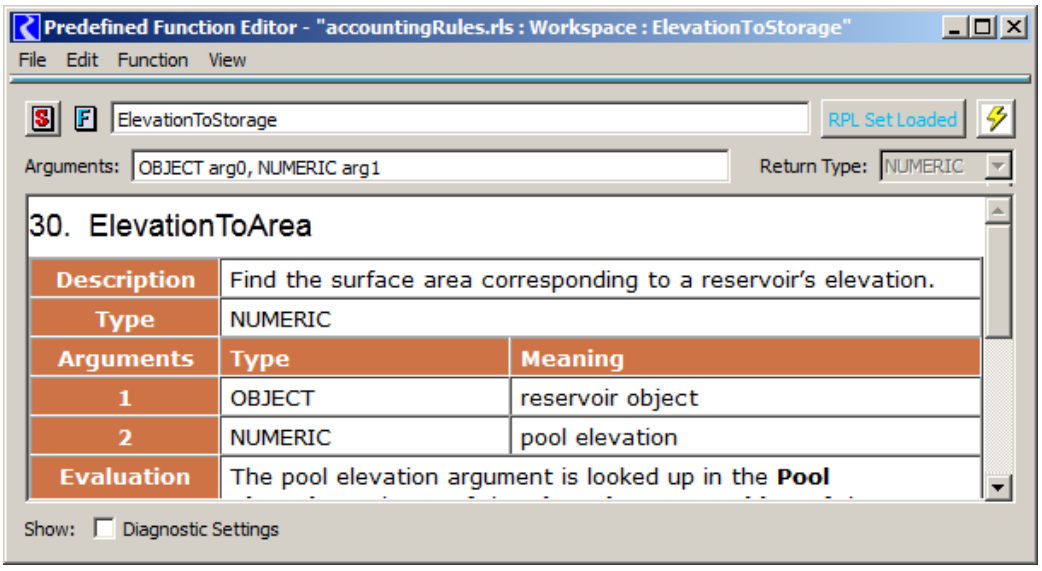
RiverWare: Improved Access to RPL Documentation / Analysis and Design

Phil Weinstein, CADSWES, 3-20-2014.

This document proposes enhancements to RiverWare (beyond version 6.4) to display online help content within RiverWare for RPL Predefined Functions and RPL language constructs (statements and operators).

Contents:



(1) Background

(1a) Task Description

This analysis and design document addresses this task:

> *Design and estimate improved access to RPL documentation.*
>
> CADSWES staff will design and estimate mechanisms for more convenient access to RPL documentation. For example, the Predefined Functions tab of the RPL palette should provide access to function documentation that is comparable to that contained in on-line help. This includes a description of the purpose of the function as well as meaningful names and descriptions for the function parameters. Similarly, the Predefined Function editor should provide convenient access to the same information. RPL editor dialogs should provide context-sensitive access to on-line help based on the current selection.

(1b) RiverWare Documentation Overview

RiverWare documentation is currently authored with Adobe FrameMaker 10. It consists of about 30 related documents with hyperlinks to specific sections across documents. A one-page top-level "menu" document provides access to the individual documents. This documentation is available from these places as PDF files:

1. Online, on the RiverWare website at: http://www.RiverWare.org/PDF/RiverWare/documentation/
2. From the RiverWare program, stored locally with the RiverWare installation, accessible from the RiverWare workspace's "Help" menu. This brings up the user's currently configured PDF reader, typically Adobe Reader.

An important feature of FrameMaker used in RiverWare documentation is the ability to compose mathematical expressions, displayed with vectored graphics in the generated PDFs. The process of generating the PDFs includes use of an external tool to implement hyperlinks between PDF documents.

Appendix A presents an outline of the types of information in the RPL components of RiverWare documentation. This represents six (6) of the thirty (30) individual RiverWare documents (including "Rulebased Simulation").

(1c) Help Content: RPL Predefined Functions

RPL Predefined Function documentation is provided in a single FrameMaker-sourced document. For RiverWare 6.4, it contains 191 functions. 77 of those contain mathematical formulas. One has an associated graph image.

- RPL Predefined Functions
  http://www.RiverWare.org/PDF/RiverWare/documentation/RPLPredefinedFunctions.pdf

In PDF, mathematical formulas look very good, e.g. when zooming. They are generated as vectored graphics in PDF files. However when generating HTML, only *raster* graphics files are generated (i.e. GIF or PNG or JPEG). The SVG scaled vector graphics format would be preferable, as that is currently supported in all modern browsers (including in Qt WebKit used in RiverWare).

Each RPL Predefined Function has its own section, with most of the information laid out in a table. Here is an example. Note: not all functions have an introductory sentence above the table. We are planning on moving that text to either the Comments section or under the table.

