RiverWare Link to External RPL Documentation

Authors: Patrick Lynn and Phil Weinstein, CU-CADSWES

This document outlines requirements, high-level design, and development issues for a new capability in RiverWare to provide convenient access to external documentation for a RPL set.

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 a set, group, function, or rule. Note that currently RPL object editors support an optional multiple-line text description, displayed in a plain-text editor box. The document support described in this document is distinct from this description text value (which is stored in the RPL set file).

1.0 Basic Requirements

In a typical scenario, as a modeler creates a RBS model and ruleset, she also uses Microsoft Word to create documentation describing the ruleset, its high-level organization, rules, and functions. The modeler then provides the model, ruleset, and documentation to anyone else that wishes to explore the model.

In order to better support scenarios such as this, we propose to add the following functionality to RiverWare:

- From a RPL dialog, the user should be able to easily access documentation for the RPL object associated with that dialog.
- Two modes of access will be 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. This is the only type of access provided from within the RiverWare Viewer.
- RiverWare will provide access to the documentation by executing a program appropriate for the documentation type (format) and the mode of access.
- For Edit mode, RiverWare will support the use of Microsoft Word for editing documents in the Word format.
- For View mode, RiverWare will support at least one of the following formats: Word, PDF, HTTP.
- RiverWare will be flexible with respect to the granularity of the documentation. 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 the access program executes, the document associated with the current RPL object is automatically opened.
- When possible, the program should 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.
    - •The access program supports external control of where it opens.
    - •The user has (somehow) indicated within the documentation which sections pertain to which RPL objects.
- If the user has already opened the access program for the desired mode, then the program should be made visible (e.g., de-iconified or brought to the foreground, as necessary).

2.0 Detailed Requirements

In this section we elaborate on the basic requirements, describing how RiverWare will collect, save, and use the information required to provide access to external documentation. The content of this section is guided by the observation that when the user launches an access program, RiverWare needs to know the following:

- Whether the user would like to view or edit the external documentation
- The location of the document
- The access program in which to open the document
- How to control the access program

2.1 Supported RPL Objects

Users will be able to associate documentation with RPL objects whose type is one of the following:

- Set
- Group (Utility, Policy and Category)
- User-defined Function (not predefined or TCL functions)
- Rule / Method / Optimization Rule

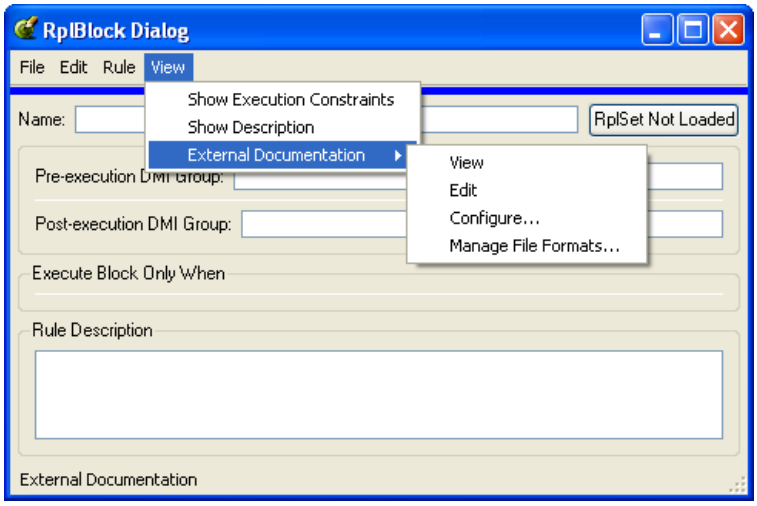
When the external documentation for a set consists of multiple files, we assume that the organization of these files mirrors the organization of the set itself. For example, if the user requests documentation for a rule but there is no document associated specifically with that rule, we present documentation associated with the parent group if any exists. If no documentation is associated with the rule's group, we present the documentation associated with the rule's set.

2.2 Changes to the RPL Editor Dialogs

For each type of RPL editor we will add a cascading menu entitled "External Documentation" to the main View menu. This new menu will contain the following options:

- View -- Launch the access program in View mode for the current object's documentation.
- Edit -- Launch the access program in Edit mode for the current object's documentation.
- Configure -- Allow the user can provide information about the external documentation associated with the current object (e.g., the document location and format). This dialog is discussed in the following section.
- Edit File Types -- Allow the user to specify which program should be used to access files in supported formats. This dialog is discussed in a later section.

