RPL Global User-defined Functions

Authors: Patrick Lynn

This document outlines requirements, high-level design, and development issues for a new capability in RiverWare to provide RPL user-defined functions which can be shared between RPL sets.

1.0 Motivation

Currently when two RPL sets require the same user-defined function, that function must be defined in both sets. There are several unsatisfactory aspects of this situation:

- In addition to creating the function within one set, the user must copy the function to each other set in which it is needed.
- If a change is made to one version of the function, all other definitions of that function must also be found and changed if their behavior is to remain identical.
- When a user is viewing and modifying a function which also appears in other sets, there is no indication that this is the case, making it more likely that the function definitions with diverge with time.

To improve this situation, we will introduce support for global functions, i.e., user-defined functions which have a single definition but which may be called from more than one set.

2.0 Requirements

- Any RPL set containing only utility groups (i.e., no policy or category groups) can be designated by the user as a global functions RPL set. Suggested design: RPL set menu Set -> Functions Globally Accessible (toggle).
- When a set contains a policy or category group, the option to designate the set as a global functions set will be disabled. Similarly, for global functions sets, the actions to add a policy or category group will be disabled.
- This is a reversible designation, that is, the user can untoggle this property at any time (though we anticipate that once this property is set the user isunlikely to change it back).
- The global functions set property will be persistent, that is, it will be saved in the RPL set file.
- Functions in a global functions set will be global in scope, that is they can be called from any RPL expression (including from within the bodies of functions within a global functions set).
- Since global functions have global scope, their names must be unique, i.e, no other function in any open set can have the same name as a global function. During validation of any set, global function name conflicts will be flagged as errors, invalidating all sets.
- Global functions set editors will be decorated in some way to make this property readily apparent.
- Functions can be added to a global functions set in the usual ways:
    - •Creation: select a utility group, select Set->Add Function, then open the function editor for the new set to define it.
    - •Drag and drop: drag a function icon from one set and drop it into the destination utility group. Note that this removes the function from the original set.
    - •Import: importing items from a file, initiated by selecting File->Import Set.

- Copy/paste: select the original function, select Edit->copy, select the destination utility group in the Global Functions Set, select Edit->Append. The original function will then need to be renamed or deleted to avoid conflicting with the global function.

- All global functions (i.e., all functions in currently opened global functions sets) will be displayed in the RPL palette User-Defined Functions tab (the other tabs are “Palette Buttons” and “Predefined Functions”). We will consider using a distinctive font or label to indicate their distinction from functions local to the set currently associated with the palette (controlled by the current global RPL selection).
- All RPL set editors will optionally display the utility groups from global functions sets, just as the predefined groups are optionally displayed there. This selection will be made via the menu item: View -> Show Global Functions, and this will be selected by default (in contrast to the Show Predefined Functions toggle). To be consistent with the other viewing options, this setting will be applied to all sets and saved as a user preference.
- Like other types of functions, the global function icon will be the letter F, but the interior color will be distinct from those used for ordinary user-defined (orange) and predefined functions (turquoise).
- The function editor for a global function can be accessed in the usual way, typically by double-clicking on the function icon in a RPL set editor. Note that this introduces a new and potentially dangerous possibility: opening a function editor for one set (the global functions set in which a global function is defined) from the set editor of another set.
- An expression which contains a call to a global function is invalid if:
  - A function of that name is not currently in scope, i.e., no such global function exists.
  - The global function is invalid
  - There is more than one function with that name in scope (e.g., a function with that name exists in one global functions set as well as in the current set or another global functions set).
- A new command (OpenRPLSet) will be added to RCL which will open a RPL set. This command will not have the side-effect of toggling on the global functions property of the set, but if the opened set does not have this property set a warning will be issued, because in this context opening but not loading a set would have no effect. If RCL is being executed during an interactive run, then this warning should be skipped.
- If a user saves changes to a global functions set, we will present them with a dialog reminding them that changes to this set can afffect multiple policies. This dialog will contain an option to never display this warning again, which will be saved as a user preference.
- (Optional development) We will improve the tracking of edits to RPL sets so that when a user attempts to close a RPL set that contains any unsaved changes, we will present the user with a confirmation dialog. If no unsaved changes have been made, we will not present this dialog. This dialog will contain an option to never display this warning again, which will be saved as a user preference.
- (Optional development) We will create a dialog in which the users can interactively create, save, and execute RCL scripts. Interactive execution of RCL scripts would streamline the typical simulation sequence: load model, open global functions sets, load ruleset, run.

3.0 Discussion of design alternatives

From the user’s point of view, the distinction between a global function and a standard user-defined function is primarily the scope of the function, so it is natural that in most contexts global functions will be treated within the interface similarly to user-defined functions. For example, they should organized into utility groups and edited with the standard function editor. However there are a several important differences which require special consideration, and in this section we present our analyses of these issues, which is reflected in the requirements presented above.