30. ElevationToArea

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

Description	Find the surface area corresponding to a reservoir's elevation.	
Type	NUMERIC	
Arguments		
1	OBJECT	reservoir object
2	NUMERIC	pool elevation
Evaluation	The pool elevation argument is looked up in the **Pool Elevation** column, of the **Elevation Area Table**, of the reservoir object argument, to determine the **Surface Area**. If the exact elevation is not in the table, the lookup performs a linear interpolation between the two nearest bounding elevations and their corresponding surface areas.	
Mathematical Expression	$area = area_{(lesser)} + \frac{area_{(greater)} - area_{(lesser)}}{elev_{(greater)} - elev_{(lesser)}}(elevation - elev_{(lesser)})$	
Comments	If the object is not a reservoir, or the reservoir does not have an **Elevation Area Table**, the function aborts the run with an error (**CRSSEvaporationCalc**, **DailyEvaporationCalc**, **PanAndIceEvaporation**, or **InputEvaporation** must be selected as the **Evaporation and Precipitation** Category selected Method. This function will issue an error if the "Time Varying Elevation Area" method, **HERE (Objects.pdf, Section 22.1.24.2)**, is selected. Instead, use the ElevationToAreaAtDate function described next.	

Syntax Example:

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

Return Example:

```
634547087.2 [m2]
```

(1d) Help Content: RPL Statements and Operators

RPL Statements and Operators represent most of the sixty (60) buttons on the first tab of the RPL Palette. (*See RPL Palette images in the next section*). They are described in the "Palette" portion of this online help document:

- RPL Data Types and Palette
  http://www.RiverWare.org/PDF/RiverWare/documentation/RPLTypesPalette.pdf

Help content for each of these items is presented as one row in a table of related items. Here is an example:

2.1 Mathematical Operation Buttons:

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

This is a good presentation for full-page media. But when displaying content for just one item (one row) in the user interface, we may prefer to rearrange the content. (This should be done automatically either in the preparation of content for RiverWare, or dynamically within RiverWare).

As a fundamental quality of the use of RPL Palette buttons, each corresponds to a class of selectable sub-expressions within a RPL block or expression shown in a RPL Frame. Clicking on a button replaces the selected sub-expression with a new instance of the button's RPL language construct.

(1e) Possible Future Document Architecture

In the future we may want to consider a more sophisticated structure and process for all RiverWare documentation which would support ideal presentations in these media:

1. Hard-copy printing / whole chapters or books, or individual items.
2. External viewer/browser.
3. RiverWare GUI / panels within existing application dialogs or separate windows.

Some of the qualities we will want from a modernized RiverWare documentation system are:

1. Single-source for all media: Print, External viewer (browser), RiverWare GUI panels and windows.
2. Variable width presentations with wrapped text. It should be possible to reasonably view documentation content within a narrow window or "panel". It should also look good in a printed page format.
3. Flexible granularity. It should be possible to present individual sections of the RiverWare documentation, e.g. down to the level of individual RPL Predefined Function *Arguments*. Also, references to relevant sections should be accessible programmatically from the RiverWare program.

Existing qualities and capabilities of our current document infrastructure which will continue to be important include:

4. Support for hyperlinks, contents and index generation.
5. Support for easy authoring of *mathematical formulas* and inclusion of externally generated images. (Ideally, when generating HTML/XML output, formula images should be generated using high-quality SVG *vectored* graphics files, rather than GIF or PNG or JPEG *raster* graphics).

We may want to eventually use a content management / publishing tool supporting the DITA ("Darwin Information Typing Architecture") standard, such as the *structured* mode of operation of FrameMaker. With this sort of system, we would probably choose to convey help content to RiverWare in XML files, and use XSLT to generate HTML inside RiverWare.

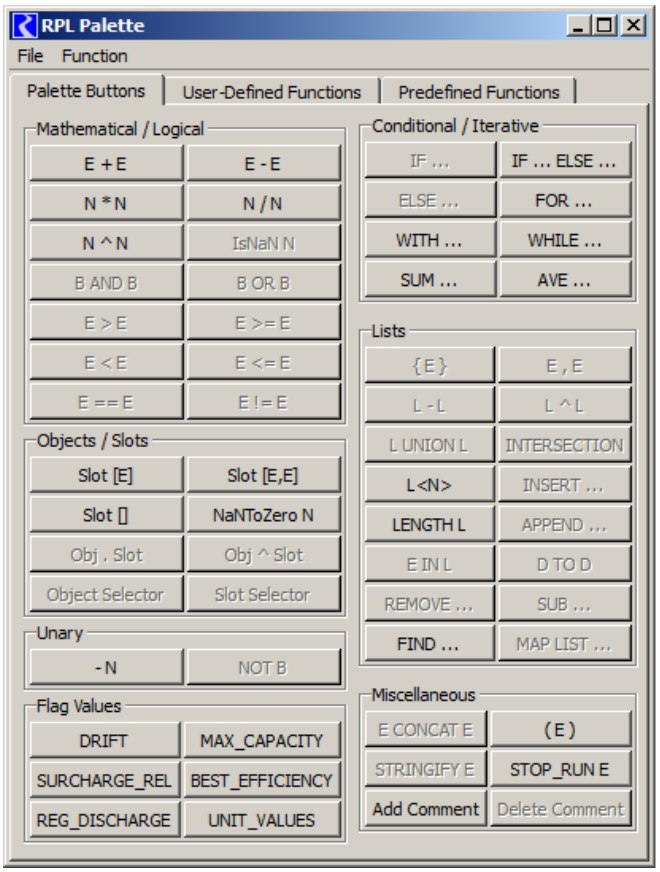
(2) Requirements Analysis

(2a) Feature Scope: What Help Topics? Where Shown?