Figure 1. Illustration of th new RPL editor cascading menu for external documentation.



When no external documentation has been linked to an object**,** the edit option will be disabled. In the RiverWare Viewer the edit option will be disabled and in addition the configuration options will be limited as appropriate.

Once an object has been configured for external documentation, RiverWare will provide easy access to the external documentation by displaying a new “View External Documentation” button in the RPL editor dialog. Clicking this button will launch the access program in View mode. This button will be hidden when there is no external documentation associated with the current object.

The following issues still need to be resolved:

- The “View External Documentation” button
  - •Location: possibly between the object name and the “RPL set loaded/not loaded” button.
  - •Button icon/label: The button could be the same as the on-line help button (a yellow question mark). This should be the preference if we intend to eventually support access to online help for predefined functions in a way that is consistent with access to external documentation of user-defined objects. I would suggest improving the icon by outlining the question mark in black, the yellow is often close to invisible. Alternatively we could create a unique icon, perhaps a “page of text icon,” or reuse some other RiverWare icon that is used in a similar way.
- Should we provide easy access for the Edit mode as well? For example, we might present another button or distinguish between different ways of clicking a single button (right click versus left click).
- Should we provide access to the external documentation actions via a context sensitive popup menu? The various dialogs involved might be different enough from each other that it would be difficult to present a consistent interface.

2.3 External Documentation Configuration

The purpose of the modal configuration dialog is to collect information from the user about how a specific RPL object is linked to its external documentation. When an object has been configured, no additional (object-specific) information should be required to launch the access program for that object. RiverWare needs to know two things:

- Where to find the documentation for the current object
- In which progam to open the documentation

2.3.1 Determining the Document Location

External documentation for a single object might be found in two different files, one for editing and one for viewing. For example, documentation might be created in Microsoft Word, saved in Word format for the purposes of future edits, and also saved to a PDF file for the purpose of viewing. Thus we allow the user to specify two locations, one for the purposes of viewing and another for editing.

When the documentation for a ruleset is split across many files, then it will often be convenient to the users for all the files to be co-located in the same directory. Thus one goal of the current design is to support this sort of organization 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, we don't want 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.

We meet this need by separating the location specification into two pieces (for path names, into a directory and a file name) and by relying on the hierarchical organization of RPL sets to share information between objects. When configuration information is provided for any object, this information is inherited by the objects it contains as their default configuration. 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.

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*. These terms are widely understood and represent a typical case, but we will not require 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 we expect 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".

2.3.2 Document Anchors

When a document describes more than one object and a request is made to view the documentation for a particular object, we would like the access program to 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.

Some document formats support the concept of a text label for a location within a document, and each format uses different terminology, for example:

- Microsoft Word: bookmark
- PDF: named destination
- HTML: anchor

In this document we will refer to these as *anchors*; within the RiverWare user interface we will attempt to use the terminology appropriate to the relevant format.

RiverWare will use 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 we will assume existence of an anchor for this object whose text is based on the object's name. In paticular, 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.

The user will be responsible for inserting anchors into the external documentation. RiverWare has no way to inspect the documentation, so care will required on the part of the user to maintain anchors correctly. As will other sorts of references from an external document to the details of a ruleset, the potential for inconstistencies 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 will support copy/paste between fields containing object names and external programs.

Ideally, when an access program is asked to go to a particular anchor, it should behave gracefully if the anchor is not valid. If possible, RiverWare would describe the nature of the problem to the user, however this sort of error handling might not be supported by the access program.

Note that not all programs support opening a file at a particular location. For example, this behavior is not supported by Microsoft Word or by Adobe Reader 7.0, but it is by Reader 8.0 and HTML browsers. Programs that support this behavior, do so in different ways. Reader 8.0 supports this behavior as a certain command line switch whereas browsers support it by using the fragment portion of a URL as the anchor text.

Since the URL is a standard used in similar ways by many applications, and since many HTTP browsers now support many different file formats (Internet Explorer supports the display of HTML, PDF, and Word files, among others), we propose that RiverWare support the "open at an anchor within a document" behavior only for documents specified as a URL. If the URL does not already contain a fragment specification and an anchor has been specified, then the anchor will be appended as the fragment. For example, for the URL:

http://www.WaterU.edu/Models/Functions.htm

and the anchor "Set_Outputs", Riverware will provide the following argument to the access program:

http://www.WaterU.edu/Models/Functions.htm#Set_Outputs

Note that the form of a URL is: <scheme name> : <hierarchical parts> [? <query>] [# <fragment>]
In the context of an external document specification, we expect the query portion to be absent and the scheme name is likely to be one of the following: http, https, nfs, or file.