3.1 Where should global function groups be presented?

Since global functions are accessible from all sets, groups of global functions should probably be displayed in the set editor for all sets. In addition, it seems useful to present them as the members of a special global functions set or sets.

3.2 Where are the global functions saved?

It is clear that global functions should not be saved with the policies which use them, because that would introduce multiple definitions which could potentially diverge with time (which is the problem we are trying to address). However, should we save the global functions with the model file or in a separate ruleset file? Saving the global functions with the model would provide consistency with the treatment of other special RPL sets (the expression slot, UDAM, and iterative MRM sets) as well as being convenient for many applications.

On the other hand, it would reintroduce the problem of divergent function definitions. If two models used the same global functions, then each model would have its own copy of the shared functions, and the user would have no good way to guarantee the consistency of the functions. This seems like a compelling reason for saving the global functions in separate RPL set files distinct from the model file.

Another reason for keeping the global functions separate from the model is that when function definitions are called as part of a policy, they are properly part of that policy, and RiverWare has maintained on principle a separation between the policy and the model. On the other hand, relevant to this point is the fact that global functions could be used by UDAMs and expressions slots and so need not be considered part of the policy. In addition, there is actually a precedent for saving policy with the model because this is an option for users of the optimization system. We could take a similar approach with the global functions, allowing the user to optionally save them within the model file, and letting the user assume the responsibility for maintaining the integrity of the global function collection.

3.3 How is a save initiated?

Assuming that the global functions are saved in a separate file from the model, then saving the model should probably not also save the global functions (this would require another file chooser, and furthermore the two actions are conceptually independent). So how should the user initiate saving changes to the global functions and specify the file to which they should be saved? If there were a single “special” set associated with the workspace, we could require the user to export it to a file to save changes to it, and import RPL set files into this set to add them to the current set of global functions, but that seems cumbersome.

I suggest that, for the purposes of opening and saving, we treat sets of global functions like any other set. That is the user can open a set from the Workspace Policy menu and save it back to the same set or another one via the File menu of any of the RPL editors associated with the set.

Probably it would be simplest to designate that a set is a global functions set when it is created or soon after. RiverWare would then only allow utility groups to be added to those sets, and interpret the functions in these groups as global. The global functions in all open sets would be accessible in all sets. We should present the global functions set editors distinctively using color and decoration, they sould not be loadable, and we should perhaps replace the “Loaded” button with an indication of the global status.

If there is a name conflict when a set is opened, then an error message would be issued and we might need to change the global status of that set.

3.4 Remembering to save the global functions sets.

The fact that global functions are visible and accessible from within the editors of all sets creates the possibility that the user might make a change to a global function, save the set from which it was accessed, and not realize that the changes to the global function have not in fact been saved. This motivates work to improve our tracking of user changes so that we can present them with a confirmation dialog in this situation.

Similarly, the fact that the global functions are global means that changes to a global function can affect the behavior of any other set with which the global functions coexist, including ones that are not even currently open or loaded. One can imagine a user opening a global functions set that is used with multiple policies, loading a single policy, deciding to change a global function, and not realizing that by saving their changes they are affecting every other policy that uses that global function.

3.5 Associating global functions sets with sets that call them

We could consider additional enhancements which would allow the automatic opening of global functions sets as necessary. Once a RPL set contains a call to a global functions set, then we could save that dependency within the ruleset, and try to open that set as well when it is opened. This would probably require some user involvement, perhaps designating directories in which to look for global functions sets or toggling whether or not this automatic opening is done. At a minimum we could open a dialog which asks them if they would like us to open a global functions set.

4.0 Software design notes

When a function call expression is validated (usually as a distinct step prior to execution), the name of the function being called is looked up in the appropriate namespace (class RplNameSpace). The namespace contains bindings of names to objects. Importantly, one RplNameSpace can be added to as a subspace; all names in the subspace are included in the full namespace. Subspaces are represented by reference; so that if a name is added to a subspace it is automatically included in the parent namespace. At validation time the function bindings are saved and used in subsequent evaluations.

Thus to make global functions have global scope, they need to be included in the namespaces used for the validation of all expressions. This can be accomplished by including the global functions in every individual RplSet's associated namespace (returned by RplSet::getNameSpace()). This namespace is created in RplSet::populateNameSpace() which mainly iterates through the set's utility groups, adding each utility group's namespace as a subspace of the set. To also add the globabl functions, we simply need to additionally loop over the utility groups containing global functions (i.e., the groups in all the global functions sets), adding their associated namespaces as subspaces. Note that it might seem simpler to just add the namespace of each global functions set as a subspace, but their namespaces also include all global functions sets, which would lead to redundancy if not circularity.