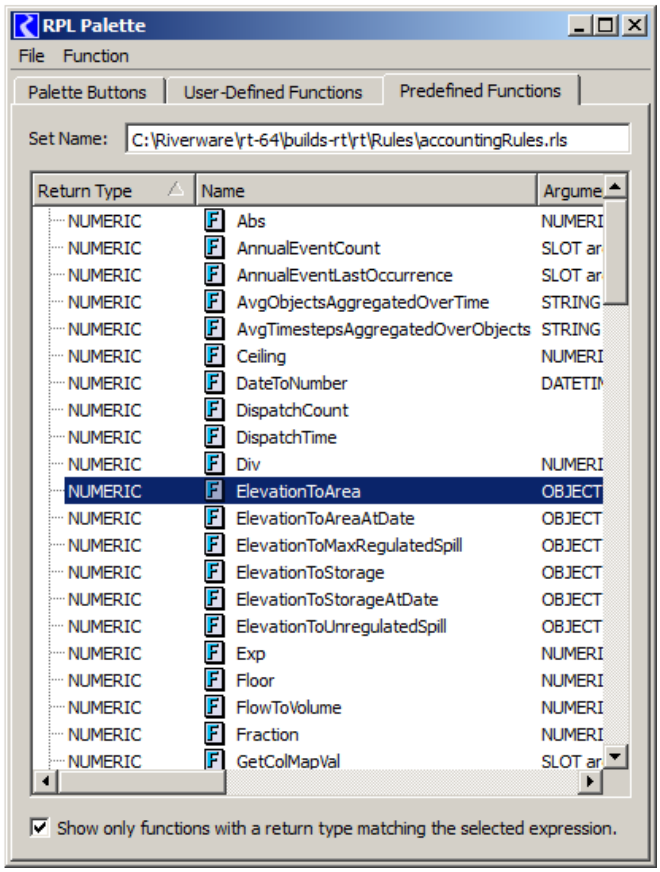
This proposal focuses on two of the "enumerated item" topics (among the various RPL documentation topics listed in Appendix A).

1. RPL Predefined Functions (191 functions).
2. RPL Statements and Operators / Buttons in the RPL Palette (60, in 8 categories).