2.3.3 Determining the Access Program

We considered several options for determining in which program to open an external document. One possibility would be to associate programs with objects in a RPL set, just as we do document locations. However if a set is documented in HTML, we do not want to require that each user reconfigure the set so that they can use their favorite HTML browser program to access the documentation.

Thus it seems natural that the determination of access program should be based on "per user" settings. One option would be to have each user specify the program that they want to use for external documentation, and save this a a login-based preference. However, if one ruleset is documented in HTML and another as PDF files, we don't want to require that the user change this setting before switching between rulesets, so we will organize the user preferences around the type of file be accessed.

Specifically, we propose that RiverWare create and maintain program associations based on file format, and save these program associations as user preferences. For example, a particular user might specify that for viewing documents:

- PDF files be opened with Adobe Reader 8.0
- Word files with Microsoft Word
- HTML files with the Firefox browser

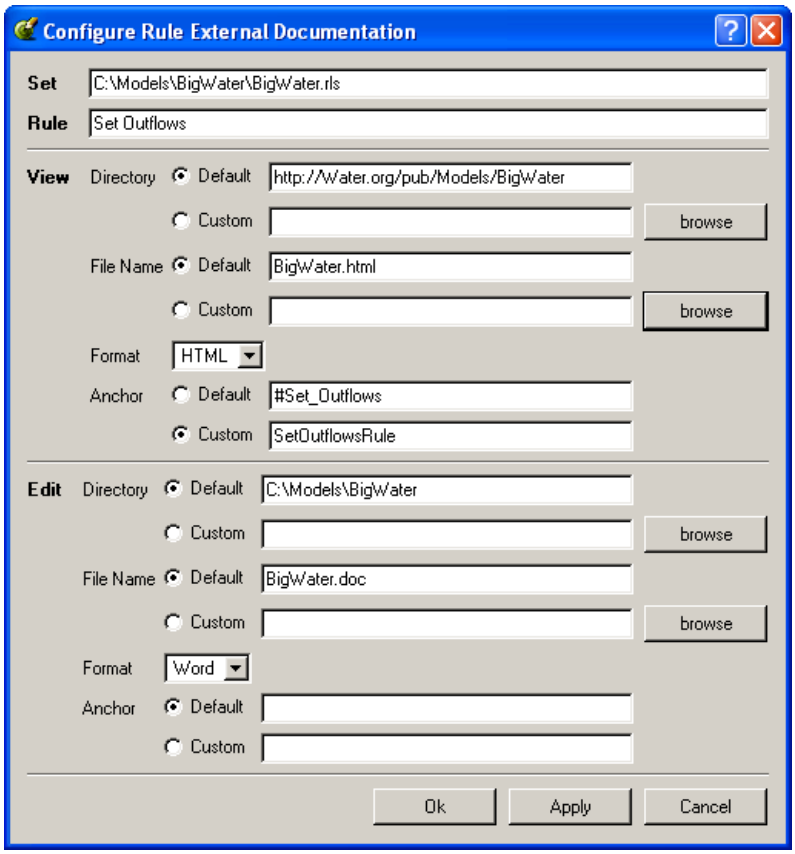
We discuss the details of how these program associations will be maintained in a later section, but for now we assume that we can reduce the problem of determining in which program a document should be opened to that of determining the document's format.

On the Windows platform, the operating system determines file type based on file name extensions. Windows provides a default set of associations between file name extensions and programs with which to open files with that extension, and allows users to change these default associations as well as create new ones for uncommon file name extensions. These associations are saved in the system registry and, as long as care is taken to name files with the appropriate extension, they provide a reliable mechanism for determining a file's type.

Solaris does not use a similar convention, but even on Windows we can not simply use the system's program associations because some users might want to view and edit the same document with different programs. For example, a user might prefer to view HTML documents with their favorite browser but edit them with a separate HTML editor. We propose that when possible RiverWare use file name extensions to make an initial guess at a file's type, but allow the user to provide specify this information when there is no file name extension or when the default association is not appropriate for a file and access mode.

