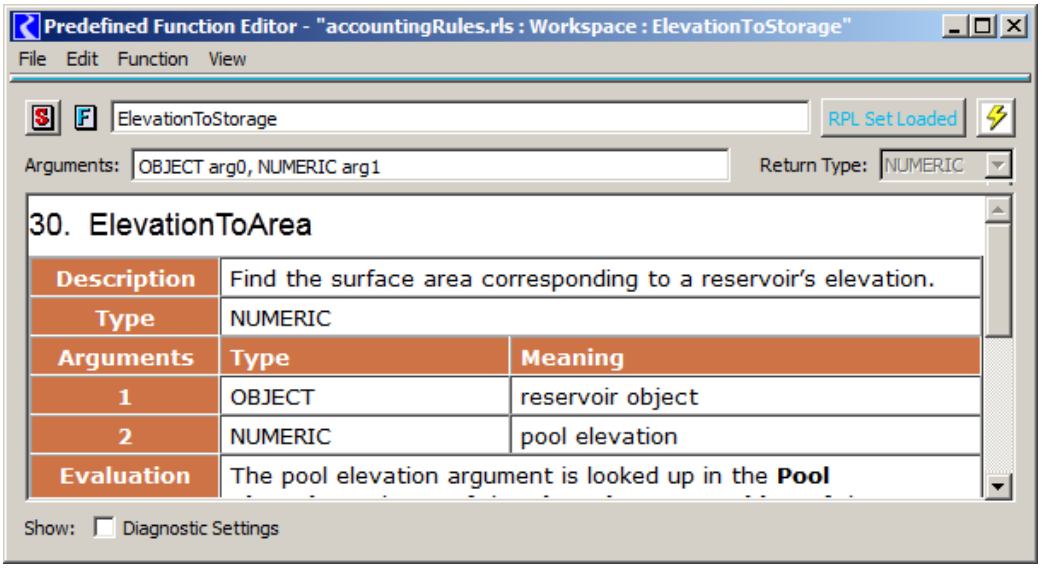
RiverWare: Improved Access to RPL Documentation / Analysis and Design

Phil Weinstein, Patrick Lynn, CADSWES, 3-09-2014

This document proposes enhancements to RiverWare (beyond version 6.4) to display online help content within RiverWare for RPL Predefined Functions and RPL expression types.

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 "menu" (table of contents) document is also provided. 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. Many of the functions have graphically implemented mathematical formulas, usually just one, but some functions have multiple. Overall, there are 77 mathematical formulas images. Two functions have associated workspace screenshot details.

- 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).

The RPL Predefined Functions are presented in alphabetical order. But a two-page section, **"Hypothetical Simulation Overview"** appears just before the relevant hypothetical simulation functions. (For the purpose of supporting dynamic handling of RPL Predefined Function help content, it may become important to move this descriptive section to a different place within RiverWare documentation).

Each RPL Predefined Function has its own section, with most of the information laid out in a table. Below is an example.