RPL Palette Buttons



Predefined Functions

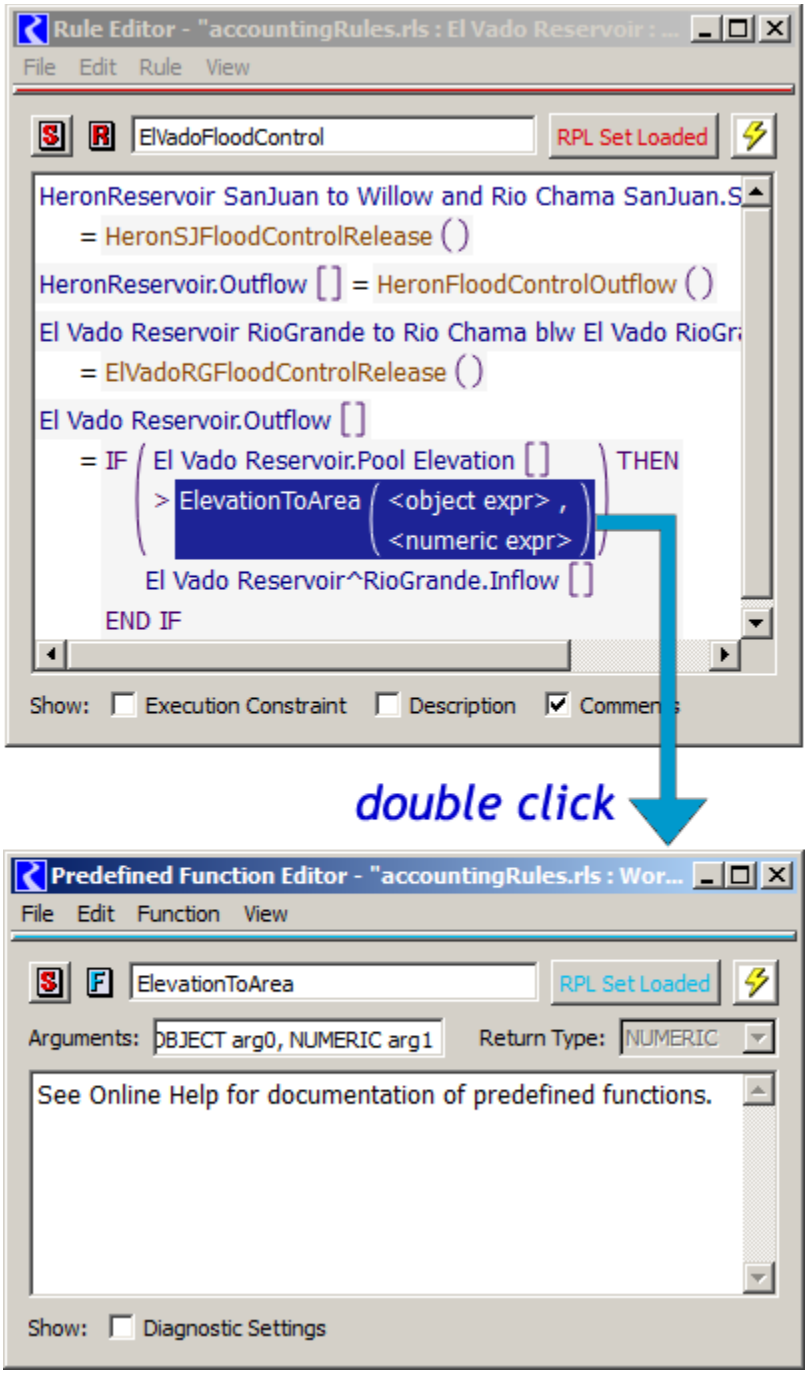


The places within the RiverWare RPL GUI in which online help for these topics is relevant are:

1. RPL Palette / Palette Buttons Tab (for "statements and operators" help).
2. RPL Palette / Predefined Functions Tab (for "functions" help).
3. RPL Frame (editor panels) in which a predefined functions, statements and operators are used / Selected sub-expression (for both types of help).

Content for these online help topics could potentially be displayed in these places. *These possibilities are discussed more below.*

1. In a panel within the dialog from which context-help was requested.
2. In a panel within the dialog corresponding to the item for which context-help is being requested.
3. In a single reusable (singleton) online help viewer window within RiverWare.
4. In an external viewer/browser (e.g. Adobe Reader for PDFs, or a web browser for HTML).



(1) **Showing help within the dialog from which context-help was requested:** There are several types of dialogs in which RPL Frames (editor panels) are used (this being one of the types of places from which context help would be requested). Each of these contexts, plus the two relevant RPL Palette tabs (*see above*) have their own GUI layout constraints, and would require different and separately implemented changes to add a "help" panel. We're deciding that this lack of conformity and broad development scope is undesirable.

(2) **Showing help within the dialog corresponding to the item for which context-help is being requested:** Of the items having help topics supported by this enhancement, only RPL Predefined Functions have an associated dialog. And, in fact, the RPL Frame panel in the RPL Function Editor dialog (for user-defined functions) doesn't currently have a use when showing a

predefined function. The panel just shows this message: *"See Online Help for documentation of predefined functions."* This is a natural place to show documentation for the predefined function.

(3) **Showing help within a single reusable (singleton) online help viewer window within RiverWare:** Being that individual RPL statements and operators don't have their own dialog, this is a reasonable place in which to show help topics for those RPL language constructs.

(4) **Showing help within an external viewer/browser.** (This is what's currently being done for online help, to show PDFs in a PDF viewer, typically Adobe Reader). There are ways of having certain versions of certain viewers navigate to specific places within an external document. How this is done can be dependent on which PDF viewer program the user has installed and configured. It is not generally a reliable approach for implementing primary help functions. If possible, this approach will be attempted for links *within* RPL Predefined Function and RPL language-construct help displayed in RiverWare (to other online help published in PDF). But we will not use this approach for that primary help content (i.e. the new functionality proposed by this document).

(2b) Help Content Architecture

Given the set of requirements of a future, modernized documentation architecture (*described above*), it's a forgone conclusion that RiverWare documentation will need to support HTML as a *display* format* in addition to PDF. This is not to say that HTML should be the *source* format. We will want to manage our content with high level tools supporting associated meta-data (so that specific content can be accessed programmatically) and integrated advanced authoring capabilities (such as for mathematical formula display).

> *Technically, conveying documentation content to the RiverWare program with XML and associated XSLT files for generating HTML would be the preferred approach. This is effectively equivalent to providing HTML for display, but also with the advantage of providing well-defined and easily usable meta-data.

With respect to the current task of enhancing RiverWare to display some RiverWare RPL documentation within the GUI, an intermediate step is doing so with the HTML format -- *regardless of how that HTML content is generated.* In this way, any use of that HTML content by the RiverWare program will not have to be changed very much when a new documentation architecture is devised.

> Note: RiverWare's recently developed model report capabilities demonstrate displaying HTML content with CSS and JavaScript. This makes use of Qt WebKit in Qt 4.8 which continues to be in active development in Qt 5. Qt WebKit does support transforming XML documents to HTML via XSLT. It also supports SVG, scaled vector graphics, image file rendering. RiverWare is currently using Qt 4.8.5. We anticipate upgrading to Qt 5.2 or beyond within the next year or two.