2.3.4 The Configuration Interface

Figure 2. Mockup of the dialog for configuring the external documentation link for a RPL object.



In the figure above, a user asked to configure the external documentation for a rule named "Set Outflows". The set itself has already been linked to a Word document for editing and an HTML document for viewing, and consequently the configuration for this rule inherits the appropriate fields from the set configuration. The user has chose for the most part to accept these default values, but chose a different anchor naming convention for the HTML file and so has specified the value for that field.

The configuration dialog has the following features:

- The name of the object being configured and its ruleset (when the object is not a ruleset) are displayed as read-only text available for copy/paste (into other fields in the configuration dialog or external documents).
- For each access mode, the user describes the documentation for the current object by specifying the following information:
    - •Directory
    - •File Name
    - •File Type
    - •Anchor
- All fields have a default value which is displayed as read only text.
- The View mode default directory and file name values are based on the parent object's value for that field, when there is a parent object. A parent object's value may itself be a default value.
- For a set (which has no parent)
    - •The View mode default directory is the directory of the current ruleset
    - •The Edit mode default directory and file name are based on the View mode values for those fields.
- For any field the user can choose to override the default, in which case they are provided with the default value as a starting point for editing.
- A directory chooser button may be used to specify the directory; similarly we provide a file chooser for the user-specified file name field.
- Standard editing operations are supported for all user specified fields (Cut/Copy/Paste), including the use of the system clipboard. That is, support is provided for copying field text between this dialog and an external application.
- When the file name has a default value inherited from a parent object, the default file type is inherited from that object as well. When the user specifies the file name with an extension (alphanumeric characters following the rightmost period character), then the default file type is based on that file name extension. If a user-specified file name does not have an extension, its default file type is Word.
- The user may reference environment variables within the user-specified directory and file name fields; they will be expanded by the operating system before the location is passed on to the access program.
- When the default file name is inherited from a parent object, it assumed that the document applies to more than one object and a default anchor is constructed from the object name. Otherwise, the default anchor is the empty string.

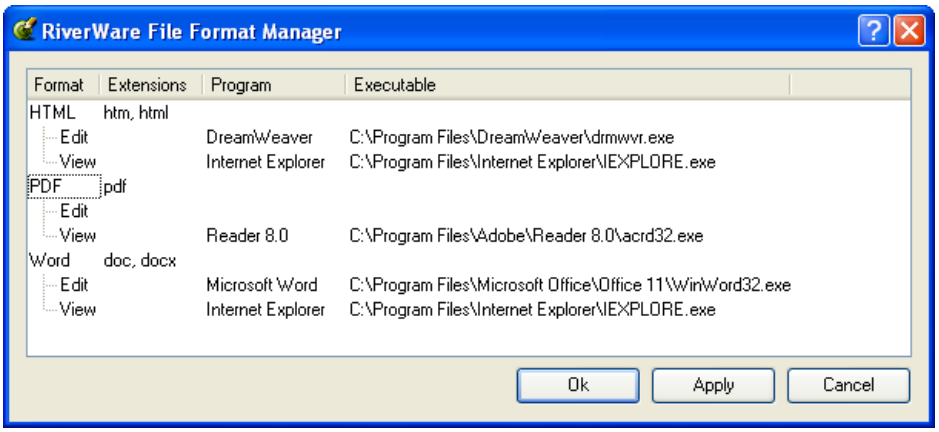
When a user launches a program to view or edit a RPL object's associated external documentation, the document's location is determined based on the object's configuration -- the full pathname is taken to be the concatenation of the directory with the file name. RiverWare inserts an appropriate character between the two strings as required by the system file path syntax. For example, if the directory is "C:\temp" and the file name is "MySet\FunA.pdf", the the full pathname is taken to be "C:\temp\MySet\FunA.pdf".

As discussed in the previous section, if the user has specified an anchor and the locations specification is determined to be a URL, then this anchor is appended to the URL specification as a URL fragment.

2.4 File Format Management

As discussed above, RiverWare needs to know in which program to open a particular document and this determination will be based on the format of the document. In a previous section we discussed how a document's format is determined; in this section we discuss how RiverWare determines the access program from the format. The basic approach is to maintain user preferences for programs with which to view and edit files in supported formats.

Figure 3. Mockup of the dialog for specifying file format preferences.



