for all the opened sets. Tool tips indicate their names.



- **Save:** The RPL set is saved to a file by selecting **File** ➤ **Save Set** in the **Set Editor** menu bar. The name and directory of the file to which the RPL set is saved is that from which it was last loaded or to which it was last saved.
- **Save As:** A new RPL set must be given a name and saved for the first time by selecting **File** ➤ **Save Set As...** in the **Set Editor** menu bar. This command is also used to save an open RPL set to a different filename and/or directory from which it was last loaded or saved. The **Save Set As...** command invokes a file chooser in which the RPL set's new path and filename must be specified.
- **Load:** A ruleset or goal set is loaded into a model for use during a run by clicking on the **RPL Set Not Loaded** button on the right side of the **Set Editor** menu bar. When the set is loaded, the button text changes to **RPL Set Loaded** and the colored bar along the top of the **Set Editor** becomes red. Only one ruleset at a time may be loaded for use by a model. Loading a set when another is already loaded will unload the first ruleset. Global Function Sets are not loaded, any open set applies to all other sets.
- **Unload:** A loaded set is unloaded from a model by clicking the **RPL Set Loaded** button on the right side of the **Set Editor** menu bar. When the set is unloaded, the button text changes to **RPL Set Not Loaded**, and the red bar along the top of the **Set Editor** reverts to its original color (blue, green, yellow, purple, etc.).
- **Close Window (Ctrl+W):** An opened RPL set is closed but not removed by selecting **File** ➤ **Close Window** in the **Set Editor** menu bar. It can be re-shown using the Policy menu on the workspace.
- **Close (and Unload) Set:** An opened RPL set is closed by selecting **File** ➤ **Close (and Unload) Set** in the **Set Editor** menu bar. Closing a loaded set automatically unloads the set from the model. Closing does not automatically save changes to the set since the last save. This option is not available for those sets always saved with the model file.

### 1.3.2 Actions specific to Accounting Method Set, Expression Slot Set, Initialization Rules, and Iterative MRM Sets

The following actions are specific to the Object Level Accounting Method set, the Expression Slot set, Initialization Rules, and the Iterative MRM set. Each of these RPL sets is saved with the model so they do not need to be opened or saved separately.

To show the Object Level Accounting Method set (when accounting is enabled):

- From the Workspace choose the **Policy** ➤ **Accounting Methods RPL Set** or choose **Accounting** ➤ **Open Accounting Methods RPL Set**

To show the Expression Slot RPL set:

- From the Workspace choose the **Policy** ➤ **Expression Slot RPL Set**

To show the Initialization Rules RPL set:

- From the Workspace choose the **Policy** ➔ **Initialization Rules RPL Set**








To show the Iterative MRM RPL set:

- From the Workspace choose the **Policy** ➔ **MRM RPL Set**. The MRM RPL set can also be opened from the MRM run control dialog. This is further described [HERE \(MRM.pdf, Section 4.3.3\)](#).

Each of these sets can be closed using the **File** ➔ **Close Set** menu. Again, each of these sets is saved with the model file. So there is no menu to save the set. Also, these sets are always active in their respective context so it is not necessary to load or unload the sets.

## 1.4 Tour of a RPL Set

In RPL sets there is an upper level construct specific to the RPL set. For example, in a ruleset, a rule is the construct; in an object level accounting method set, a method is the upper level construct. In this document, a *block* refers to the upper level construct for each of these sets and will be used in all descriptions. Following is a list of the block for each type of set:

	RPL Set	Upper level <i>Block</i> :
	Expression Slot Set	N/A
	Global Function Set	N/A
	Initialization Rules Set	Rule
	Iterative MRM Ruleset	Rule
	Object Level Accounting Method Set	Method
	Optimization Goal Set	Goal
	RBS Ruleset	Rule

**Note:** Expression Slot and Global Function Sets do not have the paradigm of a block. For Expression Slot Sets, the analogous upper level construct is the expression slot itself. Global Function Sets do not have the concept of a block as they contain only global utility groups and functions.

This remainder of this section describes the components of a RPL set when using the RPL set editor









### 1.4.1 Elements


A RPL set includes the following general elements:

- **Policy Groups:** Policy groups are containers for blocks and functions. The functions within a policy group are available only to blocks and functions within that policy group.
- **Utility Groups:** Utility groups are containers for functions. The functions within a utility group are available to blocks and functions within any policy group.

- **Blocks:** Blocks (Rules/Methods) are prioritized expressions of policy which assign slots and/or generate diagnostic messages.
- **Functions:** Functions are subroutines called from blocks or other functions, which evaluate to a value in one of the expression data types.

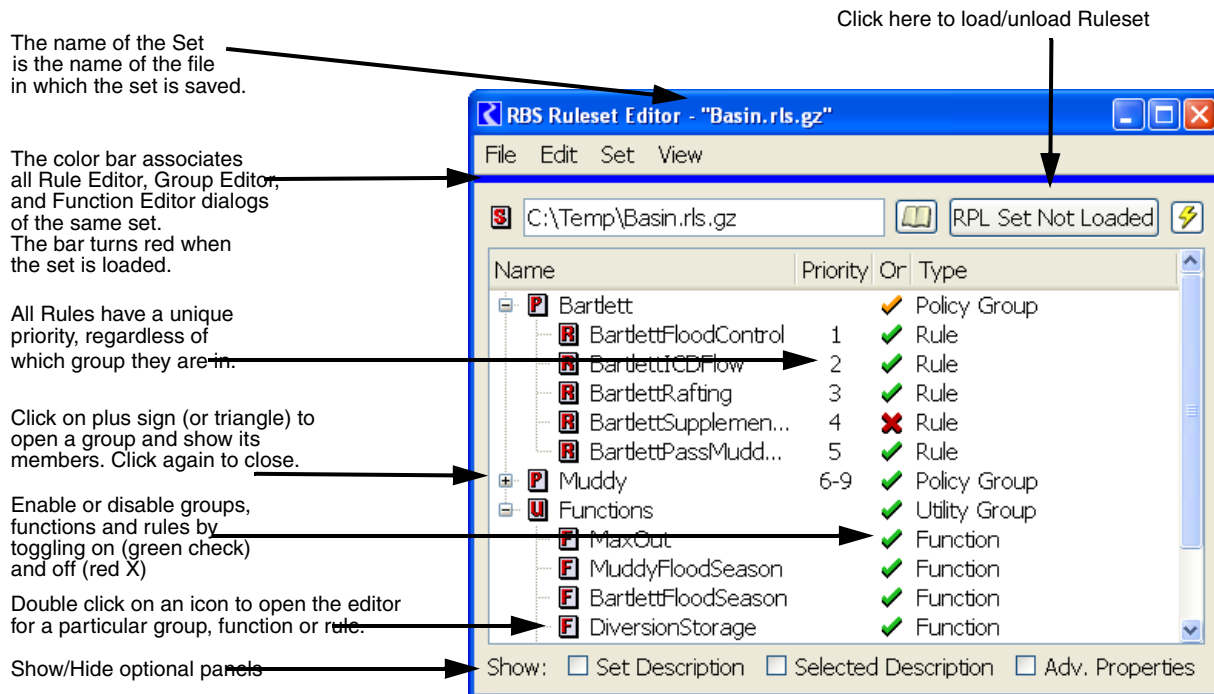
Following is a table showing all the RPL elements and their icons. Note, the icons are shown in grey but each set has its own color that is used for the letter:

Letter	Icon	Item	Letter	Icon	Item
G		Goal	S		Set
M		Method	=		Statement
P		Policy Group	F		User-defined Function
R		Rule	U		Utility Group

Predefined functions use the F icon but always have the light blue color: 

## 1.4.2 RPL Set Editor View

Information about a RPL set is displayed in the RPL Set Editor dialog. Each policy group, utility group, block, and function is represented in the RPL Set Editor on a different line.



Each line contains several pieces of information:

- **Name:** The name column contains an icon representing the type of the element and its name. All elements are assigned a default name when created.
- **Priority:** A priority is shown to the right of its name. Policy groups, utility groups, and functions do not have priorities but policy groups display the range of priorities of its member rules. Priorities number indicates the relative importance of the block. Priorities are shown but are not used in Object Level Accounting Method sets.
- **On/Off Status:** All elements may be turned on or off independently by clicking on the green check mark or red X in their line, unless mouse editing has been disabled. If mouse editing has been disabled, a block/function can be turned on or off by right clicking. This will bring up a menu to turn the element on or off. A green check mark indicates that an element is active. A red X indicates that an element has been turned off. An element that is turned off cannot be used during the model run. Turning off a policy group or utility group disables all of the blocks and/or functions within that group. Clicking on a group's red X now turns the entire group "on" without making any changes. This prevents the user from inadvertently changing the status of an item in a group that is off only to find that it is now different when the entire group is turned back

Bartlett		✓	Policy Group
BartlettFloodControl	1	✓	Rule
BartlettICDFlow	2	✗	Rule
BartlettRafting	3	✗	Rule
BartlettSupplemen...	4	✗	Rule
BartlettPassMudd...	5	✓	Rule

“on”. If all individual items in a group are turned off (red X), then the entire group is considered off and the group will also have a red X. If the user clicks on the group’s red X, a warning message will appear to indicate that at least one item in the group should be turned on. An orange check mark on a policy or utility groups specifies that one or more items in that group is “off” as shown in the following figure. As before, the user can click on the orange check to turn “off” the entire group. Clicking on it again, restores the previous state thus showing an orange check.

- **Type:** The type of each item in a set is shown in the **Type** column.

There are several additional pieces of information that can optionally be shown from a RPL set as accessed from the **View** menu:

**Expand/Collapse All:** A RPL set’s contents may be viewed with all groups expanded, some groups expanded, or all groups collapsed. To expand or collapse all groups simultaneously:

- Select **View** ➔ **Expand All Groups** or **Collapse All Groups** item in the RPL Set Editor menu bar.
- To expand or collapse a single group, click on the triangle next to the group name.



**Note:** The following three items can be shown using the View menu or clicking on the toggle in the **Show:** row. If there is an entry in these panels (or a non-default value) a box is drawn around the toggle. For example, in the above screenshot, the **Adv. Properties** has a non-default entry.

Show:  Set Description  Selected Description  Adv. Properties

**Show RPL Set Description:** A description of the RPL set may be typed into the RPL Set Description: text frame and saved in the RPL set file. The description may contain any combination of letters, numbers, spaces and punctuation, except double-quotes (“”). The area is opened and closed by toggling the **View** ➔ **Show RPLSet Description** in the menu bar or using the **Set Description** toggle at the bottom of the window.

**Show Selected Description:** A special expanded area of the RPL set is used to enter a description of any specific block. The area is opened and closed by toggling the **View** ➔ **Show Selected Description** in the menu bar of the **RPL Set Editor** dialog or using the **Selected Description** toggle at the bottom of the window. Single click on a block’s name to view the description of that specific block.

**Show Advanced Properties:** The **Advanced Properties** are accessed from the **View** ➔ **Show Advanced Properties** menu or using the **Advanced Properties** toggle at the bottom of the window. The **Advanced Properties** areas has two components to:

- **Agenda Order:** Rulesets can be configured to execute in either ascending or descending priority order. Descending priority order executes the highest priority block first, then each of the lower priority blocks (1,2,3,...). Ascending priority order executes the lowest priority block first, then each of the

higher priority blocks (...3,2,1). Ascending Priority (...3,2,1) is the default selection and the recommended priority order. The agenda order is only applicable and shown for RBS rulesets, initialization rules, iterative MRM rulesets and optimization goal sets. Also, optimization goal sets always must execute in the 1,2,3... order and cannot be changed.

- **Precision:** The precision spinner determines the number of digits after the decimal which are displayed for numbers in blocks and functions. 8 is the default value. The displayed precision has no effect on the twelve significant figures of internal precision used for calculations.

**Show Predefined Groups:** In the **View ➤ Show Predefined Groups** menu, the user can choose to show the groups for the predefined functions.

**Show Statements:** In the **View ➤ Show Statements** menu, the user can choose to show statements (assign, print, etc...) as another level in the RPL dialog's tree-view. The user can also change the name of statements once shown. To do this, the user right-clicks on that item to show the context sensitive menu. Then, selecting **Rename** will open an inline name editor and the user can enter a new name. In addition to the RPL Set view, this statement name will be shown in the RPL debugger and RPL Analysis Tool.

**Show Export Columns:** The **View ➤ Show Export Column** menu allows the user to specify if the selected row should be exported. Import/Export of RPL sets is described [HERE \(Section 4\)](#).

**Disable Mouse Edits:** It is easy to rename a group, block or function, change the priority of a block, and turn a block on or off. To prevent accidental changing of a block's priority or turning the block on or off in the Set Editor as a result of mouse clicks, select **View ➤ Disable Mouse Edits**. This will prevent editing with mouse clicks in the RPL Set Editor. To edit a block or group after mouse editing is disabled, double click or right mouse click on the block, function, or group's name to bring up an editor dialog. We highly recommend disabling mouse editing to prevent accidental changes to a RPL set.

## 1.5 Editing RPL Sets

Following is a description of how to manage and edit RPL sets once they are open. This includes creating, naming, and editing blocks/groups, adding statements, and validating blocks or the entire set. Following this is a description of how to edit individual expressions, [HERE \(Section 1.8\)](#).

### 1.5.1 Blocks and Groups

**Naming:** To name a block or group, right click or double click on the name field of the block or group. Right clicking will bring up a menu from which the editor dialog can be opened. Double clicking will bring up an editor dialog directly. Type the name in the name field.



**Add Policy and Utility Groups:** Policy and utility groups are added to a RPL set by selecting **Set ➤ Add Policy Group**, **Set ➤ Add Utility Group**, or **Set ➤ Add Global Utility Group** (for global function

sets) from the RPL set's menu bar. New policy and utility groups are added to a set directly below the currently highlighted group of that type.

**Add Blocks and Functions:** Blocks and functions are added to a RPL set by highlighting the group they are to be placed in and selecting **Set ➤ Add Block**, **Set ➤ Add User-Defined Function**, or **Set ➤ Add Global User-Defined Function** (for global function sets) from the RPL set's menu bar. New blocks and functions are added to a RPL set directly below the currently highlighted element of that type or at the bottom of a highlighted policy or utility group.

**Change Priorities.** Blocks and functions may be rearranged in a set. By changing the relative priority of blocks, the order of firing, success of slot assignments, and order of dispatching is altered. Changing of priorities is often used to evaluate the effectiveness of different operational policies. Rearranging blocks and functions in a set is simple. To move a block in the list, click on its icon and drag the block to the new desired location within the set. If you place it on another block, it will append after that block. Then, simply release the mouse button to drop the block in its new location. A dialog will ask you to confirm that you wish to complete the move. Click OK to confirm the move. All other blocks will be shifted to comply with the new priority structure. There is no need to re-load a loaded set when rearranging blocks. Moving a block or a function is done in an identical manner. Predefined functions cannot be moved.

**Delete Groups, Blocks and Functions:** Groups, blocks and functions are deleted from a RPL set by highlighting them and selecting **Edit ➤ Delete**. Deleting a policy or utility group deletes the contents of the group as well.

**Open and close Policy and Utility Groups.** Policy and utility groups which contain at least one block or function are opened in a separate window by double clicking their  or  icon or highlighting them and selecting **Set ➤ Open Editor...** They may also be expanded within the **RPL Set Editor** window by clicking on the white tree-view triangle to the left of their icon or selecting **View ➤ Expand**. An expanded policy or utility group can be collapsed by clicking on the white tree-view triangle again or selecting **View ➤ Collapse**.

**Open and close Rules, Functions, Methods, and Goals.** Blocks and functions are opened in a separate window by double-clicking their icon or highlighting them and selecting **Set ➤ Open Editor**. An opened block or function may be closed by selecting **File ➤ Close Block** or **File ➤ Close Function**. Copy and Paste Groups, Blocks, and Functions

Groups, blocks and functions may be copied and pasted within a set or between open sets. An item is copied into memory by highlighting it and selecting **Edit ➤ Copy**. An item in memory is pasted just below any highlighted item in a RPL set by selecting **Edit ➤ Append**. An item may also be copied and pasted just below the item by selecting **Edit ➤ Duplicate**. Copy/Paste/Duplicate is only available within a single model; you cannot copy/paste items from RPL sets that are opened in separate instances of RiverWare. Click [HERE \(Section 4\)](#) for more information on Import/Export to perform this task.

---

**Note:** A copied group/block/function can be pasted (or dragged and dropped) as a text representation into any text editor such as a word processor (Word), text editor (notepad), or email program (thunderbird). This includes all information about that group, block, function including the logic, description, active state, etc...

---

**Exporting and Importing Groups and Block:** In addition to copy and paste operations, it is also possible to export and import all or part of a RPL set. This is useful to share user-defined functions between two RPL sets. Using the export and import utility, the user can share groups, blocks, and functions amongst these sets. The export and import dialogs are described [HERE \(Section 4\)](#).