> Note: We did take a look at some possible technologies to display PDF (existing RiverWare documentation) with Qt in RiverWare. That would require integration of new

special 3rd-party libraries (e.g. Poppler, or the Adobe PDF Library SDK). But the fact that PDF content is presented with a fixed-width limits the value of this approach, given that, in some contexts, we would want to display documentation content in narrower panels.

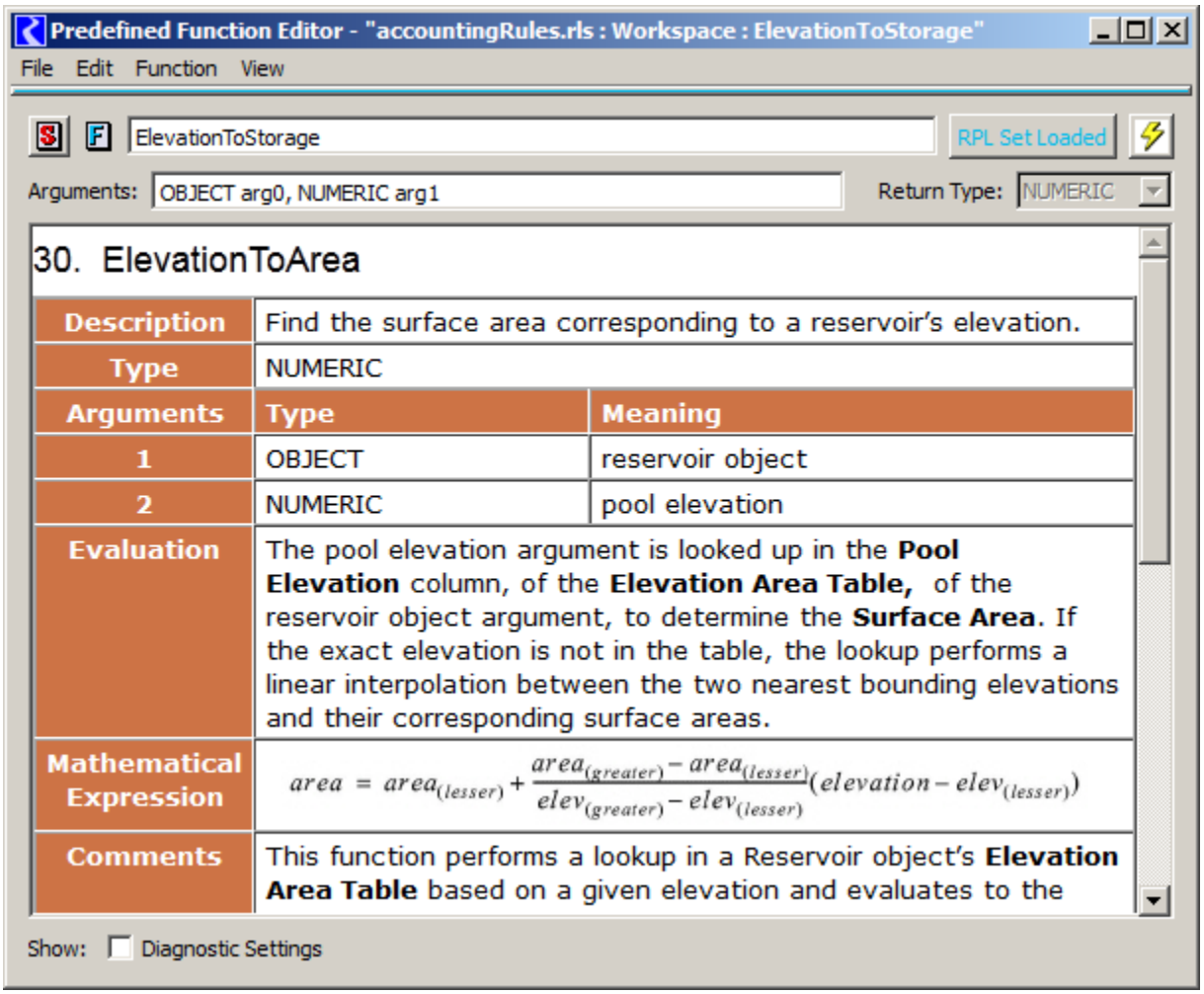
The scope of this particular development is limited to providing access to the most needed parts of the RPL documentation using our current document authoring procedures and tools (using Adobe FrameMaker), possibly accommodated with changes to the documentation source, an upgrade to a newer version of FrameMaker (upgrading from version 10 to 12), and post-processing of generated HTML files.

(3) Design

The following RPL online help topics can be displayed within the RiverWare GUI:

1. RPL Predefined Functions (191 functions).
2. RPL Statements and Operators / Buttons in the RPL Palette (60, in 8 categories).

RPL Predefined Function help content is displayed in the "editor" dialog for the function -- in place of the formerly empty panel which used to display only the message, *"See Online Help for documentation of predefined functions."* (The following image is a pre-development mockup).



RPL Statement and Operator help content is displayed in a single separate RiverWare Help window (a singleton) -- *not illustrated here*. A "history" combo-box allows the user to revisit help topics recently shown in this dialog.