The File Format Manager dialog will allow the users to associate programs with file format / access mode combinations:

- It will be accessible from
  - •The RPL editor “View->External Documentation->File Format Manager” item
  - •The RPL object External Documentation Configuration dialog (by button?)
  - •The workspace ? menu
- For each file format we distinguish between the Edit and View mode program associations.
- For each combination of file format and access mode, we associate the following fields:
  - •Format name
  - •Extensions
  - •Program
  - •Executable
- Example: HTML / htm, html / Firefox / C:/Program Files/Mozilla/Firefox/firefox.exe
- Initially we will support the following formats: HTML, PDF, Word (Doc). In the future we could support the addition of other formats by the user.
- The users may edit the Name of Program and Path of Program fields and may add to the extensions.
- On Windows, the registry is used to provide defaults for the user editable field.
- A file chooser is available for setting the Path of Program field.
- The program associations will be stored in a login-based persistence mechanism, e.g. Qt’s QSettings.
- The current settings apply to all objects in all RPL sets.

2.4.1 The Access Program for RiverWare On-line Help

RiverWare already makes use of one program for access to external documentation: Adobe Reader for RiverWare’s on-line help (currently in the PDF format). The user is responsible for setting one or two environment variables:

- ACROREAD_PATH - the full path of the executable
- ACROREAD_OPTIONS - text which is provided on the command line (in addition to the name of the file to be opened).

In the long term, we do not want to have two different mechanisms within RiverWare for specifying a program to access a PDF file, especially if we support PDF as a format for the user's documentation. Because we have already used the environment variable mechanism for allowing the user to specify a program for RiverWare to execute, we propose to continue to support this method of specifying a PDF viewer. However, if no ACROREAD_PATH is specified, we will use the view program specified in the File Format Manager.

2.4.2 Controlling Access Programs

For RiverWare to manage the access programs with a high degree of control, for each access program RiverWare would we need to know how to control it in the following ways:

- Open: with a given document (specified as a file system path or a URL)
    - optionally, open at a given anchor within the document
- Raise: for an existing process, if iconified, deiconify; bring to forefront of the screen.
    - optionally, go to a given anchor within the document
- Exit

Because not all access programs will support all of these behaviors and for those that do it is not feasible for RiverWare to maintain program specific control protocols, we propose that RiverWare make only basic assumptions about the access programs. In particular, we will assume that the following command line form will open a document in an access program:

<program executable path> <document location specification>

For example:

C:\Program Files\Microsoft Office\Office11\WinWord.exe C:\Models\BigWater\BigWater.doc

Furthermore, we will assume that the access program is able to determine if it is already running and open as a separate process or open the new file in the existing executable consistent with the user's wishes. As with the on-line help reader program (currently Adobe Reader), the user is responsible for exiting the access program when they are through viewing or editing the documentation.

3.0 Documentation for RPL Predefined Functions

We should eventually provide access to the on-line help for RPL predefined functions in a way that is consistent with the support for user-created external documentation. Specifically, from the editor dialog for a RPL predefined function they should be able to launch a viewer for the associated on-line help. As with documentation for user-defined objects, the access program (Acroread currently) should open to the portion of on-line help which discusses the current function.

4.0 Development Estimates

In summary, we propose to add a RPL external documentation configuration dialog accessible from all of the RPL object editors. In this dialog the user specifies where to find the document associated with that object, what its format is, and where within that document the access program should open. Initially the format options wil be: Microsoft Word, PDF, or HTML. In a separate file format management dialog, the user will specify which program should be used to access documents of each format of concern to them. RiverWare supports a distinction between viewing versus editing an external document, allowing the user to provide different documents for the two access modes as well as use different programs to access a given format in the two modes.

Total Hrs	**Task**
13	RPL Editors GUI work (new menud, buttons)
14	RplObj: add external documentation configuration data, set/get methods (encapsulate in new class)
20	RplObj: support save/load of new configuration data
53	Configuration dialog (GUI): form creation, connections, cut/copy/paste, validation
13	File Format Manager: basic class
33	File Format Manager: dialog (GUI)
20	File Format Manager: initialization (on Windows, interaction with registry)
20	File Format Manager: persistence as user settings
14	Implement launch viewer/editor behavior (execute and manage processes)
27	Extend functionality to identify URLs, support anchors
40	System testing, refinement
40	Documentation
307	**Total**

Total hours: 307 hours

This estimate is to develop all the functionality described in this document.

Cost Estimate: $35,000