### 1.5.2 Validity

Before loading a newly built set into a model (or running with one of the other types of sets), the validity of the set must be checked. This can be done manually by the user or is automatically done at run start. Manually validating a RPL set, however, allows the user to perform these checks without having to bring up the Run Control dialog, start the run, and stop the run or wait for it to finish. Validating a RPL set checks for unspecified expressions, illegal object and slot names, conflicting expression types, and syntax errors. It *does not* check the consistency of unit types in mathematical expressions. Unit consistency is done while the block is evaluated during the run.

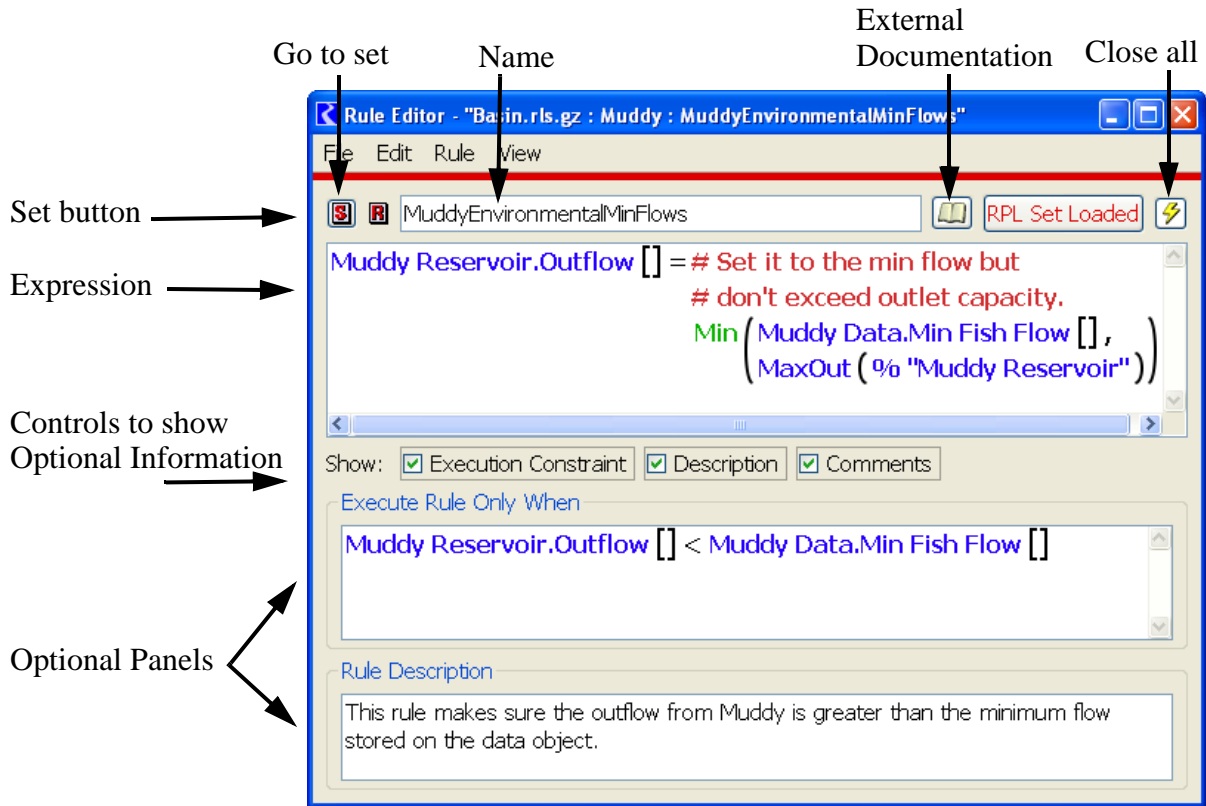
- To check the entire set, in the RPL Set Editor, select **Set** ➔ **Check Validity**.
- To check a single block or user defined function, select **Block** ➔ **Check Validity**.

If the block or set is valid, a confirmation window appears. Click **OK** and continue. Otherwise, a validation error dialog appears and the diagnostic window posts the location of the invalid expression. The user then must fix the block.



## 1.6 Working with RPL Dialogs

The RPL dialogs consist of menus, name and tool buttons, the expression, and optional additional panels as shown in the following screenshot. These items are described in this section.



- **Type of Set:** The icon indicates the type of set to which this editor belongs.
- **Name:** The name field is where the user specifies the name.
- **External Documentation:** The button, when configured, opens the external documentation as described [HERE \(Section 5\)](#).
- **Close All:** The button invokes a dialog in which the user can close all open RPL dialogs except the top level editor.
- **Set Button:** Icon buttons allow the user to show the containing set for each RPL dialog.
- **Expression:** This area contains the expression.
- **Optional Information:** Optional information can be shown for various types of RPL dialogs. These are described in the following sections. If there is an entry in these panels (or a non-default value) a box is drawn around the toggle.



### 1.6.1 Execution Constraint / Execute Block Only When

A block can be set up to execute only when a given condition is true. This functionality can be used to increase performance or it may be necessary to implement your RPL set correctly.

- From a RPL editor, select **View** ➔ **Show Execution Constraints** or click the **Execution Constraint** toggle.

An area is added to the bottom of the window. By default, the block executes only when **TRUE**. This means that the block will always execute when it comes up on the agenda. The user may want this block to execute only when some other condition is true. The **TRUE** expression can be replaced with any logical structure that can be constructed from the palette

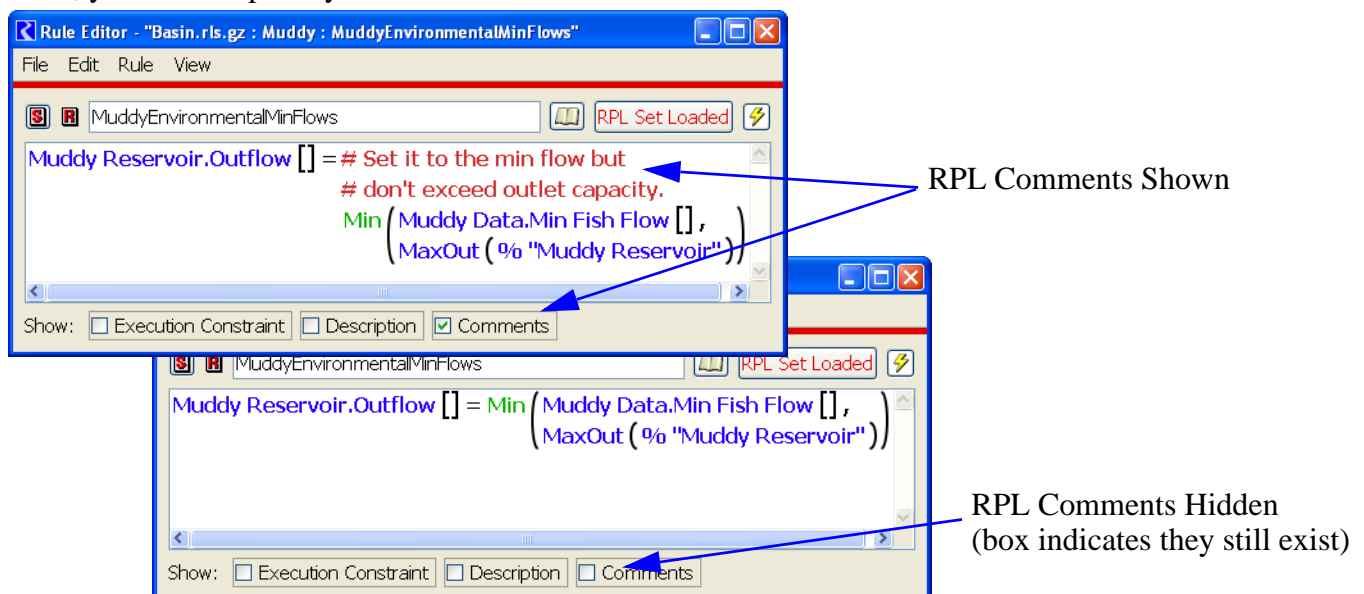
**Note:** If you enter a boolean expression to replace the default TRUE, then clear this expression, an unspecified expression will remain. This unspecified expression will invalidate the rule. To prevent this, if you clear out this boolean, make sure you re-type TRUE to get the default behavior.

### 1.6.2 Description

A special expanded area of the RPL dialog is used to enter a description of the block. The area is opened and closed by toggling the **View** ➔ **Show Description** in the menu bar or click the **Description** toggle at the bottom of the window.

### 1.6.3 Comments

Inline RPL comments (added from the palette) can be optionally shown/hidden using the **View** ➔ **Show Comments** menu or the **Comments** toggle. Note, if there are any comments defined, the **Comments** toggle will have a rectangle around it. Also, **Comments** are shown by default, if you do not want to see them, you must explicitly hide them.



### 1.6.4 Executing DMIs from Blocks

RPL blocks can be used to execute DMIs or DMI groups. DMIs are described [HERE \(DMI.pdf, Section 2\)](#). Select **Rule/Method ➔ Add Pre-execution DMI Group** or **Rule/Method ➔ Add Post-execution DMI Group**. Then, type in the name of a DMI or DMI group that is already defined in the model. A pre-execution DMI is executed as the first step of the execution, a post-execution DMI is executed as the last step of execution. Values imported by the pre-execution DMI are available for use by the statements.

To prevent unexpected solving of objects and/or conflicts between values, input DMIs executed from RPL blocks can only import values to **unlinked series slots on data objects**. Furthermore, values imported from a DMI executed from a RPL block are given the following flags:

- Control File-Executable DMIs: imported values are given the INPUT (I) flag (priority zero).
- Database DMIs: imported values are given the OUTPUT (O) flag and the priority of the rule executing the DMI.

An example application of this feature is executing an external water quality model at each timestep. In this example, the water quality model might take as input RiverWare values which reflect the current state of the system and return a recommended reservoir release value. At each timestep the block would first execute an output DMI to export current values from relevant RiverWare slot(s), e.g., inflow on a reach. An input DMI then runs the external water quality model (which reads the data from the output DMI) and then imports the resulting recommended reservoir release values.

### 1.6.5 Stop On NaN

Typically, when a RPL block accesses a slot but the slot is NaN, the block terminates early but the run continues. Under certain conditions, you may wish to specify that a block does not terminate early but instead stops the run when a NaN is encountered. This could be useful for data checking rules; if the referenced slot value has not been specified, then you wish to stop the run and fix the data error.

To specify that a block should “Stop On NaN”, use the **Block ➔ Stop On NaN** menu. When enabled, a check mark is added next to the menu. Any time a slot expression accesses a NaN, the run will stop and a diagnostic will be posted that explains which block referenced a NaN.

## 1.7 Statements

As described [HERE \(Section 1.1\)](#), blocks are the upper level construct of a RPL set; for example, rules and accounting methods are blocks. Blocks are made up of statements which are different structures depending on the desired result.

- In the RPL set editor, open a block by double clicking or right clicking on its icon. This will bring up the **<block> Editor** dialog. Rename the block by typing in a new name in the Name text field.

Blocks are constructed from basic statements and expressions. The following basic statements are available in the **Set** menu in RBS rulesets, initialization rules, MRM rulesets, and accounting method sets. They are:

**1. Assignment:** An assignment statement assigns to the slot on the left-hand side (LHS) of the equality the numeric value evaluated on the right-hand side (RHS) of the equality. The basic assignment is:

```
<numeric expr> = <numeric expr>
```

where the LHS <numeric expr> is a slot and the RHS <numeric expr> is any expression which evaluates to a value in the unit type of the LHS slot.

**2. Print:** A print statement evaluates its expression and formats the result into a message. The message is displayed in the **Diagnostics Output** window when the **Print Statements** diagnostics group is enabled. The basic print statement is:

```
Print <expr>
```

where the <expr> is any expression or concatenated expressions which can be fully evaluated and represented as a string. For more information on the Print statement click [HERE \(Diagnostics.pdf, Section 3.3\)](#)

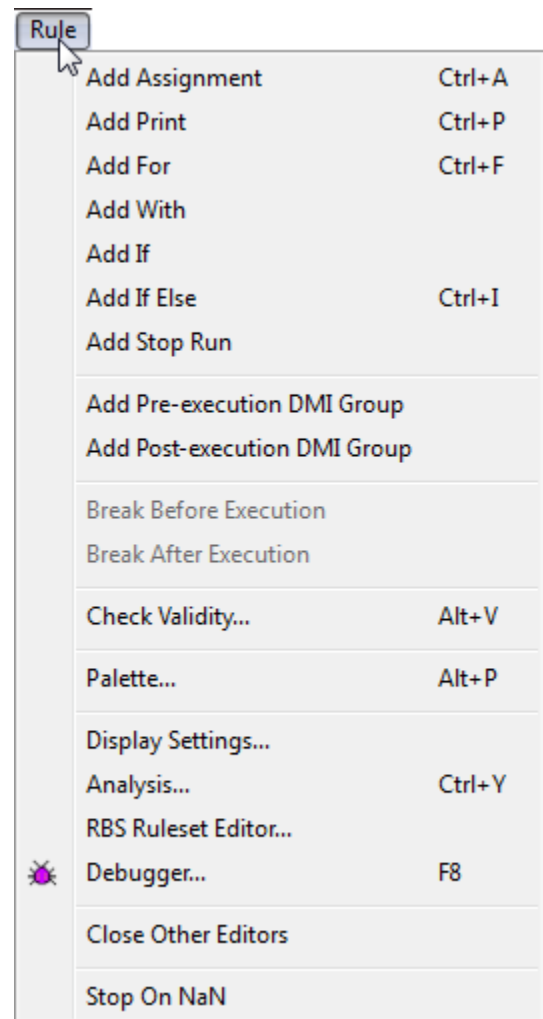
**3. For:** Iterative loops can be very useful for computations and multiple assignments. ForEach loops are available to make multiple slot assignments from similar logic and calculations. An index variable is assigned a new value for each iteration of the loop. The inside of the loop is one or more regular assignment statements which should use the index variable to perform a different assignments for each iteration of the loop. This variable may be used in both the LHS slot assignment and the RHS evaluation to affect slightly different behavior within each pass. The default foreach loop is:

```
FOR (NUMERIC index IN <list expr>) DO
  <numeric expr> = <numeric expr>
END FOR
```

where the number of loops/assignments is determined by the number of elements in the <list expr>, the **NUMERIC** label indicates the expression data type of the elements in the <list expr>, and the **index** is the variable name which will take on the value of each element for use inside the loop. All of these parts of the foreach statement may be modified.

Note, there is also a For expression on the palette as described [HERE \(RPLTypesPalette.pdf, Section 2.6\)](#).

**4. With:** Evaluate the expression and set the result to a local variable with the given name and type. then evaluates the contained statements, which may reference the variable. The default foreach loop is



```

WITH (NUMERIC val = <numeric expr>) DO
  <numeric expr> = <numeric expr>
END WITH

```

Note, there is also a WITH expression on the palette as described [HERE \(RPLTypesPalette.pdf, Section 2.6\)](#).

5. **If:** Make a statement conditional on a boolean expression without an ELSE.

```

IF(<boolean expr>) THEN
  <statement>
END IF

```

Note, there is also an IF expression on the palette as described [HERE \(RPLTypesPalette.pdf, Section 2.6\)](#).

6. **If Else:** Make a statement conditional on a boolean expression with an ELSE.

```

IF(<boolean expr>) THEN
  <statement>
ELSE
  <statement>
END IF

```

Note, there is also a IF ELSE expression on the palette as described [HERE \(RPLTypesPalette.pdf, Section 2.6\)](#).

7. **Stop Run:** Abort the run and post the provided expression as a diagnostic error message. When executed from within an iterative MRM rule, this statement aborts the MRM run.

```

STOP RUN <expr>

```

## 1.8 Editing a RPL Expression

Expressions in the RPL editor are displayed as <expr>, <numeric expr>, <boolean expr>, <list expr>, <string expr>, <object expr>, <slot expr> or <datetime expr>. The type of the expression indicates the valid values that can go in that expression. Click [HERE \(RPLTypesPalette.pdf, Section 1\)](#) to see a description of each data type.

There are two ways to enter data into an unspecified expression, typing and using the palette. Any expression can be typed by first double-clicking then typing in the expression. There are many problems with this approach. First the user must know the exact syntax to be typed. Second, the user must type all strings perfectly as typos in strings may not be caught on validation. A better approach is to use the palette to build expressions. The user only needs to type when they are specifying the inner most expression like exact numbers (100 “cfs”), boolean (true, false), datetimes (@“t”), or strings (“Entering Runoff season”). Following is a description of using the palette.

### 1.8.1 Using the palette

First, the palette buttons are described [HERE \(RPLTypesPalette.pdf, Section 2\)](#). Each of the buttons in the RPL Palette represents an expression. These expressions are convenient operations which evaluate to

one of the expression data types just mentioned. Buttons on the RPL Palette are enabled and disabled dynamically. When an expression is highlighted in the Editor, the Palette buttons that satisfy the expected data type are enabled. Thus the user creates expressions in the following manner: click on an unspecified expression, click on a palette button or function. Click on another unspecified expression and click on a palette button or function. Repeat this process until the expression is created. Then, if necessary, double click on any remaining un-specified expressions that cannot be created from the palette (like entering numbers) and type in the value.