Content for these two different types of dialogs is provided to RiverWare as HTML and may contain references to **image files** of a format typically supported in webpages (PNG, GIF, JPG, SVG). To the extent possible, text wraps to the available visible horizontal space, though the

presence of an image (e.g. the mathematical expression shown above) may constrain the minimum content width. In that case, a horizontal scrollbar is displayed when the panel is narrow.

The help content includes **hyperlinks** to particular "named destinations" within the 30 (or so) RiverWare help PDF documents. When shown in RiverWare dialogs, these links at least show the referenced PDF file in the user's configured PDF viewer (generally Adobe Reader). It may or may not be possible to navigate directly to the named destination within the target document. (This may depend on the user's environment).

A **RPL Predefined Function** "edit" dialog (showing help content) can be shown from these places within RiverWare, as indicated:

1. From a RPL Frame (editor), double clicking on the use of a RPL Predefined Function.
2. From the RPL Palette's "Predefined Functions" tab (supporting single-item selection in a list or tree of the available predefined functions):
    1. Function >> "Show Help..." menu operation.
    2. "Show Help..." context-menu (right-click) operation
    3. Question-mark-circle icon button, to the right of the "Set Name" row.

The new RiverWare Help dialog (initially used only for showing help for a **RPL Statement or Operator)** can be shown from these places within RiverWare, as indicated:

- From a RPL Frame (editor), double clicking on the use of a RPL Statement or Operator.
- From the RPL Palette's "Palette Buttons" tab, Right-Clicking OR Shift-Left-Clicking on a button.
  ... the tooltip on those buttons show this message on the 2nd line: "Shift-Click to show help".

Note that, in all three tabs of the RPL Palette, *double clicking* an enabled item (or just single clicking on a palette button) causes the selected sub-expression in the current RPL Frame to be *replaced* with the palette item's content! So obviously those events can't be used for showing help content.

> The new "help" functionality may exacerbate a usability problem -- it may be too easy to unintentionally change a RPL block or user-defined function. We might want to consider adding a RPL Edit Lock which disables changes to any RPL code within RiverWare.

Technical Design Issues

The help content is "logically" provided to RiverWare as individual HTML documents -- one for each help topic. But instead of distributing hundreds of separate help HTML and image files with RiverWare, these files are embedded within the RiverWare executable using Qt Resources. (Qt Resources have previously been used in RiverWare for binding icon image files). RiverWare needs to associate the embedded help topic "documents" with the corresponding entities within

RiverWare (i.e. RPL Predefined Functions, and such). This is done through file and Qt Resource naming conventions.

(4) RPL Document Processing

It is important for us to maintain only a single copy of RiverWare Online Help document source files. We need a way of getting the help content from our FrameMaker-sourced documents into RiverWare.

The largest part of the development process will be the creation of a *publishing process and tools* (a post-processing script) used in generating the required data *from* FrameMaker. The end-product of that publishing process consists of a set of files which are bound to the RiverWare executable. The format design of those files is effectively an interface from which both "sides" can be independently developed:

1. Generation of help content and supporting data from FrameMaker help document source.
2. RiverWare enhancements to support that help content.

The latest version of FrameMaker (version 12) has a couple different HTML and XML-generation capabilities. Before starting this project, we will first upgrade FrameMaker -- and our supporting tools and procedures -- from version 10 to version 12. However FrameMaker 12 will not be able to completely provide the necessary and desirable data provisions in the generated files. We will be able to address certain data generation issues with changes to the relevant source documents. But for some requirements, will need to write a post-processor to munge the files generated by FrameMaker. (I recommend using the Python scripting language which supports both DOM and SAX XML document processing).

We have experimented a bit with HTML document generation using FrameMaker 12, and have observed the following issues (some of which we may be able to address with either FrameMaker configuration changes or document content provisions):

1. Text style properties are not optimal (e.g. larger font for non-font-size related differences).
2. Table cell background color is either not generated, or applies to the whole table instead of individual cells.
3. It's possible that only the "traditional" way of generating HTML (which lacks certain newer provisions) has the ability to generate separate files for different sections (e.g. defined as starting with a "Header1" element). But even with that, there doesn't seem to be a way of having the file name correspond to a content-dependent character string.
4. Similarly name-anchor ("named destination") strings are not based on content strings.
5. The generated HTML has *in-lined* CSS styles, and a ton of them. This is not a particularly helpful use of CSS, and makes the generated files much larger than they need to be. This is significant because we will be embedding these files within the RiverWare executable.

6. Supported image file formats for *mathematical expression images* are limited to the basic *raster* formats (GIF, PNG, JPEG). Vectored graphics (SVG) doesn't seem to be supported for that.

We're proposing that a post-processor be developed, to be applied to files generated from FrameMaker, to effect the following changes in new generated files:

- Extract only the actual structure, content and hyperlinks from the generated HTML, dropping all in-lined CSS style attributes.
- Translate HTML CSS style class names to semantic classes based on content. (*See discussion of the example, below**).
- ... similarly, translate name-anchors to content-based strings, and image file names.
- If necessary, split monolithic HTML files into separate well-named files.
- Generate Qt Resource definition files (.qrc) for binding with RiverWare.