To make the global functions show up in the RPL palette, we need to modify the method: RplPaletteDlg::updateUserFunctions(). One might imagine that this method could just get the namespace from the selection, but in fact it finds the set associated with the current selection and iterates over the utility groups in that set, adding their functions to the palette's list (as well as the functions in the selection's group if it is a policy group). Thus we will need to add another loop over the global functions groups here as well, adding their functions.

5.0 Related screen snapshots

Figure 1. The Workspace Policy Menu.

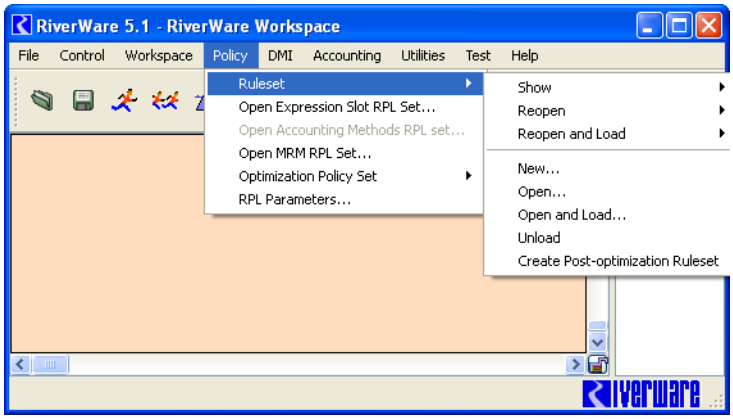


Figure 2. The RPL Set Editor.

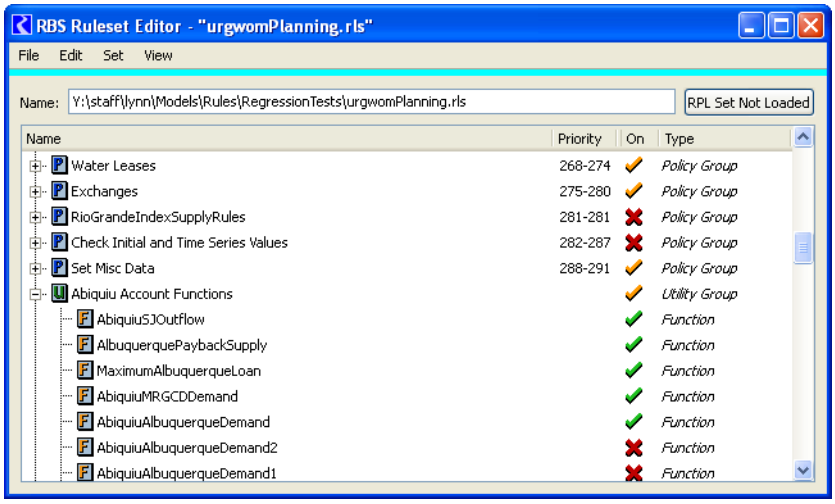
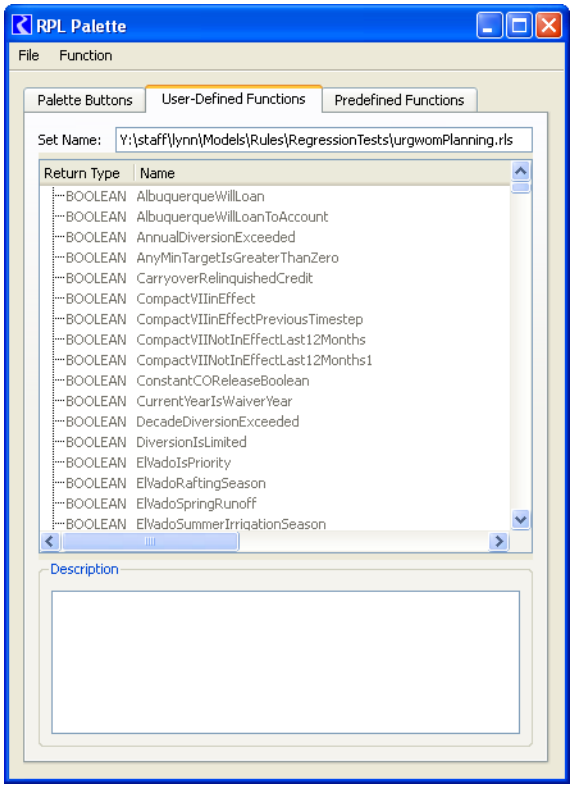


Figure 3. The RPL Palette.



6.0 Development Schedule

Days	Task
3	Primary GUI work (new toggle, logic for enabling controls, new icon, global functions set decoration)
1	Persistence of “global functions” set property (RPL language, RplSet save, RplSet data member)
4	RPL internal use of global functions (adding subspaces to namespaces as appriate, possibly callbacks)
2	Validation complications
1	Supporting changes to RplSetMgr and RplDlgMgr
1	Detailed design support (Patrick)
12	**Total**
2	Better change management (largely done as part of undo/redo, using QCommandStack)
?	Interactive RCL editor