### 1.8.2 Entering Values

To replace an unspecified expression with a literal value (like 10 “cfs” or TRUE) the user double clicks on the expression and types in the value. The user must include any symbols (like @, \$,%) to represent the type. All the data types are described [HERE \(RPLTypesPalette.pdf, Section 1\)](#). For NUMERIC data types, the user may also enter a unit. If the unit has a “-” or “/” in it, the user must include the quotes when typing. Without these special characters, the user can type the square brackets and no quotes. Here are some examples of correctly specified units: 10 [cms], 100 [“acre-ft”], 50. Illegal entries include 100 [acre-ft/day} (no quotes), 100 [“cfx”] (no unit of that type). Additional information on entering units can be found [HERE \(RPLTypesPalette.pdf, Section 1.1\)](#).

Another illegal entry is: 1,000 “acre-ft”. This is because the comma (a thousands separator) is not allowed as an input. Instead, RiverWare gives the option of whether or not to display a comma as a thousands separator for all values throughout the model and RPL set. This option is selected or deselected by clicking on the main **Workspace** ➔ **Show Commas in Numbers** option, and is described in more detail [HERE \(Workspace.pdf, Section 5.7\)](#).

Note, after typing a literal value, the user no longer needs to hit enter to commit the change. Enter or any mouse click outside of the editor will signal completion of the inline editing.

### 1.8.3 Undo and Redo

When editing RPL expressions, undo and redo are available from the:

- **Edit** menu on RPL dialogs
- Right-click context menu
- Keyboard shortcuts
  - Undo is ctrl-z
  - Redo is ctrl-shift-z
- **Expression** menu on Expression slots.

Any action that edits a RPL expressions can be undone/redone including:

- Replacing the selected expression with a new expression by clicking on a palette button
- Cutting or deleting the selected expression.
- Pasting an expression in the place of the selected expression.
- Adding or removing a statement from a block.

- Adding or removing a constraint from a function.
- Double-clicking on an expression and changing it via an inline editor.

Undo and redo are on a per-dialog basis; the dialog must be selected before the undo/redo operation is performed. Also, the history of changes is preserved if a RPL dialog is closed and re-opened, but is not preserved if the set is closed. On expression slots, the history of changes is lost when the slot is closed.

The number of undos/redos is only limited to take the user back to the original expression; there is no artificial limit imposed.

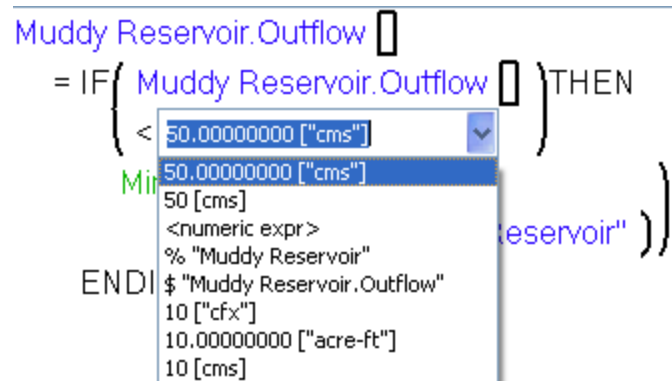
Finally, sections on the RPL set level cannot be undone including adding or deleting rules and re-arranging rule priorities.

### 1.8.4 Using the History

When the user double clicks an unspecified expression or literal value, a pull-down menu appears to the right of the inline editor. This pull-down menu tracks the history of past edits that have been used in this dialog. The user can then use the pull-down menu to choose a value from the list. The arrow keys can also be used to scroll through the list.

The history can also be used as a starting point if the user types in an invalid expression. It provides an opportunity to correct the entry and try again without having to type it from scratch.

For example, the user double clicks on an expression and types the string "Flood Season" but forgets the closing quote. RiverWare will issue an error that the expression cannot be parsed and will revert to the original expression. The user can then select "Flood Season" from the pull-down history, add the closing quote, and hit return.

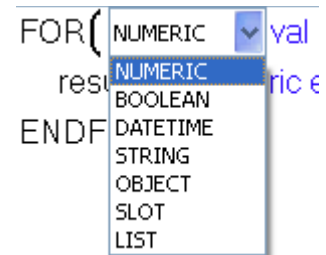


**Note:** The saved history is a per-dialog history only but does persist when the dialog is closed and re-opened. It does not persist if the RPL set is closed.

### 1.8.5 Data types for looping variables

In the looping functions, FOR, WHILE, WITH, SUM, AVE and in the FOR statement, the user specifies the Data Type (More on Data Types can be found [HERE \(RPLTypesPalette.pdf, Section 1\)](#) of the looping variable. For example, the first line of a default FOR statement looks like FOR (NUMERIC val IN <list>) WITH NUMERIC result = <numeric> do

The user can change the type of the looping variable NUMERIC and the name of the variable. Pull-down menus provide an easy way to do this. The user double clicks on the NUMERIC and is provided with a pull-down menu listing the 7 data types in RPL as shown to the right. A second pull-down menu is provided for the looping variable; this provides a history of all string values that have been previously entered in the dialog, similar to the history described in the previous section.



### 1.8.6 RPL Short Cuts

If part of an expression can be used in another assignment, rule, or function, rather than repeating the above process for each new expression, highlight the portion of the expression that can be used elsewhere. From the menu bar, select **Edit** ➔ **Copy** or type **<Ctrl C>** to copy the highlighted portion. Right clicking on the highlighted portion will bring up a menu with the option to copy. In the location in which the expression is to be pasted, highlight the **<expr>** and select **Edit** ➔ **Paste** **<Ctrl V>**, or right click and select **Paste** from the menu. This will paste the expression into the new location.

Also, expression can be deleted using the **Cut** **<Ctrl X>** or **Delete** **<delete>** operations. The **Cut** operation, puts the expression in the copy buffer, the **Delete** operation does not.



**Note:** There is one exception to the cut/delete behavior. When deleting an item in a list, the first cut or delete operation removes the expression but leaves a blank expression. A second cut/delete removes the item from the list. For example:

Statements may be cut, copied, pasted from the menu bar of the RPL Editor, using **<Ctrl C>**, or right clicking and selecting **Copy** from the menu. Select **Paste** to paste over an existing statement, **Insert** to add it above the currently selected statement, or **Append** to add below the last statement.

```
{1.00000000, "Flood", @"t"}
```

↓  
1st cut/delete  
clears item

```
{1.00000000, <expr>, @"t"}
```

↓  
2nd cut/delete  
removes item

```
{1.00000000, @"t"}
```

### 1.8.7 Disabling an Item in a List or a Statement

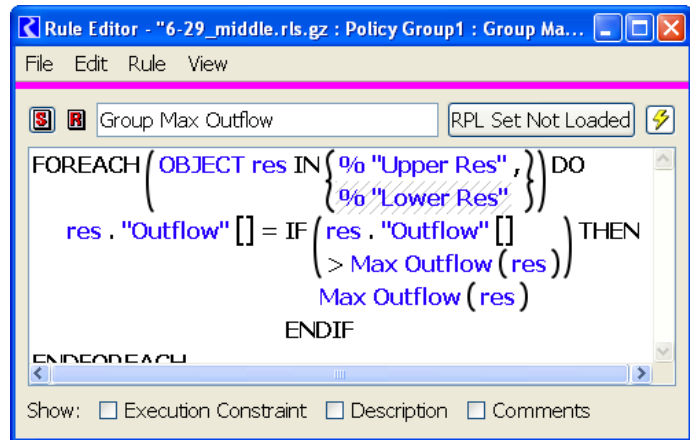
On the RPL set editor, functions and RPL blocks can be disabled by clicking on the green check mark turning it into a red X. Within a block, it is also possible to disable an individual item in a list or an entire statement.

- In the RPL editor, highlight the statement or item in the list that you wish to disable.

- Right click on the selected expression. Notice that there is a check mark next to the word “Enabled” indicating that the item is enabled.
- Click “Enabled” to toggle off the check mark thereby disabling the item. The disabled variable now shows up with cross hatching. The cross hatch color is defined in the set layout dialog [HERE \(Section 3.2.2\)](#). When the expression executes, it will skip the disabled item(s).

In general, disabling an assignment or an item in a list should only be used in the debugging and model building process. For example, you might want to have two assignment statements in one block. While debugging the model, it

may be necessary to disable one assignment to ensure the other assignment is working correctly.

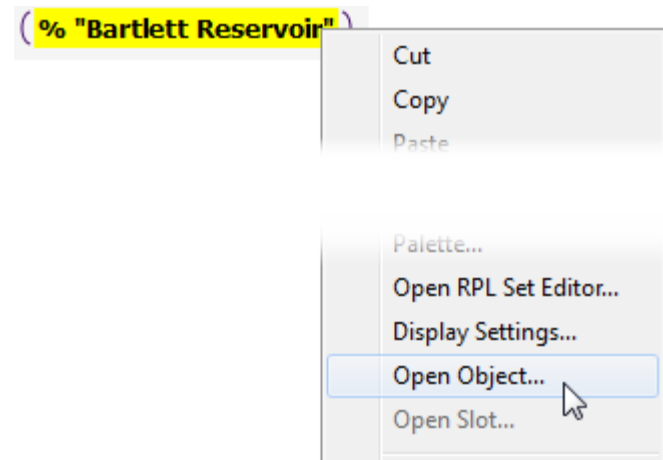


### 1.8.8 Open Slots and Objects from RPL dialogs

When the selected RPL expressions is a valid OBJECT or SLOT, the right click context menu

- **Open Object...**
- **Open Slot...**

will open that object or slot dialog. Note that this only works for expressions for which the workspace object can be determined without evaluation.



## 1.9 RPL Search and Replace Dialog

The RPL search and replace dialog supports flexible replacement of strings within a RPL set. Within this dialog the user can search for all occurrences of a string and replace all or some of them with another string.

### 1.9.1 Accessing the dialog

The RPL search and replace dialog is accessible in many ways:

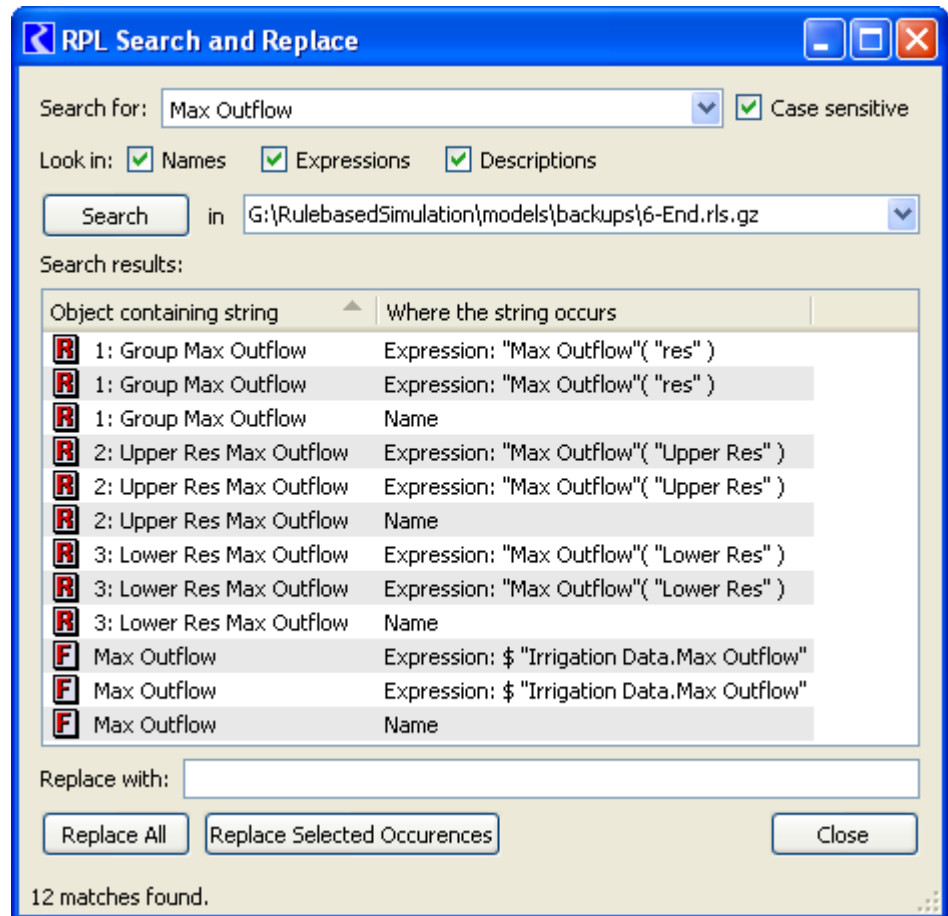
- Selecting “Search and Replace...” from the Edit menu of the RPL editor and its children dialogs (the RPL Set, Function, and Rule/Method editors) or from the RPL Set analysis dialogue
- Pressing the “F4” key from any of the RPL editors or the analysis dialogue
- Clicking on the right mouse button when editing any RPL expression and selecting “Search and Replace...” from the resulting popup menu

When the dialog first comes up it will be associated with the RPL set corresponding to the dialogue through which it was accessed. The name of this set will be displayed in a box to the right of the “Search” button. You may switch from this set to any other open set by clicking on the name of the current set. This will give you a menu containing the name of all open RPL sets and you may select any of these.

### 1.9.2 Searching for occurrences of a string

To search for occurrences of a string, begin by typing the search string into the window labelled “Search for:”, then either hit “Return” or click on the “Search” button.

When you initiate a search by hitting “Return”, the search string is added to a “history” of searches. To re-select a previous search string as the current search string, click on the triangle just to the right of the



search string. This will present a menu containing previous search strings, allowing you to select one as the current search string.

Exactly where and how a search is conducted can be controlled by selecting various check boxes:

- Case sensitive -- when checked, RiverWare searches for strings which match the search string exactly, otherwise it will ignore case.
- Look in Names: when checked, RiverWare tries to match occurrences of the string within the names of RPL objects in the set. This includes the names of the RPL set and its contained groups, rules/methods, and functions.
- Look in Expressions: when checked, RiverWare tries to match occurrences of the string within all expressions in the RPL set. Eligible sub-expressions within an expression include values of type STRING, OBJECT, and SLOT, as well as function calls (the name of the function being called is eligible for a match)
- Look in Descriptions: when checked, the contents of all (set, group, and rule/method) descriptions are searched for matching strings.

The results of the search are displayed in the Search results table which contains one row for each string that matches the search string. Each row displays the name of the object which contains the matching string (a set, function, group, rule/method, or slot with expression) as well as where within that object the matching string was found (the object's name, description, or one of its expressions). In the case of a match within an expression, the actually matching sub expression is also displayed.

Note that a search string can match multiple times within a given object. For example, when searching for the string "res", a rule named "Forrest Reservoir Flood" will match two times.

Double-clicking within a row of the Search results table will bring up an editor for the object containing the matching string corresponding to that row. If the match is within an expression, the matching sub-expression will be the selected expression in the editor.

### 1.9.3 Replacing matching strings with another string

Once a search has been conducted, one can replace all or some of the matching strings with another string. First, type the replacement string into the field labelled "Replace with:". To replace all of the occurrences then click on the "Replace All" button. To replace only some of the matching strings, select those that you wish to replace in the Search results list (left mouse selects a single row, Ctrl-left mouse allows multiple independent selections, Shift-left mouse selects a range of contiguous rows), then click on the "Replace Selected Occurrences" button.

## 1.10 Functions

Following is a description of functions and their use in RPL sets.

### 1.10.1 Predefined Functions

Predefined functions are also useful in constructing rules, methods, and functions. Predefined functions are a set of mathematical, look-up, and mass balance routines that may be accessed from within any

other expression. Predefined functions are coded into the RiverWare source code. Because of this, predefined functions cannot be modified. All predefined functions return their results in one of the standard data expression types; they can be substituted for any unspecified expression of that type. Predefined functions are accessed in the RPL Palette, [HERE \(RPLTypesPalette.pdf, Section 2\)](#), under the Predefined Functions tab at the top of the Palette dialog. Predefined functions are described in greater detail in the RPL Predefined Functions online help documentation. Click [HERE \(RPLPredefinedFunctions.pdf, Section 1\)](#) to go to that file.