*Appendix B provides file-system links to several versions of a manually cleaned up HTML file (e.g. with exact style information removed), and the new external CSS file which provides all styles. This sort of CSS file would be used for all RPL Predefined Function and RPL language construct topic documents within RiverWare.

(5) Development Tasks

(A) RPL Document Processing

The goal is to devise and document a process and create supporting tools to export HTML (or XML) from the two relevant FrameMaker source files (using FrameMaker 12) and from that, (primarily, probably with a Python script) generate HTML and supporting data files to be built with RiverWare. While it's likely that some goals can be accomplished with changes to FrameMaker settings and technical changes to the source document, at least some post-processing will ultimately be required. That being the case, we probably shouldn't spend too much time on the former types of changes since, once a script infrastructure is developed, it should be relatively easy to effect needed changes with mainly a post-processing script.

FrameMaker setting changes and technical source document (to coerce the generated content into desirable forms) will mostly be an iterative process with post-processor script development. After some initial preparation, those won't be separate development tasks. See also the prior section, "RPL Document Processing".

Development tasks include:

1. Insure that the two source documents include distinctly styled headings which can be discerned in the generated HTML file. If needed this may require going through all of the 191 functions and 60 language constructs to apply new semantic-level FrameMaker styles.
    1. RPLPredefinedFunctions.pdf -- RPL Predefined Functions
    2. RPLTypesPalette.pdf -- Palette Buttons
2. Create Python Script framework and running stub. The script reads in the generated HTML file(s) as an XML DOM document, and trivially iterates over the elements. (It is important that FrameMaker generates HTML files which are proper XML).
3. Predefined Function Processing.
    1. Discern function content pieces with FrameMaker style classes and magic keywords in textual content.
    2. Create dictionaries for translating "named destinations" and image file names to natural names (using actual RPL Predefined Function names). (This is needed for RiverWare to be able to associate help content with particular RPL functions and language constructs).
    3. Image file processing. This is just copying image files generated from FrameMaker to files with the required names; no actual image processing is involved.
    4. Generate new HTLM output for each function, referring to our own CSS styles. (See Appendix B example).
4. Palette Button (Statements, Operators) Processing
    - Similar steps to previous item, but reusing some common code.
5. Add generation of Qt Resources index file (.qrc). This is used in the RiverWare build to bind the generated HTML and image files.

6. Document the generation process, including operation of FrameMaker with the two source documents.

(B) RiverWare Development

1. Integrate generated Qt Resources index file (.qrc). Implement and test retrieval functions for generated HTML and image files based on the names of RPL Predefined Functions and supported RPL language constructs.

2. RPL Predefined Function "editor" dialog enhancements. Deploy a QWebView (from Qt WebKit) in this dialog. All features here are display-only (except for the issue addressed in the next step). Note that the code to show this dialog from a RPL Frame (double clicking on the use of a pre-defined function) is already in place.
3. Explicit handling of hyperlinks in the QWebView created in the prior step. If it is possible to determine that the link is to another predefined function, show that function's "editor" dialog. Otherwise, open the referenced PDF document (with a correctly devised path) in the user's standard PDF viewer application. If possible, navigate to the specified "named destination" in that document.
4. Add "Show Help..." hooks from the RPL Palette's "Predefined Functions" tab. Also, Question-mark-circle icon button.

5. New Help Viewer dialog (used initially only for supported RPL language constructs: the 60 buttons). This includes a QWebView (similar to above). The only active component is a history combo box to jump to previously (or again, subsequently) viewed topics.
6. In RplFrame, double clicking on a Statement or Operator, show the Help Viewer with corresponding help topic.
7. In the Rpl Palette’s "Pallet Buttons" tab, implement special button processing and tooltip. Shift-Clicking shows the Help Viewer with the corresponding topic.

8. Correctness and usability testing, review and revisions.
9. Feature Documentation.

(6) Development Estimate

Task	Est. Hours	Task Description
(A) RPL Documentation Processing (total: 96 .. 128)		
A1	4 (..12)	Initial Document Preparation / Semantic Styles (where needed)
A2	16	Python Script framework, running stub.
A3	40 (..56)	Predefined Function Topic Processing
A4	24 (..32)	Palette Button Topic Processing
A5	8	Generation of Qt Resources index file (.qrc)
A6	4	Process Documentation
(B) RiverWare Development (total: 88)		
B1	8	Integrate generated Qt Resources. Develop lookup utilities.
B2	16	RPL Predefined Function "editor" dialog enhancements
B3	16	Explicit handling of hyperlinks
B4	4	RPL Palette Predef Functions: Add "Show Help..." hooks
B6	16	New Help Viewer dialog
B7	4	RPL Palette Buttons, special button handling
B8	16	Correctness and usability testing, review and revisions.
B9	8	Feature Documentation
	184 (..218)	TOTAL [hours]

Appendix A: RPL Documentation Topics

RiverWare 6.4 Documentation has six (6) documents covering "RPL" topics (including "Rulebased Simulation"). For the purpose of analyzing the types of information to which this RiverWare enhancement could apply, three types of topics have been characterized:

(A) Enumerated Item Topics
(B) Discussion Topics
(C) User Interface Description

(A) Enumerated Item Topics:

- RPL Predefined Functions (191 functions with images for about 77 formulas, 3 workspace screenshot details, 1 graph).
- RPL Statements and Operators / Buttons in the RPL Palette (60, in 8 categories).
- RPL Data Types (7) (10 pages).