### 1.10.2 Writing a Function

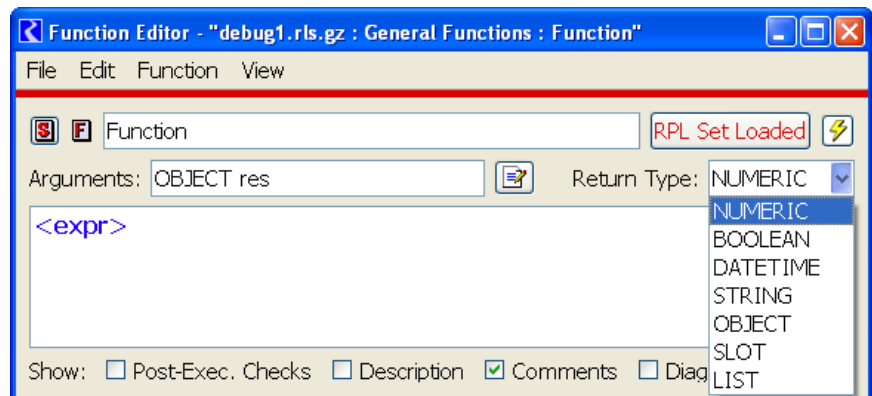
Functions are constructed in much the same way that blocks are constructed, using the RPL Palette. Expressions are simplified by using functions to perform logical and computational operations. To make a function flexible so that it may be used in a variety of situations, arguments are added to the function. Arguments permit the same function to behave differently, depending on from where it is called or by which object. For example, an internal function could be created which forecasts the evaporation from a reservoir based on the reservoir's surface area. This function could be of use at several reservoirs, but would have to know at which reservoir's data to look. The reservoir for which the function should compute the evaporation could be an argument to the function.






Arguments to functions can be of any data expression type. There is no limit to the number of arguments a function may take, but the number of arguments is not dynamic with respect to block execution. The exact number and type of argument(s) is fixed in the function definition. Dynamic argument lists and default argument values are prohibited.

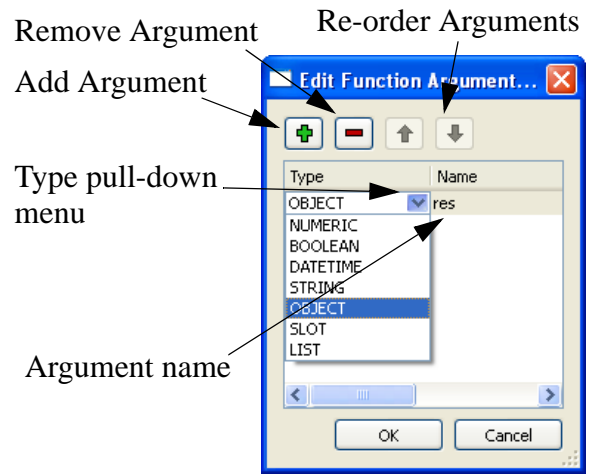
- Add a function to the desired Policy or Utility Group by selecting **Set ➔ Add Function**.
- Name the function and type <Return> after naming.
- In the field **Return Type**, select the data expression type you wish the function to return. For example, if NUMERIC is selected the function will return a numeric value to the block in which the function is called.

If the function is going to be general to other functions or blocks, adding an argument might be useful depending on the return type of the function. In the above graphic, the **Arguments:**

**OBJECT res** allows the variable “res” to be used inside of the function anywhere an **<object expr>** is allowed. This will allow the function to look at different reservoirs' data without creating a separate function for each reservoir. If the function is going to be used only in a limited manner, arguments are not necessary.



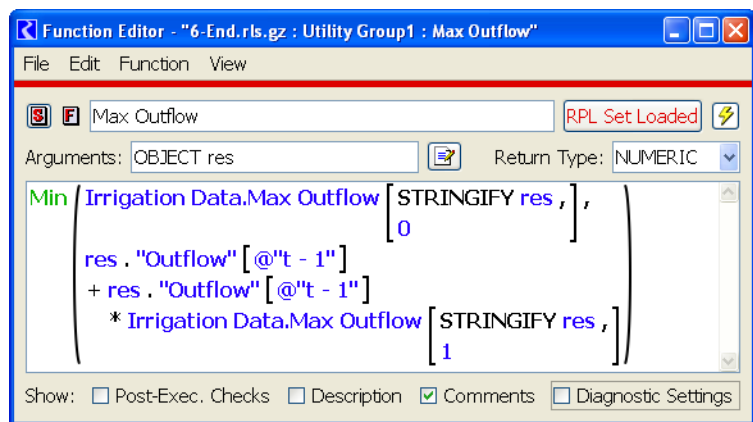
Arguments to functions can be entered in two ways, using the editor and by typing. The editor is accessed from the arguments button  to the right of the Arguments line as shown to the right: Initially, the dialog is blank. The user adds arguments by clicking on the green check button.  The red minus  is used to remove arguments and the arrows   are used to re-order arguments. The default type of an argument is NUMERIC. To change this, use the pull-down menu and select a different type. To change the name of the argument, double click on the name and type a new name. Click **OK** when finished. The screenshot shown to the right results in the argument shown below.



Typing was the original way to enter arguments. The user types into the Arguments line using the syntax `TYPE argument`. Multiple arguments are separated by commas. For example, `NUMERIC flow, LIST elevations, OBJECT res`. The syntax, spelling, and capitalization must be exact or an error will be issued.

Functions are constructed like blocks and use the RPL Palette to build arguments. Below is a sample function constructed using predefined functions and RPL Palette buttons.

The same short cut copy and paste abilities seen in blocks apply to functions as well. Any time an expression is used multiple times, whether within a function or between functions, time can be saved by copying and pasting that expression into the other locations in which it is used. As with blocks, it is wise to check the validity of an individual function after it is built and fix it rather than wait until the beginning of a run to check the validity of all blocks and functions.



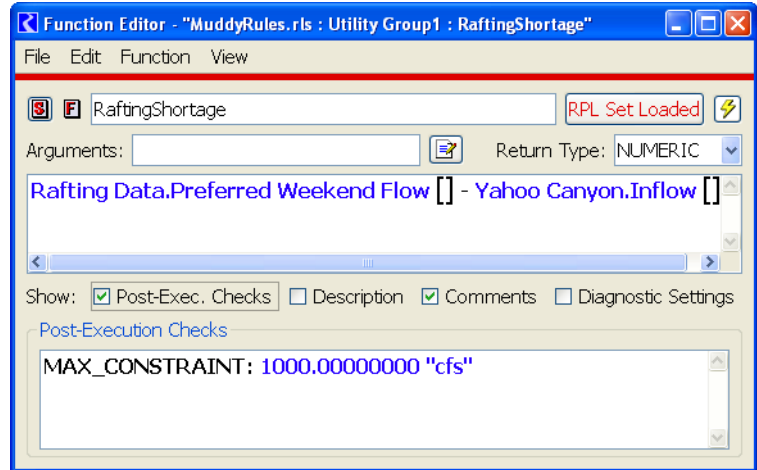
### 1.10.3 Constraints on Functions

Functions can be constrained to evaluate to either a minimum or maximum value. Or they can be configured to flag an error if a minimum or maximum constraint has been violated.

- From the Function Editor, Select **View** ➔ **Show Post-Execution Checks** or click the **Post-Exec Checks** toggle.

A constraint can be added using the **Function** menu item.

- Select **Function** ➔ **Add Min Constraint** or **Add Max Constraint** (as shown in the screenshot). The palette is then used to set up the conditions of the selected constraint.



### 1.10.4 Time Invariant Functions and Function Value Caching

The function editor's Edit menu provides a toggle control labelled "Set Time Varying" which is set to **on** by default. Toggling this property **off** communicates to RiverWare that the function is guaranteed to evaluate to the same value each time it is evaluated, i.e., it is *time invariant*. Note that functions with arguments will almost certainly not be time invariant; if a function has an argument, then presumably there are some argument values for which the function will evaluate to different values. If a function with no arguments is time invariant, then the first time the function is called within a run, the body will be evaluated and the result saved internally. For all subsequent calls of that function within the run, the cached value will be returned without further computation, reducing computation time. Note that incorrect application of caching to a time varying function will lead to incorrect results, so we recommend that the time varying property be toggled **off** for a function only when it is definitely time invariant, run time is critical, and RPL set analysis has indicated that a significant portion of the run is spent evaluating the function.

Note that during block evaluation (e.g. within a rule or accounting method), the workspace remains unchanged (because RPL expressions evaluate without side effects), so it is safe to cache values for functions without arguments within a single block. RPL does this automatically for all functions without arguments; multiple evaluations of such a function within the same block execution will cause the function to evaluate only once.

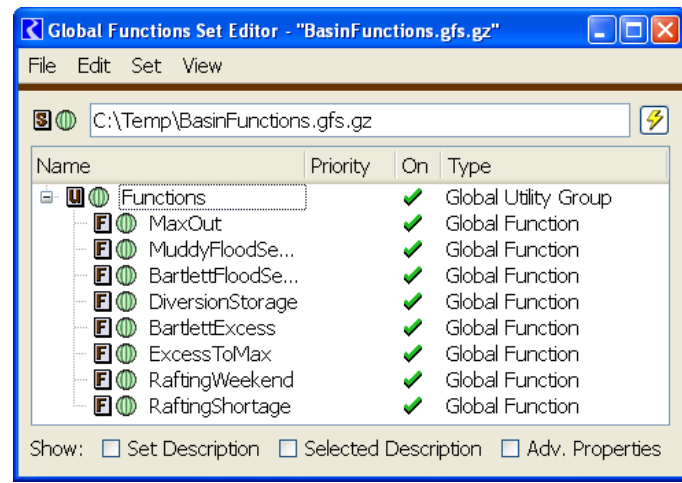
## 1.11 Global RPL Functions

Global functions allow the user to define a function in one location and then use it in any of the RPL applications including Rulesets, Optimization Goal Sets, Object Level Accounting Method Set, Initialization Rules, MRM Rules, and RPL Expressions Slots.

Global RPL Functions exist within Global Functions Sets, organized in Global Utility Groups. Multiple Global Function Sets can be opened within a RiverWare session.

A RPL set is considered “global” when it is opened as a Global Set and any RPL Set file can be loaded as a Global Function RPL Set. (Only the RPL Set’s Utility Groups -- and not its Policy Groups -- are relevant in Global Function Sets).

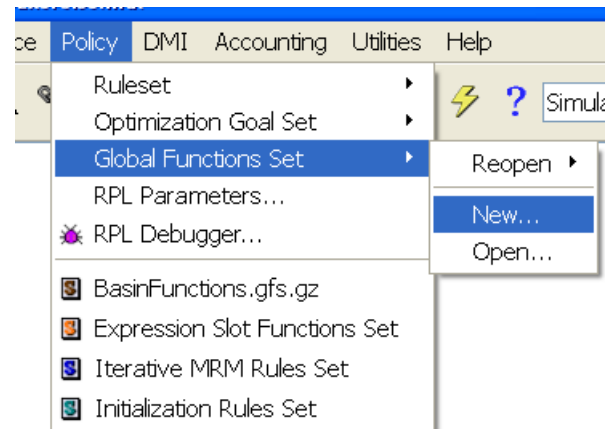
Global RPL Functions can be called from RPL Functions and Rules (and other forms of RPL Blocks) in any open RPL Set. Global Functions can call other Global Functions, either in the same Global Function Set, or a different Global Function Set.



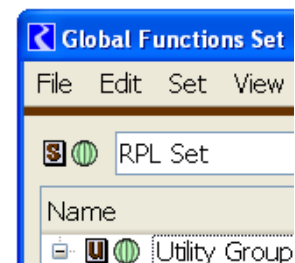
### 1.11.1 Creating a new Global RPL Function Set

A new Global RPL Function Set can be created from the RiverWare Workspace Menu, with this menu item:

Policy ➔ Global Function Set ➔ New...



This opens up a Global Function Set editor. This is a RPL Set Editor for Global Function Sets, distinguished with a *green globe* icon in the upper-left area of the editor dialog:



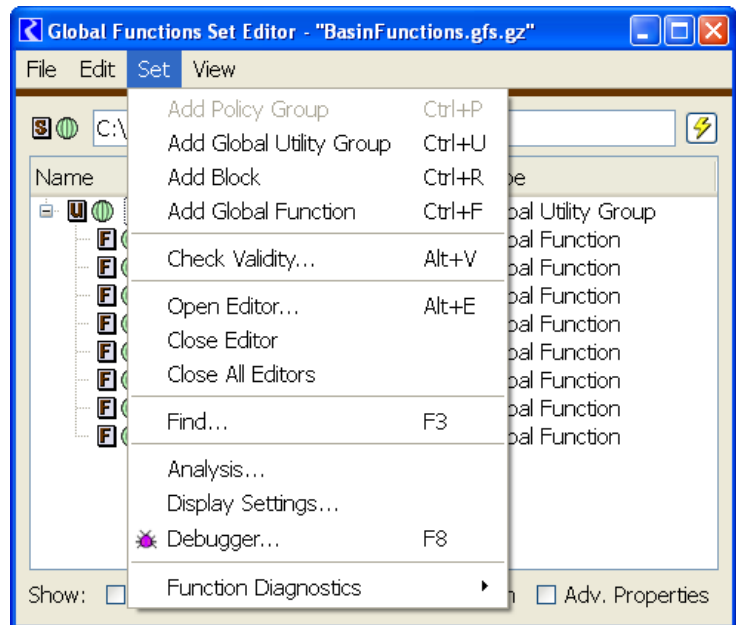
As with all RPL Sets, a Global Utility Group must first be added before new user-defined RPL Functions can be defined. In a Global RPL Function Set, this is done from the RPL Set Editor's menu item:

### Set ➤ Add Global Utility Group

Any function added to the new Global Utility Group is, by definition, a Global Function. A Global Function can be created in any of the following ways (not illustrated):

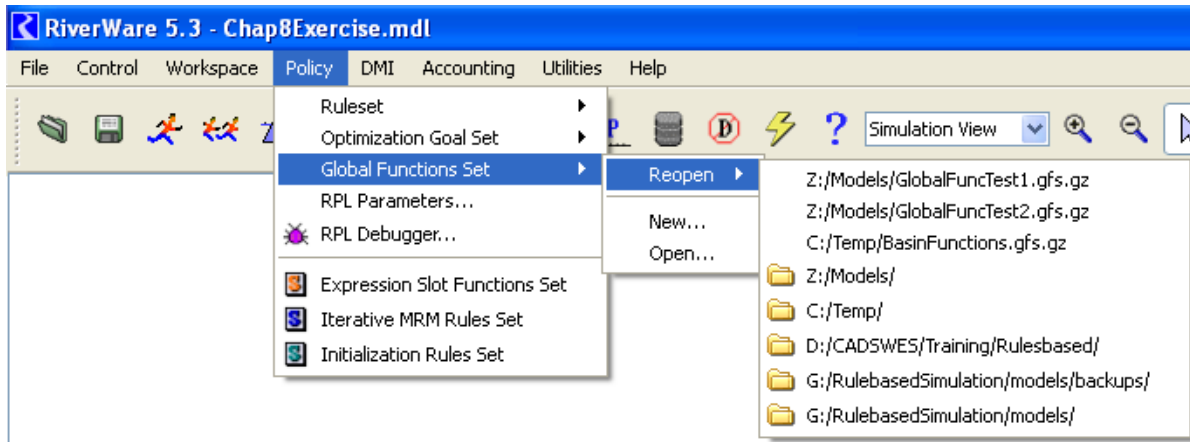
- From the RPL Set editor, with the desired Global Utility Group selected, using the Set ➤ Add Global Function (Ctrl+F) menu item.
- From the RPL Set editor, using the Context Menu (right-click) on the desired Global Utility Group item, Add ➤ Global Function.
- From the Global Utility Group Editor, using the Group ➤ Add Global Function (Ctrl+F) menu item.
- From the Global Utility Group Editor, using the Context Menu (right-click) anywhere within the function list, Add ➤ Global Function.
- By Copying a Function from another RPL Group or RPL Set, and Appending into the Global Utility Group, using Context Menu operations.
- By Drugging a Function from another RPL Group or RPL Set into the Global Utility Group. *NOTE:* The Drag-and-Drop functionality currently implemented in the RPL Set and RPL Group editors has some limitations, including the availability of that operation only if the target function list already has at least one function.
- By importing Utility Groups from a RPL Set file. This is done from the RPL Set editor, File ➤ Import Set... menu item.

Only one instance of a function can occur within all loaded RPL sets and Global Function Sets. For example, you cannot have a function named "GetMinimumFlow" in a utility group in your ruleset and another named "GetMinimumFlow" in an open Global Function Set. An error will occur in this case.



### 1.11.2 Opening an Existing Global Function Set

The RiverWare Workspace Policy menu supports the opening of a RPL Set as a Global Function Set in ways similar to opening other RPL Set files.



Any RPL Set file can be loaded as a Global Function Set, however only the Utility Groups within the RPL Sets loaded as Global Function RPL Sets are usable -- *Rules (or other forms of RPL Blocks) within the set's Policy Groups are not.* If a RPL Set file containing Policy Groups is opened as a Global Function Set, the illustrated warning dialog is shown, and a similar warning message is written to the RiverWare diagnostics output.



It is important to open any Global Function Sets before trying to load a ruleset that uses functions in a global set.

A distinct file and directory "history" for opening and saving is maintained for Global Functions Sets. This history is saved in user login-based settings (implemented with Qt QSettings). When a file dialog is shown for loading or saving a Global Function Set, the default filename filter shows files with the ".gfs" or ".gfs.gz" (gzipped) file extensions. These are only recommendations for filename extensions for Global Function Set files -- the user can use any filename extension for Global Function Set files.

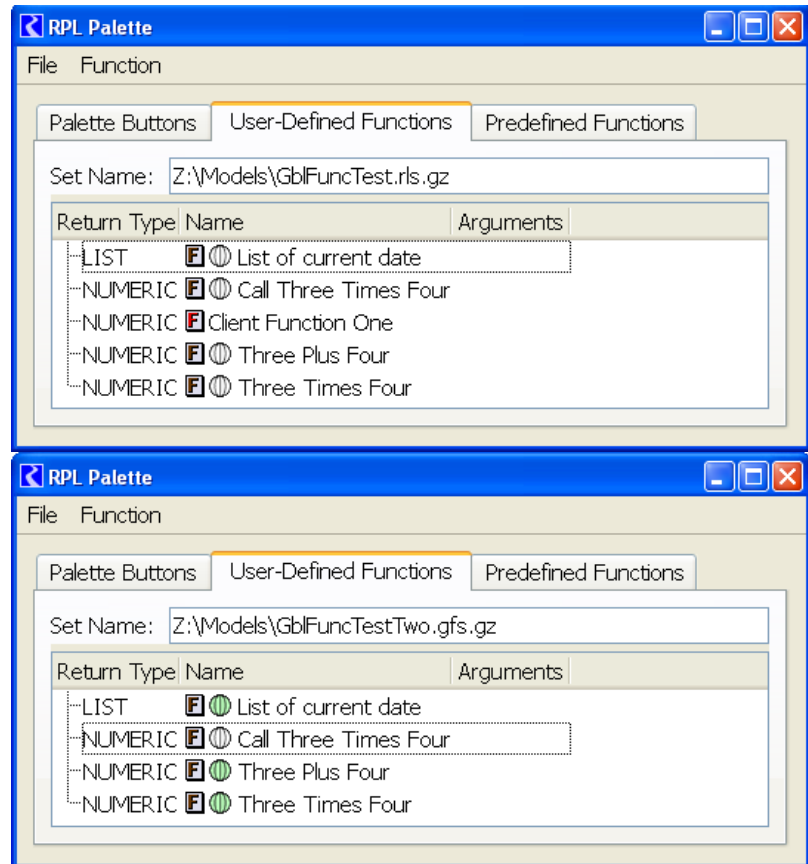
### 1.11.3 Using Global RPL Functions

When editing a RPL Expression, any Global Function in any open Global RPL Set can be called. The standard way of adding a call to any user defined function is via the RPL Palette's User-Defined Functions tab.

Within the User-Defined Function list, global functions are indicated with one of two *globe* icons:

- A **white globe** -- indicates an “external” Global Functions... i.e. a global function in a RPL Set other than the set containing the RPL expression being edited.
- A **green globe** -- indicates a “local” Global Function... i.e. a global function within the same RPL Set as the one containing the RPL expression being edited.

During evaluation of a Global RPL Function, the specific behavior is consistent with the caller's RPL Application. For example, “@t” (the current timestep) represents the Rulebased Simulation Controller's current timestep when the function is called from an Rule. But when called from a Series Slot with RPL Expression, “@t” (current timestep) represents a particular series timestep date/time.



### 1.12 Example: Creating a new ruleset

Following is an example of how to build a new ruleset. Similar actions can be used to build any of the other types of RPL sets.

- **Policy** ➔ **Ruleset** ➔ **New...** from the main menu bar. This will bring up a blank RPL Set Editor dialog.
- Save the ruleset to the desired directory by selecting **File** ➔ **Save** or **File** ➔ **Save As** from the menu bar of the Ruleset Editor dialog. The name the ruleset is saved with becomes the name of the ruleset.

**Adding Groups, Rules, and Functions:** To begin the ruleset, add the desired number of policy groups and utility groups by selecting:

- **Ruleset** ➔ **Add Policy Group** or **Ruleset** ➔ **Add Utility Group** from the RPL Set Editor's main menu bar.

- Name the Policy or utility Group by double clicking on its icon to open the Policy Group Editor or Utility Group Editor. Right clicking on an icon will bring up a menu from which the Policy Group or Utility Group Editor can be accessed. Name the Policy or Utility Group and type <Return> after entering the name.

Policy groups are used to organize blocks and functions. Utility groups are used to organize functions. Although an entire set can be created entirely within a single policy group, it is not recommended. Blocks should be split up into different policy groups according to object, geography of the model, operation type, and/or priority of operations. Supporting functions which are common to several rules should be placed in utility groups.

To add a block or function:

- Click on the Policy or Utility Group so that it is highlighted.
- Select **Ruleset ➤ Add Rule** or **Ruleset ➤ Add Function**.

To view the elements in the policy or utility group, click on the tree-view triangle left of the group name.

Unspecified expressions are portions of the block that have not yet been set when building a rule. These expressions are represented as the expression data type.

- Add an Assignment using the **Rule ➤ Add Assignment** menu.

Expressions are built using the RPL Palette. The RPL Palette contains most of the expression elements needed to build a block, including logical operators, slot and object access expressions, flow operators, list operators, predefined and user made functions. Click [HERE \(RPLTypesPalette.pdf, Section 2\)](#) for more information on the palette. Depending on the data type of the expression highlighted in a block, the RPL Palette only enables buttons for expressions of that data type. The data types are described [HERE \(RPLTypesPalette.pdf, Section 1\)](#).

To open the RPL Palette:

- select **Rule** → **Palette** from the menu bar of the **Rule Editor** dialog.

To finish building the rule:

- Highlight the left **<numeric expr>**. This will be the slot to which the rule will assign a value.
- In the **Objects/Slots** section of the RPL Palette, click on the **Slot[]** button. This replaces the unspecified numeric expression with a series slot expression.

The square braces following a slot expression refer to the timestep or the row and column of the slot. There are three slot expression possibilities:

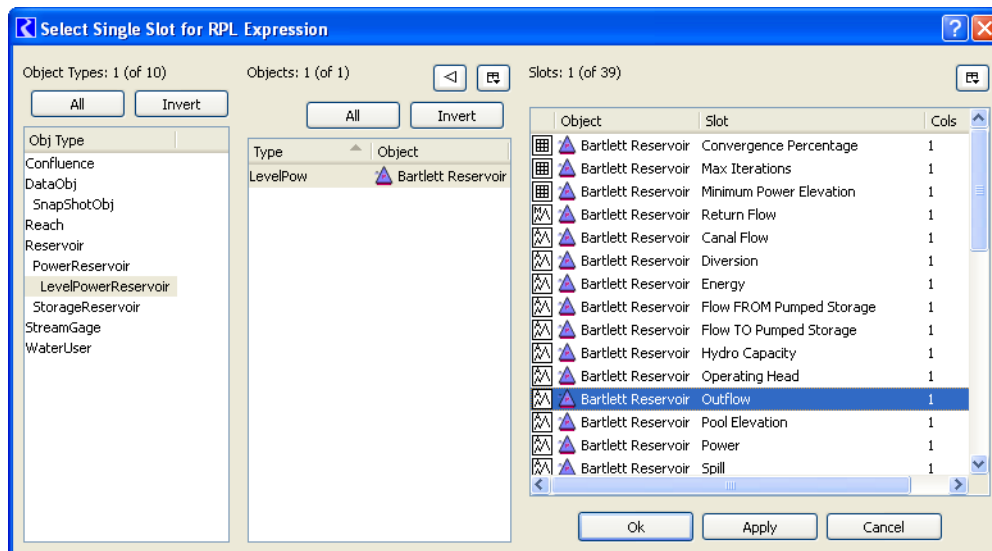
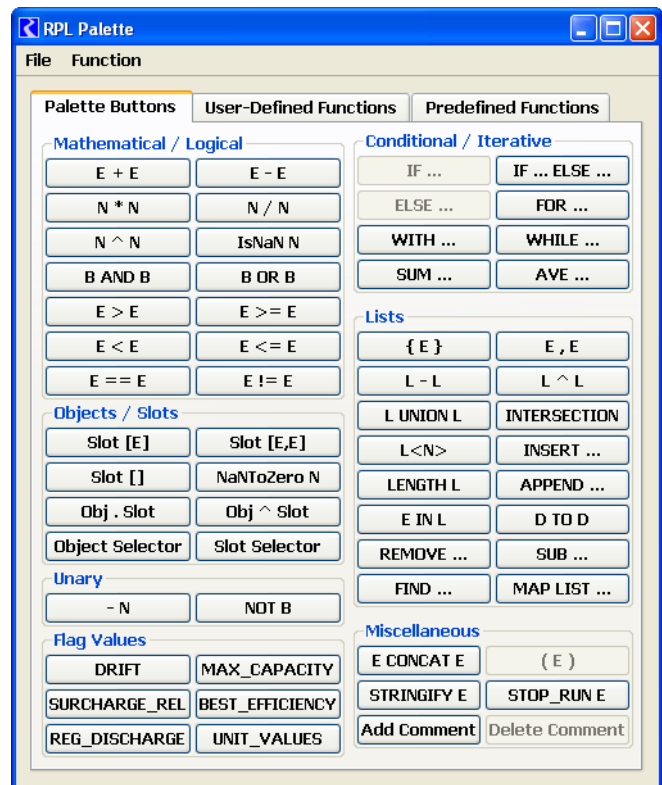
**Slot [ E ]** A series slot at the timestep indicated by **E**.

**Slot [ E, E ]** A table slot at the row and column indicated by **E** and **E**, respectively.

**Slot [ ]** A series slot at the current controller timestep.

Next:

- Highlight the unspecified portion of the series slot expression, **<expr>**, and click on the **Slot Selector** button in the **Objects/Slots** section of the RPL Palette. This will bring up the slot selector dialog.



- In the Single Slot Selector, select the appropriate **Object Type**, **Object**, and **Slot**. Then click **OK**.

Finish building the rule by building the right-hand side of the rule. The RHS should return a numeric value and units that are to be set on the slot specified on the left-hand side.

## 2. Developing Efficient RPL Expressions

RPL expressions are built using the palette ([HERE \(RPLTypesPalette.pdf, Section 2\)](#)). Together with user and pre-defined functions, [HERE \(RPLPredefinedFunctions.pdf\)](#), these provide all of the pieces necessary to create a simple or complex RPL expression. Typically, an analysis of your RPL set's performance is necessary to locate slow or inefficient items. Tools for this can be found [HERE \(RPLDebugging.pdf, Section \)](#) and in particular the **RPL Analysis Tool** located [HERE \(RPLDebugging.pdf, Section 5\)](#).

Following are some suggestions to writing efficient RPL sets in terms of performance.

- Use predefined functions or operators when available. In general, always prefer a built-in operator or predefined function to a user-defined function (or complex expression) which performs the same computation.
- Move complex logic to user defined functions. This not only makes the logic more readable and easier to debug, but it makes performance analysis easier. This is because the RPL set analysis tool reports the times for function calls but not for other parts of expressions (i.e., the granularity of the performance information is “per function”).
- Use WITH expressions to avoid re-evaluating expensive expressions.
- Make sure there is no unnecessary LIST processing or STRING manipulation.
- OBJECT and SLOT representations of workspace objects and slots are generally more efficient than string representations of them. For example, the expression

```
GetSlot( (STRINGIFY reservoir) CONCAT “^” CONCAT account CONCAT “.Storage” )
probably takes significantly longer to evaluate (and is more complex) than
reservoir ^ ( account CONCAT “.Storage” )
```

- For a large model, it is computationally expensive to obtain a SLOT given the full slot name (i.e., a STRING representation of the slot) because RiverWare must first break the string into its components, then look for an object with the appropriate name. Once the object is found, then for accounting slots RiverWare must search through the accounts. Finally a search is conducted for a slot on the object/account with the given slot name. Thus when referencing slots, one should take special care to apply the suggestions mentioned above.
- Make sure that time consuming functions like FloodControl() or the hypothetical simulation functions are not being called more than necessary.
- Use functions with no arguments to cache values. If a function has **no** arguments, then the first time it is executed in a block (a rule, accounting method, optimization goal, initialization rule, MRM rule or expression slot), the return value is cached. For the remainder of the block execution, the function need not be evaluated again, the cached value is used. Thus, if you have multiple assignments within one rule that call the same argument-less function, the function will be evaluated once and the value will be used for all function calls.
- If you are sure that a function with no arguments is always going to return the same value regardless of the timestep, set the **Set Time Varying** toggle to be off. This will lead to the function's first return value being reused throughout the run. Click [HERE \(Section 1.10.4\)](#) for more information on this feature.

- MAPLIST is an efficient expression which operates on a list *and* returns a list, but if one wants to operate on a list and compute a single value, FOR expressions (or a variation of FOR, like SUM or AVE) are more efficient. E.g.

```
Sum( MAPLIST (...))
is slower than
FOR ( ... ) SUM
...
END FOR
```

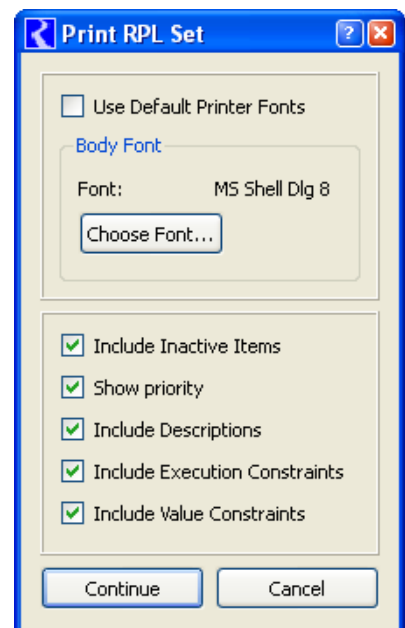
### 3. RPL Printing and Formatting

Blocks written in the RiverWare Policy Language can be complex, and with lengthy verbose object, slot, and function names, blocks can have very long lengths. RiverWare now provides a mechanism to print blocks and dynamically format blocks to fit the width of desktop window or a printer page.

#### 3.1 Printing

Rules and user defined accounting methods written in the RiverWare Policy Language can now be printed. A Printing dialog is available at each editor level from the **File** ➔ **Print X** menu. This provides the user with the means to print an individual block from the Editor, an individual function from the Function Editor, an individual group from the Group Editor, or the entire RPL Set from the RPL Set Editor.

Using the print dialog, the block, function, group, or RPL set can be sent directly to the printer or to a file. The user may select landscape or portrait printing. The user may choose to include or exclude inactive items, descriptions, execution constraints, and value constraints. The user may also choose the print font. A note on font selection. The font selection dialog presents a list of fonts that are available on computer console, these fonts may or **may not** all be available on the printer. If the font is not available on the printer, a default font will be chosen by the printer and the printout may be misaligned -- formatting will be corrupted. Some experimentation may be necessary to determine which fonts an individual printer supports.




#### 3.2 Formatting - The Display Settings

Expressions written in the RiverWare Policy Language can be complex and very long. RiverWare provides a **RPL Display Settings** dialog to control the fonts and colors and a dynamic formatting algorithm. This algorithm formats the blocks to fit the width of desktop window or a printer's page. The dialog is accessed through the **Display Settings...** button in the editor's **Set** menu.


Display settings may be imported and exported to/from an ASCII file through the “File” menu. These preferences will be automatically saved in user’s preference file.

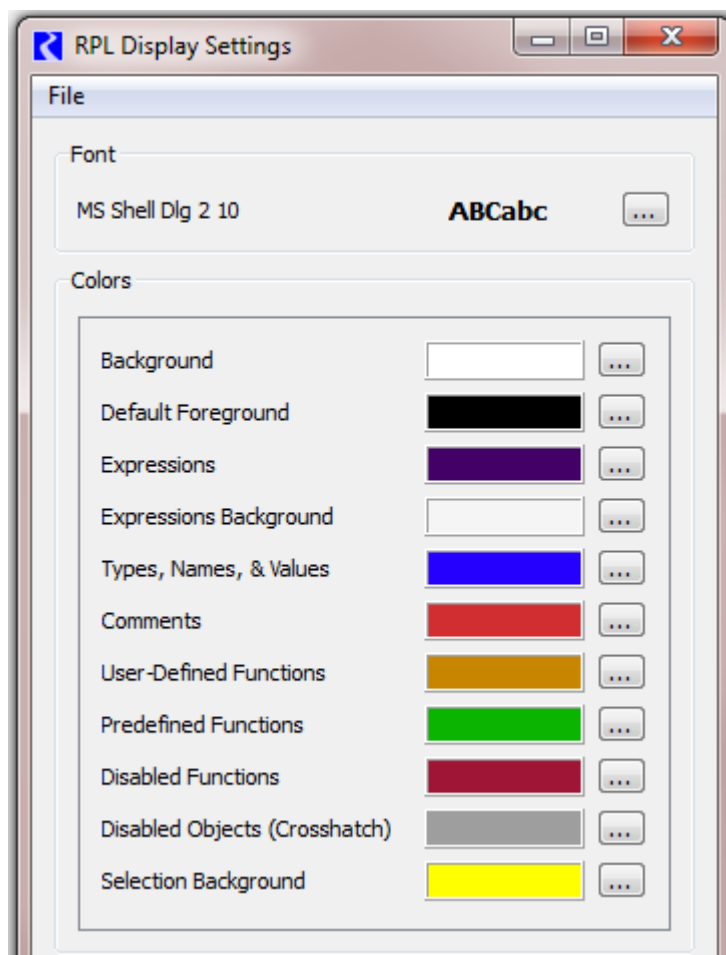
There are three area to this Display Settings dialog, **Font**, **Colors**, and **Line Breaks** described as follows:

### 3.2.1 Font

The fonts used in RPL expressions can be changed by clicking on the “...” button,  then select the desired font, style, size, and effects.