(B) Discussion Topics

- Units in RPL: Unit operators / Slot Value Units (2 pages)
- RPL Language Structure (8 pages)
- Developing Efficient RPL Expressions [in RPL User Interface] (1 page).
- Hypothetical Simulation Overview [in RPL Predefined Functions] (1 page).
- How Rulebased Simulation Works [in Rulebased Simulation]
  ... Rules and Rulesets, Rule Execution, ...

(C) User Interface Description

- RPL Debugging and Analysis Tools
  - Building and Validation Errors
  - Evaluation and Runtime Errors
  - RPL Debugger (12 pages)
  - Diagnostics (4 pages)
  - Rulebased Simulation Model Run Analysis Tool (*link to Model Run Analysis doc*)
  - RPL Analysis Tool (9 pages)
- RPL Editor Dialogs (32 pages)
- RPL Printing and Formatting (6 pages)
- Exporting and Importing RPL Sets (4 pages)
- RPL External Documentation (user defined) (14 pages)

Appendix B: Sample Function HTML and CSS help files

As a demonstration, I edited an HTML file generated from FrameMaker 12 into a "clean" format, with semantic tags. All formatting and display style information was removed from the HTML file. Instead, display styles are specified by the illustrated CSS file (*see below*). *The mockup HTML screenshots in this document are of this munged HTML and CSS files -- rendered in FireFox.*

The idea is that this could be done programmatically, e.g. with a Python script with the use of XML DOM (and maybe SAX for reading input).

See sample HTML files in this RiverWare documentation directory:

- R:\doc\RPL\EmbedRplDoc\2014\HtmlSpecExamples\
  - ElevationToArea-FmOutput.html -- *original HTML file from FrameMaker*
  - ElevationToArea.html -- *cleaned up HTML file using CSS shown below.*
  - ElevationToArea-NoCss.html -- *lacking CSS file link*
  - ElevationToArea-NoImage.html -- *image omitted*

CSS File -- Semantic and Formatting Classes/Styles Example:

```
@charset "utf-8";

/* ----------------------------------------------------------------
 * File: RplFunc.css
 * Edit: Phil, 3-17-2014
 * Mockup for Predefined RPL Function HTML doc import into RiverWare.
 * ---------------------------------------------------------------- */

/* Semantic Classes */
.RplFunc_Name       {}  /* function name*/
.RplFunc_RetType    {}  /* return type */
.RplFunc_ArgType    {}  /* argument type (attribute: argInx [1..]) */
.RplFunc_ArgName    {}  /* argument name (attribute: argInx [1..]) */
.RplFunc_Evaluation {}  /* evaluation description */
.RplFunc_Comments   {}  /* other comments */

/* .RplFunc_Syntax_Examp  -- syntax example; defined below. */
/* .RplFunc_Return_Examp  -- return example; defined below. */

.RplFunc_FunctionHdr {
        font-family: Arial, Helvetica, sans-serif;
        font-size: larger;
        margin-top: 0px;
        margin-bottom: 8px;
}
.RplFunc_CellRowHdr {
        font-family: Verdana, Geneva, sans-serif;
        color: #FFF;
        background-color: #cd7345;
```

```
        vertical-align: top;
        font-size: small;
        font-weight: bold;
}
.RplFunc_CellRowHdr:first-child {
        text-align: center;
}
.RplFunc_CellText {
        font-family: Verdana, Geneva, sans-serif;
        vertical-align: top;
        font-size: small;
        margin-left: 4px;
}
.RplFunc_CellText p {
        margin-top: 10px;
        margin-right: 5px;
        margin-bottom: 2px;
        margin-left: 5px;
}
.RplFunc_CellText p:first-child {
        margin-top: 2px;
}
.RplFunc_GUItext {
        font-weight: bold;
        font-family: Verdana, Geneva, sans-serif;
        font-size: small;
}
.RplFunc_ExampleTitle {
        font-family: Verdana, Geneva, sans-serif;
        font-size: small;
        margin-top: 6px;
        margin-bottom: 6px;
}
.RplFunc_Syntax_Examp, .RplFunc_Return_Examp, .RplFunc_RuleCodeIndent {
        font-family: "Courier New", Courier, monospace;
        margin-left: 20px;
        font-size: small;
        margin-top: 6px;
        margin-bottom: 6px;
}
.RplFunc_BlueLink {
        color: #03F;
        text-decoration: none;
}

/* --- (end RplFunc.css) --- */
```

--- (end) ---