### 3.2.2 Colors

The colors used in RPL set editors can be changed to fit your needs. You are able to specify the colors for items shown in the following table. To change, click on the “...” button, , then select the desired color from the color chooser.



Item	Description
<b>Background</b>	The background color for statements and other areas (except expressions) that are not contained within the current selection.
<b>Default Foreground</b>	The color used to draw items in expressions or statements whose color is not governed by another color setting. For the current selection, this color is used for all items; outside of the current selection, this color is used to draw mathematical symbols, operators, and delimiters. Note, the selected expression’s foreground text is either white or black (dynamically computed) to contrast with the configured background color.
<b>Expressions</b>	The color for key words in expressions.
<b>Expressions Background</b>	The color used for the background of the bounding rectangle for expressions.

Item	Description
Types, Names, & Values	The color of types (e.g., NUMERIC), names of variables, and literal values.
Comments	The color of in-line comments.
User-Defined Functions	The names of user-defined functions.
Predefined Functions	The names of predefined functions.
Disabled Functions	The names of user-defined functions which are disabled (i.e., functions for which the red X appears in the “On” column of the functions RPL Set and Group editors).
Disabled Objects (Crosshatch)	The color of the crosshatching superimposed on disabled statements or expressions.
Selection Background	The background color for statements and expressions that are contained within the current selection.

### 3.2.3 Line Breaks

The formatting algorithm is designed to provide the user with control over the formatting process. The formatting algorithm is multi-pass algorithm. Each pass will attempt to find a position in the block where a line break can be placed to fit the block to the device width. The algorithm will cease, once the block can be rendered in the desired width, or no more positions where the block is allowed to be broken exist. The user can define, through the **RPL Display Settings** dialog, the tokens (i.e., block positions) a line break can appear before or after, the indentation following the line break, and the priority (the pass) at which the line break should be inserted. Line breaks with a priority of zero will always be broken, while the maximum value will indicate that a line break at the token should never occur.

Token	Indent Before	Break Before	Indent After	Break After	Priority
*	0	✗	0	✗	3
+	0	✓	0	✗	3
,	0	✗	0	✓	1
-	0	✓	0	✗	3
<	0	✗	0	✗	6
<=	0	✗	0	✗	6
<assignment =>	5	✗	8	✗	2
<comment>	0	✓	0	✓	Always

A block is formatted using the following algorithm:

1. The formatting algorithm makes an initial pass, breaking the block at priority 0 tokens. All line break defined with priority 0 (i.e., Always) in the **RPL Display Settings** dialog will be formatted in this initial pass. This pass always occurs.
2. Next the formatting algorithm checks the length of block. If the block will fit in the width of device, the formatting terminates. If the block will not fit the width of the device, the priority is incremented, and line breaks are made at all tokens defined with priority 1.
3. Repeat step 2 through all priorities until the block fits in the device.

If the block exceeds the screen width after all line break positions have been exhausted, the window will be made scrollable. If the block exceeds the printer's page width after all line break positions have been exhausted, the excess will be printed on adjacent pages.

To edit any number in the dialog, click on the number and then type in a new value. You can type in either "Always" or 0 for very high priority values. Type in "Never" or a large number (greater than about 40) for those items with low priority.

---

Example:

Given the line break settings:

Token	Break Before	Break After	Priority
THEN	no	yes	Always (0)
END IF	yes	yes	Always (0)
+	yes	no	1
AND	yes	no	2

The block being fitted to the box:

Before Formatting:

```
IF (SomeFunctionWithAVeryLongName() AND AnotherFunctionWithAVeryLongName) THEN Some-
ObjectSlotWithAVeryLongName + AnotherObjectSlotWithALongName END IF
```

After the initial pass (priority 0):

```
IF (SomeFunctionWithAVeryLongName() AND AnotherFunctionWithAVeryLongName) THEN
    SomeObjectSlotWithAVeryLongName + AnotherObjectSlotWithALongName
END IF
```

After the second pass (break on all priority 1 positions):

```
IF (SomeFunctionWithAVeryLongName() AND AnotherFunctionWithAVeryLongName) THEN
    SomeObjectSlotWithAVeryLongName
    + AnotherObjectSlotWithALongName
END IF
```

After the third pass (break on all priority 2 positions):

```
IF (
    SomeFunctionWithAVeryLongName()
    AND AnotherFunctionWithAVeryLongName()
) THEN
    SomeObjectSlotWithAVeryLongName
    + AnotherObjectSlotWithALongName
END IF
```

Rule Fits the box. Stop.

Notice that with this naive algorithm, had the object slot names been small, they would have still been broken at the “+” symbol, since, the “+” had a higher priority (1) than the “AND” and the block did not yet fit the box.

Before Fomating:

```
IF (SomeFunctionWithAVeryLongName() AND AnotherFunctionWithAVeryLongName) THEN Object
Slot + AnotherObjectSlot END IF
```

After the initial pass (priority 0):

```
IF (SomeFunctionWithAVeryLongName() AND AnotherFunctionWithAVeryLongName) THEN
    ObjectSlot + AnotherObjectSlot
END IF
```

After the second pass (break on all priority 1 positions):

```
IF (SomeFunctionWithAVeryLongName() AND AnotherFunctionWithAVeryLongName) THEN
    ObjectSlot
    + AnotherObjectSlot
END IF
```

After the third pass (break on all priority 2 positions):

```
IF ( SomeFunctionWithAVeryLongName()
    AND AnotherFunctionWithAVeryLongName() ) THEN
    ObjectSlot
    + AnotherObjectSlot
END IF
```

Rule Fits the box. Stop.

---

## 4. Exporting and Importing RPL sets

In this section we describe how to export all or a portion of a RPL set to a separate RPL set file, as well as how to import an existing RPL set file. The export and import mechanisms provide a way to incorporate all or some of one policy into another policy, and it is available for all application of RPL within RiverWare, including:

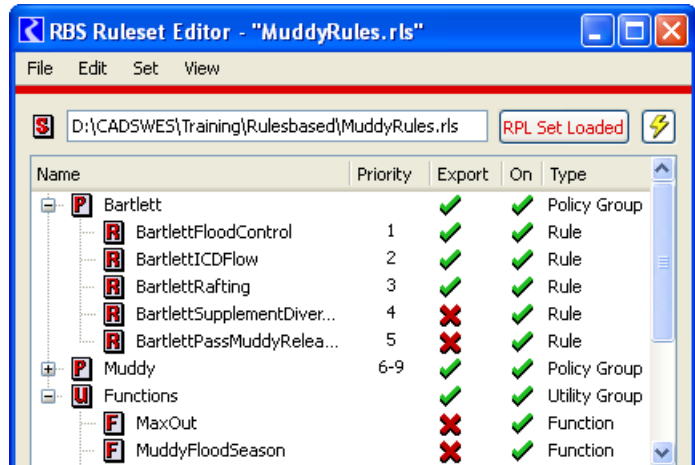
- Rule sets - collections of rules and functions organized into policy and utility groups.
- Optimization goal sets - collections of optimization goals and functions organized into policy and utility groups.
- The object-level accounting method set - a collection of methods and functions organized into pre-defined category groups and utility groups.
- The Expression Slot set - a collection of functions organized into utility groups. These functions may be called from an expression defining an expression slot.
- Initialization Rules - collections of rules and functions organized into policy and utility groups.
- Iterative MRM RPL sets - collections of rules and functions organized into policy and utility groups.
- Global Function Sets - collections of functions organized into Global Utility Groups.

There are many situations in which export and import functionality is useful. For example, assume that you are writing a policy which could make use of some functions written as part of an existing policy. To avoid having to rewrite all these functions, you could open both sets and use the Copy/Paste mechanism to copy the functions in question from the existing policy to the new one. However, if you then modify the original functions, you might want to do the copy again to keep the functions identical, but you would have to first remove the copies of the functions before doing this. Furthermore, in the context of the object-level accounting method set it is not possible to have two such sets open simultaneously, so this mechanism would not work. The RPL set export and import functionality provides a simpler and more flexible method to implement sharing of policy elements. Alternatively, you could use Global Utility Groups described [HERE \(Section 1.11\)](#).

## 4.1 Export

The RPL set export mechanism allows the user to save a portion of a RPL set as a separate RPL set file. The first step is to show the Export column of the RPL set editor. If this column is not currently shown (between the columns labelled “Priority” and “On”), then it can be shown by selecting **View ➤ Show Export Column**. Initially no item is selected for export, this is indicated by a red “X” in the Export column for each item.

Click in the Export column of the items that you would like to export. This will convert the red “X” in that cell to a green check mark. Note that selecting a block or function for export automatically selects the containing group, and unselecting a group will unselect all of its members.

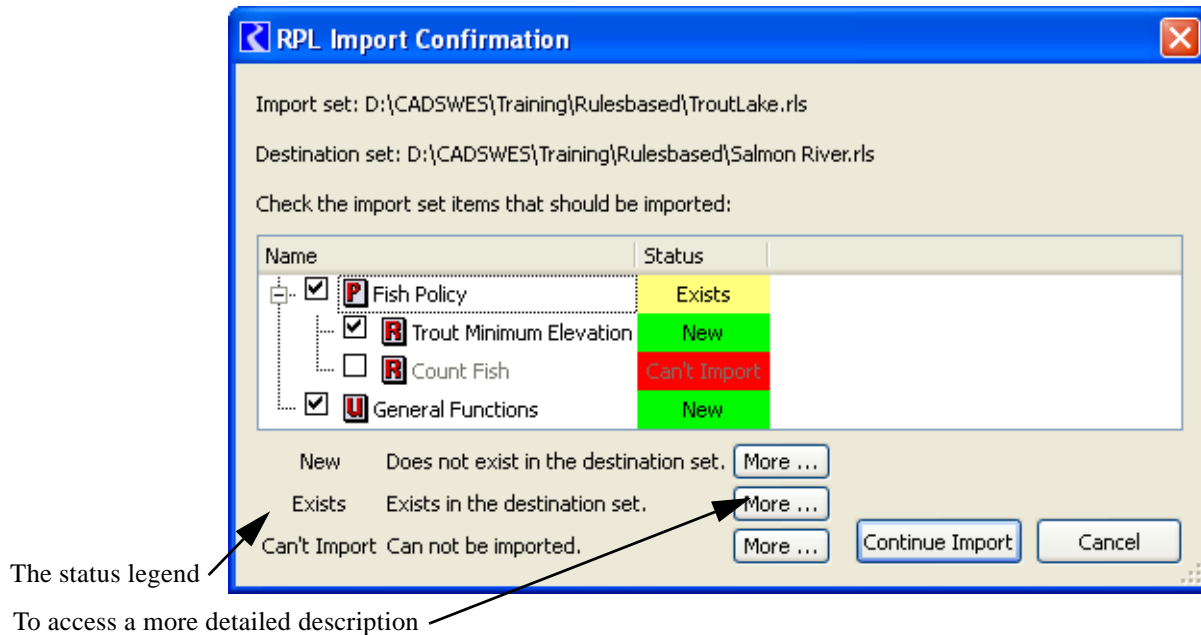


Once the items to be exported have been identified in this way, the export operation is initiated by selecting **File ➤ Export Selected Items**. You will then be presented with a confirmation dialog summarizing the items to be exported. If you choose to continue the export operation, a file chooser dialog appears, allowing you to specify the file to which the items will be exported. Once a file name has been selected, the file is created and a RPL set containing only the selected export items is written to that file.

## 4.2 Import

RPL set import allows you to incorporate all or some aspects of an existing RPL set policy into a set that you are editing. To initiate an import, select **File ➤ Import Set**. You will be presented with a file

chooser dialogue for specifying the name of a RPL set file. Once you have selected a file and clicked the **Open** button, you are presented with the RPL Import Confirmation Dialog.



This dialog presents a listing of the import set. This layout is similar to the RPL set editor's presentation of the set, but each item being imported is classified as having one of the following statuses:

- **New** - The group, block, or function being imported does not appear to exist in the set into which it is being imported (the destination set).
- **Existing** - The group, block, or function being imported appears to exist in the set into which it is being imported (the destination set). In the case of a group, there is a group of the same type with the same name in the destination set. In the case of a block (or object-level accounting method), there is a block with the same name in a group which is of the same type and has the same name as the block's parent group. Similarly, in the case of a function, there is a function with the same name in a group which is of the same type and has the same name as the function's parent group.
- **Can't Import** - The group, block, or function being imported can not be imported. There are several reasons why this might be the case and a warning is posted to the Diagnostic Output Dialog for each item that can not be imported, explaining the problem. Possibilities include an attempt to import a TCL function (not supported) or an attempt to import a function into a policy group when there is already an item of the same name in the global name space, e.g., a function of the same name exists in a utility group.

Before continuing with the import operation you should review the items being imported and indicate which items should in fact be imported. To select an item for import click in the box to the left of its name in the list. Importing a new group will append it to the list of groups; importing a new function or block into an existing group will append it to the groups of functions which already exist. Importing an already existing group will copy its description to the existing group and copying an existing block or function will replace the existing block or function with the imported block or function. Initially new items are selected for import and existing items are not, but these selections can be modified. To complete the import click on the **Continue Import** button. Clicking on the **Cancel** button at any point will cancel the import operation and leave the RPL set unchanged.

# RPL External Documentation

---

## 5. RPL External Documentation

### 5.1 Overview

This document describes RiverWare's capability to link a RPL set to documentation that resides in a separate file viewable by commonly used viewing and editing programs.

RiverWare's RPL editors support an optional multiple-line text description, displayed in a plain-text editor box and is stored in the RPL set file. Often this text description is not sufficient to adequately describe a complicated rule or policy set. The purpose of a link to external documentation is to allow the user to develop more detailed documentation that will reside in a separate application like MS Word, PDF, or an HTML file. From the RPL editors, the user can click to go to the documentation file. Note, the external documentation support described in this document is distinct from this description text value.

A RPL set consists of rules (or user-defined accounting methods) and functions organized into groups. In this document, the term RPL **object** refers generically to a RPL set component of any sort, i.e., to the Set, Policy or Utility Group, Function, Rule, or Method.

The following is an overview of the capabilities:

- From a RPL dialog, the user is able to easily access external documentation for the RPL object associated with that dialog.
- The four supported types of documents are:
  - HTML,
  - MS Word,
  - PDF, and
  - plain Text
- The user is able to specify the applications for both viewing and editing each type of document
- Two modes of access are provided:
  - Edit - the user can view, create, and change the contents of the document.
  - View - the user can view but not change the contents of the document.
- The granularity of the documentation is flexible. Users can associate a separate document with each RPL object within a RPL set or they can document an entire RPL set with a single document.
- When using HTML, the application can open to the most relevant portion of the document. This applies when:
  - There is more than one RPL object described in the associated document, e.g., the documentation is a single file describing the entire ruleset, and,

- The user is using standard named anchors within the documentation to describe the sections that pertain to each RPL objects.
- The user can create an HTML template of the RPL set that contains the description field for each object. The user can then expand the documentation using an HTML editor.

The following sections describe the external document link feature. Configuration of this feature has two parts: first, the user must specify the application to be used to edit and/or view one or more types of files; second, the user specifies how the document is structured by associating RPL objects with external files.

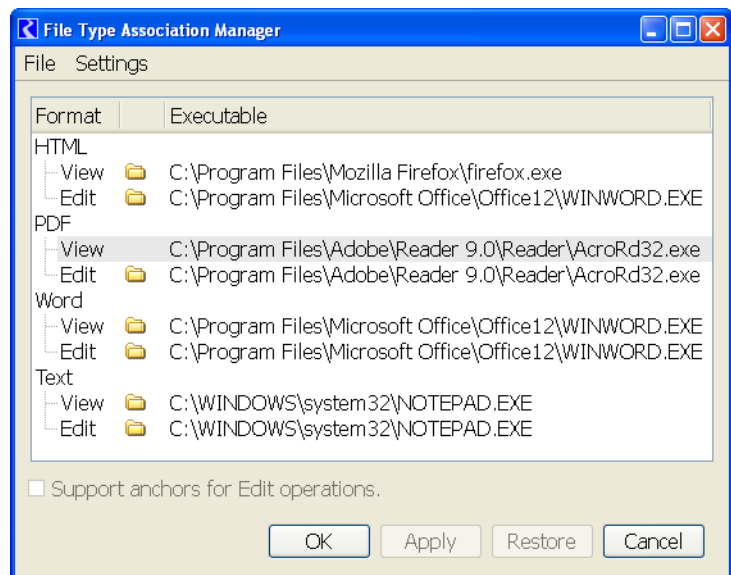
## 5.2 File Association

The RiverWare **File Type Association Manager** is used to specify the executable programs used for viewing and editing documents of the four supported Format Types (HTML, PDF, Word, or Text). The file paths are stored with the user's settings. It is accessible from:

- The RPL editor **View** ➤ **External Documentation** ➤ **File Type Associations** menu
- The workspace **Utilities** ➤ **File Type Associations** menu
- The **Utilities** ➤ **File Type Associations Manager** from the **Configure External Document Reference** dialog.

Each file format has a distinct Edit and View mode program association. This allows the user to edit with one program and view with another. Only the document formats that the user wishes to use need to be specified. The others can be left as the default or left blank.

Executable file paths can be edited (including pasting from the system clipboard) by double clicking an item within the **Executable** column or can be picked with a File Selector by clicking the folder icon. On Windows, all of the executable file paths can be set to the default executable associated with the user's account using the **Settings** ➤ **Set Default Executables** menu operation. Changes can either be saved (by clicking OK or Apply) or dropped (by clicking Restore or Cancel).



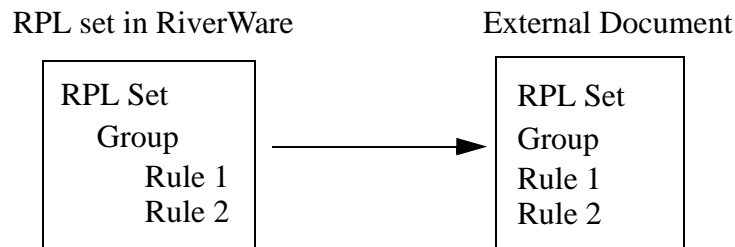
If the ACROREAD\_PATH environment variable is defined, the PDF View item is set to the value of that variable, is non-editable, and is shown with a grey background. The ACROREAD\_PATH is used by RiverWare to locate the acrobat reader path for RiverWare Help and other purposes. It is typically set by the user when installing RiverWare.

Currently, the **Support anchors for Edit operations** check box is not operable. When opening a RPL Object External Document for Editing, the anchor text (used to automatically scroll to a particular section within the document) is only supported for HTML. For all types, the name of the RPL Object name text is copied to the system clipboard and can be used to paste into the “Search” function within the editor or viewer program.

### 5.3 Document Structure

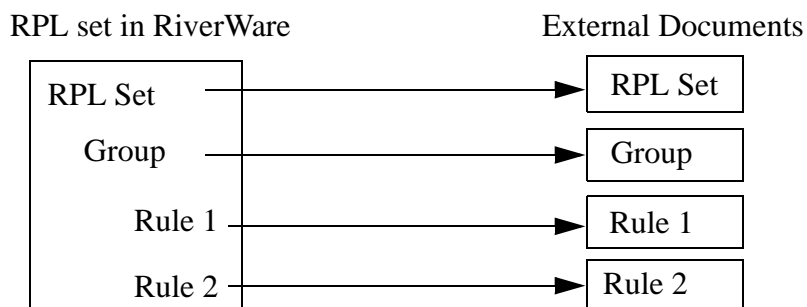
This section provides an overview of the possible document configurations including some advantages and disadvantages to both. Following are setup configurations that have been envisioned:

#### One document that describes the entire set:



- Advantages: one document is typically easier to maintain and share. Also, the user only needs to configure the external link for the overall RPL set, not for each individual RPL object within the set.
- Disadvantages: unless the viewing document is HTML, there is no way for the user to automatically open the document to the correct location. For example, if the user clicks to view a rule’s document (say in PDF) they must still search for that rule within the PDF document. Also, multiple model users have to coordinate if they wish to develop documentation simultaneously.

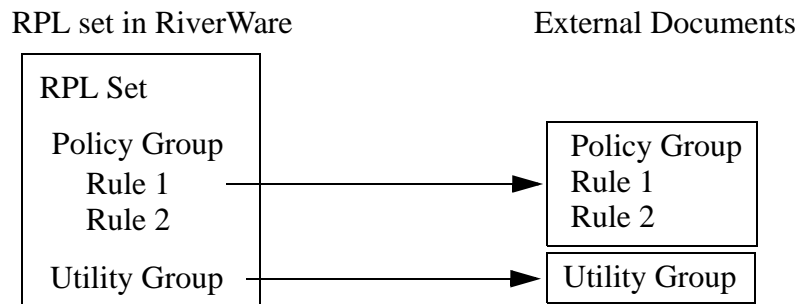
#### One document for each RPL object in the set.



- Advantages: one document for each RPL object allows the user to easily view the external document specific to that object. For example, the user clicks on the view button for a rule and the document specific to that rule opens. Multiple users can more easily develop documentation simultaneously.
- Disadvantage: It is more time consuming to configure and maintain. When sharing, there are

many more documents to pass around. If there are a large number of RPL objects, there will be a large number of documents.

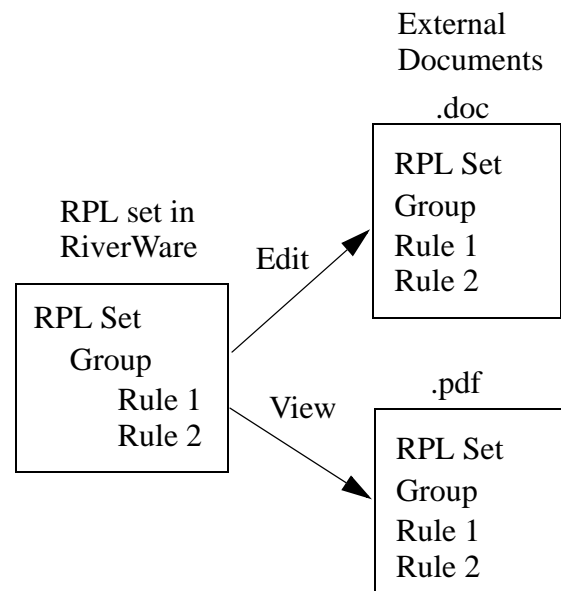
**Multiple documents with one document for each policy group and utility group.** This structure is a hybrid of the two above and has some of the advantages and disadvantages of both. This structure is useful because exceptions to the overall scheme are natural. For example, most of the set could be described by one document, while special objects could be described in a document of their own.



Combinations of the above could be envisioned. The choice of the structure to use depends on the format and the end use.

**Separate View and Edit Documents:** External documentation for a single object might also be found in two different files, one for editing and one for viewing. For example, documentation might be created in MS Word, saved in Word format for the purposes of future edits, and also saved to a PDF file for the purpose of viewing. Thus the utility allows the user to specify two locations, one for the purposes of viewing and another for editing. The following diagram shows an example as two separate documents, one for viewing, one for editing.

When the documentation for an object is split across many files, then it will often be convenient to the users for all the files to be located in the same directory. This organization is supported without requiring the user to provide redundant information (i.e., repeat the full path to the documentation for each object in the ruleset). At the same time, there is enough flexibility so as not to impose a specific organization on the documentation, either that there be a one-to-one correspondence between ruleset objects and documents or that multiple documents appear in the same directory.



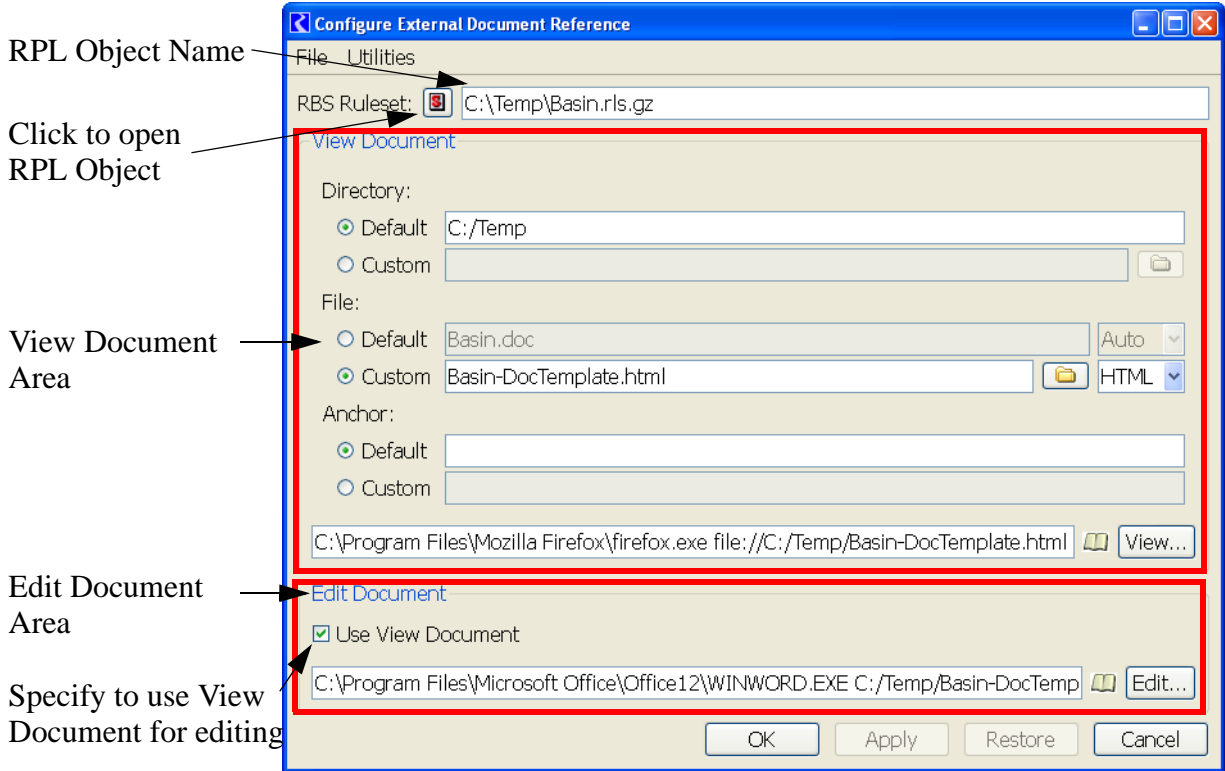
## 5.4 Configuration and User Interface

The document structure is specified in the RPL set by supplying path names to a directory and then specifying file names within that directory. Further, because RPL objects are organized in a hierarchical organization (RPL sets contain policy and utility groups, policy groups contain rules and functions, utility groups contain functions) the hierarchical organization enables RPL objects to share information between objects. When configuration information is provided for an object, the containing objects inherit the configuration from their parent as their default configuration. The user can change the containing objects' configuration as necessary. Thus to specify a single directory in which all the documentation for a RPL set is contained, the user need only specify that value as the directory associated with the set; all groups, functions, and rules in that set will inherit this as their documentation directory.

In addition to providing flexibility with respect to the organization of the documentation, this approach has the advantage that when documentation is moved (say from the modeler's file system to a stakeholder's file system), the ruleset will typically only need to be updated in one place to reflect this change. Environment variables can be used to further simplify moving RPL set documentation between machines. With the use of environment variables, the base directory of all RPL set documents within a users configuration can be set without any changes to the document link configuration within RiverWare. The environment variable substitution mechanism uses unix-style syntax (even on windows): any environment variable name (case sensitive with only alphanumeric characters and underscores) preceded with a dollar sign "\$" is translated to the environment variable defined in the user session, outside of RiverWare. For example, an environment variable name "ALLUVIAL\_RPL\_DOC" is defined as "C:\Projects\docs", then within the RPL external documentation link, "\$ALLUVIAL\_RPL\_DOC\forecastDraft\" will refer to C:\Projects\docs\forecastDraft. This is specifically designed for portability of external RPL documentation across machines and platforms. Slashes can be forward or back slashes and redundant slashes are condensed.

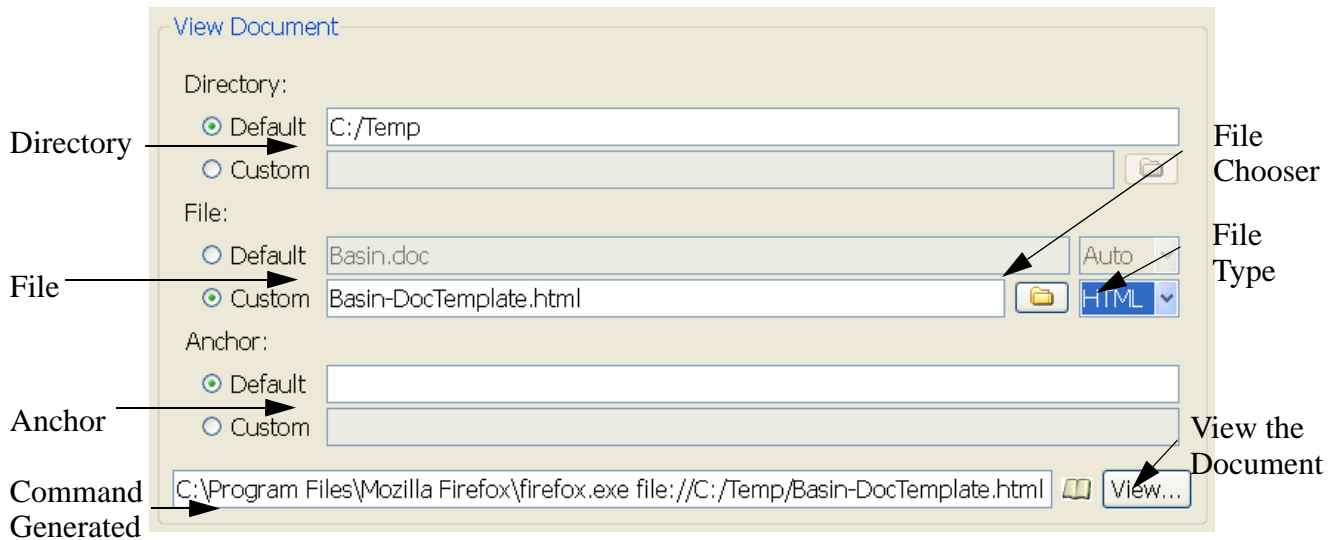
In this documentation and in the user interface, we refer to the two parts of the location specification as the **Directory** and the **File name**. There is nothing that requires that the two parts correspond to a directory and file name in the local file system. Another possibility is that the two parts together specify the location of a document as a URL (Uniform Resource Locator, a type of URI). In this case, the first part to be something like "http://www.WaterU.edu/Models", which is not strictly speak a directory. Note also that the file name specification could be an absolute path specified relative to the directory specification, e.g., "MyFunctions/FunctionA.htm".

The configuration dialog is available from the RPL editor **View** ➔ **External Documentation** ➔ **Configure** menu. The following screenshot shows the dialog for the RPL Set:



In this dialog, the RPL object is listed at the top and there is a button to take you to that RPL object. Then, there are two areas, one for the **View Document** and one for the **Edit Document**. If the **Use View Document** check box in the **Edit Document** area is checked, the remainder of the **Edit Document** area is hidden. This toggle specifies that the **View** and **Edit** documents are the same. The process of

specifying a View and Edit document is the same, so it is only described for the View Document. The View Document contains the following:



For the **Directory**, **File**, and **Anchor** (when using HTML), the user can specify to use the listed **Default** or click to use a **Custom** configuration. When **Custom** is first clicked, the **Default** value is copied down from the **Default** field. For the **Custom** configurations, the user can either type in a value or use the File Chooser button to specify. For the **File**, the user can also specify the file type or use the auto selection. Anchors are currently only supported for HTML files and are described [HERE \(Section 5.5.1\)](#)

The default **Directory** is the directory containing the RPL set itself. The default **File** is the name of the RPL set with a modified filename extension. It is typically “.doc” or “.docx”, but will instead be “.html” if a file with the resulting name at the path actually exists.

For RPL objects (e.g. a group or rule) within a set, the default directory and file names are inherited from the containing RPL object, and the default anchor (when HTML is used) is based on the RPL object name. Note, spaces and most punctuation is removed. Note that the default anchors for all the objects within a RPL set are reported in a table in the generated HTML template file (described [HERE \(Section 5.5.1\)](#)).

At the bottom of the View area is the resulting command that is generated and will be passed to the operating system to start the application. This is the result of the configuration that has been defined. It is the chosen application followed by the command used by that application to open the specified file. Clicking the **View** (or **Edit**) button will execute the command.


As noted above, the Edit Document area is similar to the View Document area. At the bottom of the window are OK (apply and close), Apply, Restore to previously applied values, and Cancel buttons.

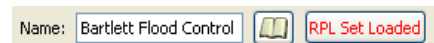
Once applied, the configuration settings are propagated to the default value of contained RPL Objects. Thus, the user only needs to specify as much information as necessary to fully configure the document. If the user is using the layout [HERE \(One document that describes the entire set\)](#), he/she only needs to

configure the RPL set and all rules, groups and methods will be configured correctly. If the user is using the layout [HERE \(One document for each RPL object in the set.\)](#), he/she needs to make sure that each RPL object is configured correctly but using a consistent naming convention can be used to simplify the configuration. If there are exceptions to the naming convention, the user would need to configure the RPL objects that have the exception.


## 5.5 Viewing and Editing

This section first describes viewing and editing documents in general, then describes peculiarities and specifics for each type of supported document. Once configured, viewing can be done directly from the RPL Object Editor using one of the following:

- Click on the external documentation icon . It is on the top of the dialog by the object name. Note, this icon is only shown if a document actually exists at the configured file path (including the default path if that is a correct configuration). Note, if the path leads to a web address, i.e. http://, the icon is always shown. RiverWare does not attempt to verify the existence of such pages on the web.
- Use the **View** ➤ **External Documentation** ➤ **View Document...** menu.

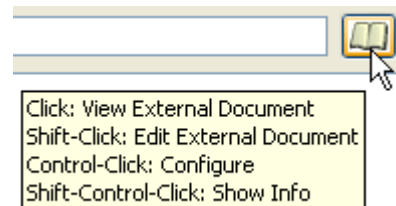



Similarly, editing can be done directly from the RPL object using one of the following:

- Shift-Click on the external documentation icon .
- Use the **View** ➤ **External Documentation** ➤ **Edit Document...** menu.

**Note:** Note, mousing over the External Documentation icon provides tool tips on the use of this button as follows:

Click: View External Document  
Shift-Click: Edit External Document  
Control-Click : Configure Document  
Shift-Control-Click: Show Info



Although Anchors (described [HERE \(Anchors\)](#)) are only supported when viewing HTML documents, the actual name of the RPL Object is copied to the system clipboard. This can then be used to paste into the “Search” function within the editor or viewer program. For example, if viewing a PDF, the user can click the external documentation icon  for a function, then click Ctrl+F to bring up the search field/dialog, then type Ctrl+V to paste the name of the function in the box.

The following sections describe the specific application recommended (and not recommended) to view and edit each type of document and then the mechanism to configure the application that is associated with a document type.

- Hyper Text Markup Language (.html)
- MS Word (.doc or .docx)
- Portable Document Format (.pdf)

- Text (.txt or other)

### 5.5.1 HTML

Hyper Text Markup Language (HTML) is a common format used by web pages. There are numerous tools to develop and view these types of files.

**Viewing:** HTML can be viewed using any common browser.

**Editing:** HTML can be edited using web development tools such as Adobe (Macromedia) Dreamweaver. MS Word can also be used edit HTML files.

**Anchors:** When a document describes more than one object and the user requests to view the documentation for a particular object, the program, if configured correctly, can open to the most relevant location within the document. In order to accomplish this, there needs to be some sort of connection between locations within the document and objects in the set. In this document we will refer to these as **anchors**. In HTML, locations within a document are defined using named anchors.

RiverWare uses anchors to provide a mechanism for navigating to a particular location within a document. For a given object that inherits its document location from a parent object, by default RiverWare assumes the existences of an anchor for this object whose text is based on the object's name. In particular, we assume that the anchor's label is the object's name, with illegal characters replaced with legal ones. The user may override the default value and edit it or specify that no anchor exists.

Typically, the user is responsible for inserting anchors into the external documentation (see [HERE \(Generating HTML Template File\)](#) for a tool to help create a document with anchors). RiverWare doesn't attempt to inspect the documentation, so care will be required on the part of the user to maintain anchors correctly. As with other sorts of references from an external document to the details of a ruleset, the potential for inconsistencies is large. For example, if the user changes the name of a function in a ruleset, then the name of the anchor for that function in the documentation would need to be changed as well. In addition, access programs will often not normally make anchor text visible, making it more challenging for the user to verify that the anchor labels are correct. To support the management of anchors, RiverWare supports copy/paste between fields containing object names and external programs.

For example, if the external document points to the following HTML file:

`http://www.WaterU.edu/Models/Functions.html`

and a function named "Set Outputs" is given the default anchor "SetOutputs". RiverWare will then provide the following to the access program:

`http://www.WaterU.edu/Models/Functions.html#SetOutputs`

**Generating HTML Template File:** The External Documentation feature provides a utility to generate a template of the RPL objects in the set. This can be used as a starting point in which the user can fill in the missing pieces. It allows someone not too familiar with HTML to quickly generate a working document and all the associated anchors and formatting.

From the RPL set's **Configure External Document Reference**, the user can select **Utilities** ➔ **Generating HTML Template File** to generate an HTML file that contains all of the RPL objects in the set. The file is created in the directory specified in the View Document configuration. It is given the name specified in the File field if it is an HTML file or the default name with a "-DocTemplate.html" appended if the File field contains a non-HTML file.

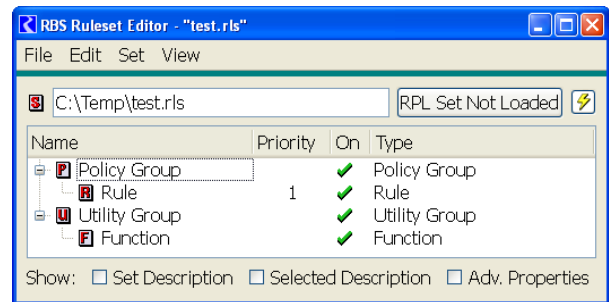
The HTML document also contains a table of contents that lists the type of RPL object, the name of that object (and a link to the referenced section) and the anchor that it uses. Shown in the body of the file is the RPL object's name and any description (i.e. in the **View** ➔ **Show Description** in RPL editor) that existed in the RPL object at the time the template was created. Once the template is created, it can be viewed/edited using the view/edit button. The user can then fill in pieces as necessary and save the resulting file. Then there should be no need to re-generate this template again.

An example from a one group with one rule and one function is shown. The text came from the description field in each object.

### 5.5.2 MS WORD

MS WORD (.doc or .docx) files are commonly developed using MS Word but more recently can be viewed (and even edited) using many browsers

**Viewing:** Traditionally, .doc files could only be opened using MS Word. Now, these can be viewed with certain browsers including Internet Explorer. Note, choosing to view a .doc file with MS Word and Explorer do not automatically make the document non-editable. The file system must be used if the user wishes to make the document read-only.



Template: C:\Temp\test-DocTemplate.html  
RPL Set: C:\Temp\test.rls  
Generated: 12:29 August 27, 2010

Type	Name	Anchor
RPL Set	<a href="#">test</a>	test
Policy Group	<a href="#">Policy Group1</a>	PolicyGroup1
Rule	<a href="#">Rule1</a>	Rule1
RPL Group	<a href="#">Utility Group1</a>	UtilityGroup1
User-defined Function	<a href="#">Function1</a>	Function1

#### RPL Set: test

Full name: C:\Temp\test.rls

This is the RPL Set Description

#### Policy Group: Policy Group1

Policy Group1 is used for ...

#### Rule: Rule1

Rule 1 does ...

#### RPL Group: Utility Group1

Utility Group1 contains functions for ...

#### User-defined Function: Function1

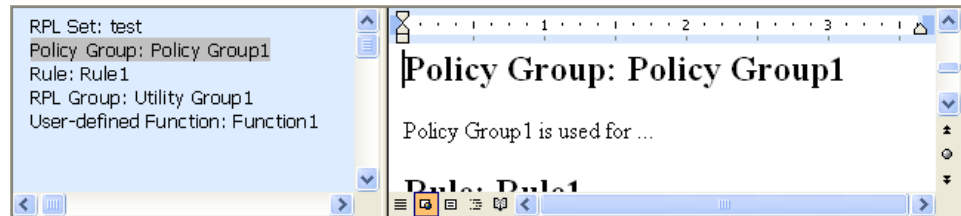
Function1 has the following arguments... It does ...

---

**Editing:** Again, MS Word is typically used to edit .doc files. The latest versions of Internet Explorer also has capabilities to edit these types of files.

For both editing and viewing Word and HTML files, in MS Word has feature called the Document Map that is useful to navigate through documents. Similar to bookmarks in a PDF file, a pane is added to the document that allows the user to navigate to defined headings in the document.

To access this feature, use the **View ➔ Document Map** in MS Word. A sample is shown to the right for the HTML document shown [HERE \(Generating HTML Template File\)](#).



### 5.5.3 PDF

Adobe's Portable Document Format (PDF) has become the standard document format when sharing read-only versions of documents. These documents can be viewed by the free Adobe Acrobat Reader and more recently by many browsers.

**Viewing:** Viewing of PDF files is accomplished using Adobe Acrobat Reader. This is the same format and program that RiverWare's Help file uses. As a result, RiverWare knows where Acrobat Reader is using the ACROREAD\_PATH environment variable. Thus, the user cannot change the location of the application for viewing PDF's.

**Editing:** Although limited editing of PDF can be done using the full version of Adobe Acrobat, this is not a supported editing tool. Otherwise, PDFs are read-only and cannot be edited. They are typically created from some other document editing application like MS Word or Adobe FrameMaker. We will not discuss editing PDF further.

### 5.5.4 Text

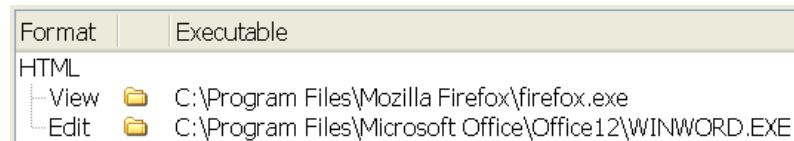
Text files can be viewed (and edited) by any text editor and many web browsers. No further information is provided.

## 5.6 Use Example

Following is an imaginary use example to show the process. Perhaps, we are a water agency and have a complex ruleset. We wish to share this ruleset and documentation with stakeholders in the basin but we do not wish to let them have an easily editable copy of this document. Also, we already have some descriptions entered in the ruleset but no other documentation. In this example, we wish to use our favorite web browser to view the document and MS Word to edit the document. Once the document is complete, we will post it to a web page so that stakeholders can see it. As a result, we will use HTML as the type of document.

We set this up as follows:

1. Define the file associations: We will use FireFox to view the document and MS Word to edit the document. We set up the File Type Association Manager [HERE \(Section 5.2\)](#) as shown:



Because we are using HTML, we can use anchors, [HERE \(Anchors\)](#), to automatically open the HTML file to the right place. As a result, we only need one document for the entire RPL set. Thus we are using the structure described [HERE \(One document that describes the entire set\)](#). We do want to have separate documents for viewing and editing. While making changes to the ruleset and hence the “Edit” document, we will still have an official “View” document that stakeholders can use. A diagram of this structure is shown [HERE \(Separate View and Edit Documents\)](#). Hence, we will need to configure only the top level of the RPL set.

2. From the RPL set, we open the configuration menu using the **View** ➔ **External Documentation** ➔ **Configure** menu.

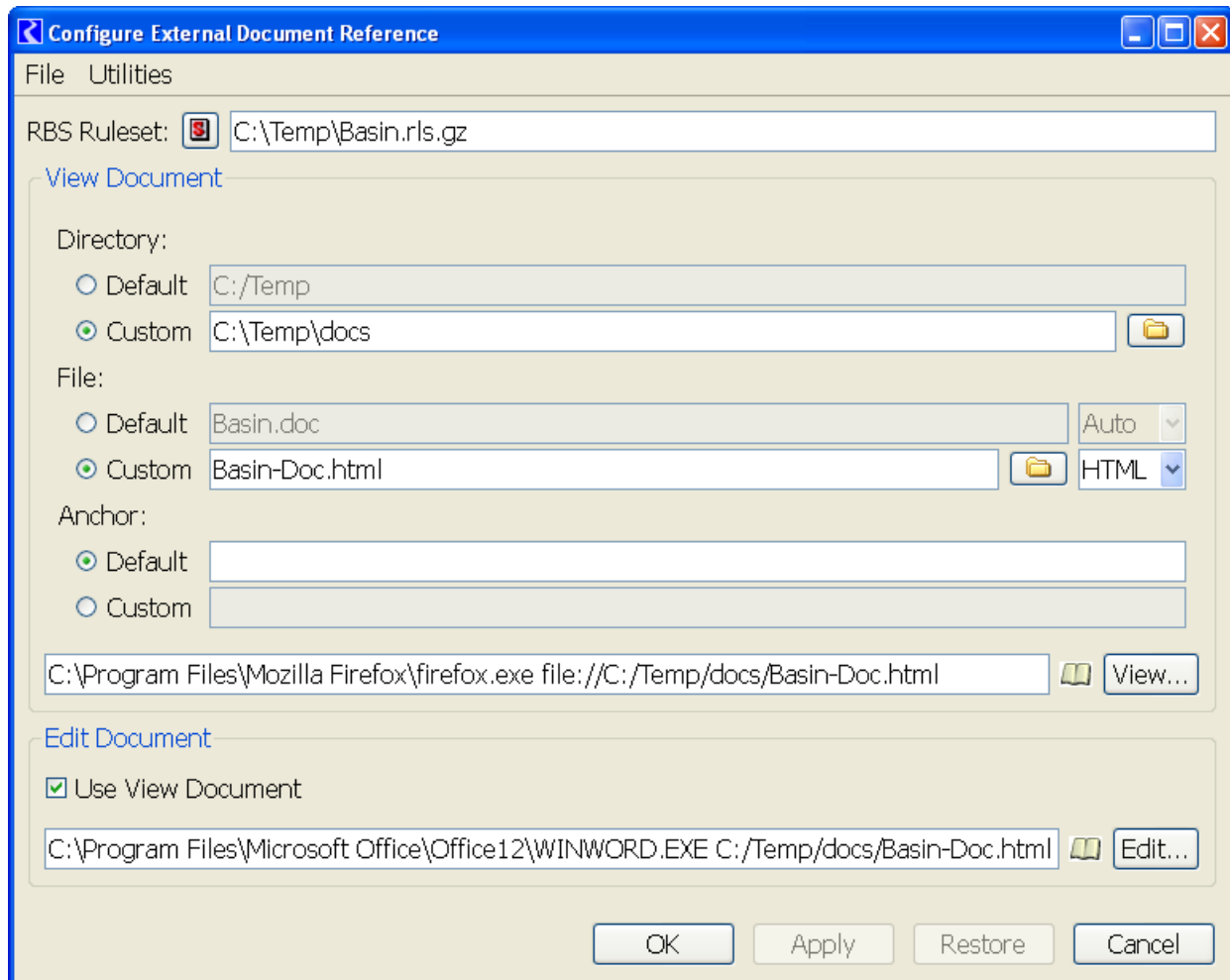
Because we already have some descriptions in our RPL set and we wish to use HTML, we will use the automatically generated template as a starting point. First lets specify a location to put the file, say C:\Temp\docs

3. In the View area, click the **Custom** toggle for the Directory and enter C:\Temp\docs

Now we will generate the template:

4. In the Configure dialog we choose **Utilities** ➔ **Generating HTML Template File**. It gives us a warning that a file named C:/Temp/docs/Basin-DocTemplate.html will be saved. Note our RPL set is named Basin.rls.gz.
5. We do not like the name of this file (and to avoid overwriting our completed external document with an inadvertently generated template file in the future), we change both the file name and the name in the configuration. We change the file to Basin-Doc.html using windows explorer. In the File area of the configuration, we change the name to Basin-Doc.html and click **Apply** when complete.

For now we will leave the View document in the C:\Temp\docs folder. Once we are complete we will move this to a web server. Our configuration is shown in the following screenshot.




6. Now we click on the **View** button and see the template file in Firefox.

We see that we have a pseudo table of contents with links, then each object is listed with the text that came from the description.

7. We click the **Edit** button and see the file open in Word.

8. We edit the document in Word by adding text, pictures, graphs, flowcharts, etc. We make sure to save it as the same name. We can edit the formats of the headings and text. When editing the headings, we make sure to use the **Format** ➔ **Styles and Formatting** menu in Word to change all instances at once.

9. Anytime we wish to see how this document looks in the browser. We can save the file from Word and switch back to the RPL set and click on the external documentation icon . This takes us directly to that RPL Object's section in the document.

10. When editing is complete, we copy the final HTML to a web server.

11. Then we go back to the configuration menu for the RPL Set and change the View document. First we want to click that the Edit document is now different, so we toggle off the **Use View Document** option. We make sure the Edit document area is now correct.

12. Now we want to change the View area to point the Custom Directory to a web server:  
<http://cadswes2.colorado.edu/docs/>

The configuration for this setup is shown in the following figure.

If we click on the external documentation icon for any RPL object, it will take us to this web page and automatically scroll to the section related to that object. We now share our model and ruleset with our stakeholders.