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]
```

We would like to make the following changes to the current RPL Predefined Function documentation:

1. Not all functions have an introductory sentence above the table (as depicted above). Where present, that text is redundant with the "Description" row in the display table. We would like to either remove that introductory sentence or merge it with the "Description" text.

2. The descriptions for each of the arguments are either one-or-two words, or full sentences. We would like to have something usable as a *formal argument (parameter) name* distinct from a longer description. Having formal argument names available for function arguments would allow us to use those names within RiverWare in place of the generic "arg0", "arg1", "arg2", … parameter names.

3. In documentation, RPL Predefined Function arguments are numbered starting at one (1). But in the user interface, e.g. where generic numbered arguments are referred to, they start at zero, i.e. "arg0", "arg1", "arg2",…. We will want to reconcile this.

(1d) Help Content: RPL Expression Types

RPL Expression Types 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

The “RPL Palette” section contains the following eight subsections, each with a table of related items, one row per expression type:

1. Mathematical Operation Buttons
2. Logical Operation Buttons
3. Object and Slot Lookup and Assignment Buttons
4. Unary Operation Buttons
5. Buttons for Setting Flags on Slots
6. Conditional and Iterative Operation Buttons
7. List Operation Buttons
8. Miscellaneous Buttons

The Mathematical Operation Buttons section is shown below. That section consists of only the table of operations. Some sections have additional paragraphs of text.

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.

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.

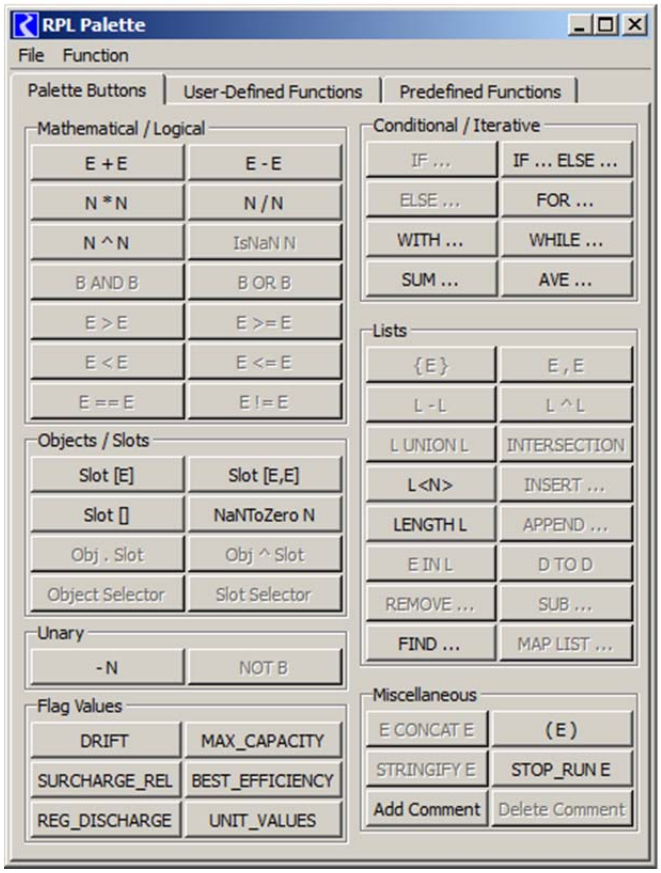
(2) Requirements Analysis

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

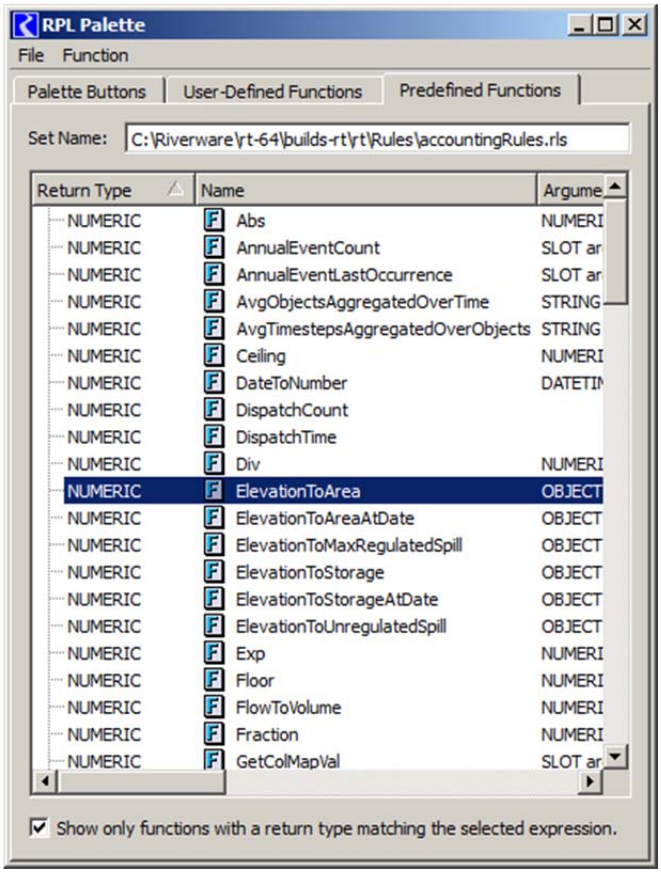
This proposal focuses on two of the "enumerated item" topics (among the various RPL documentation topics listed in Appendix A).

1. RPL Predefined Functions (191 functions).
2. RPL Expression Types / Buttons in the RPL Palette (60, in 8 categories).

RPL Palette Buttons



Predefined Functions



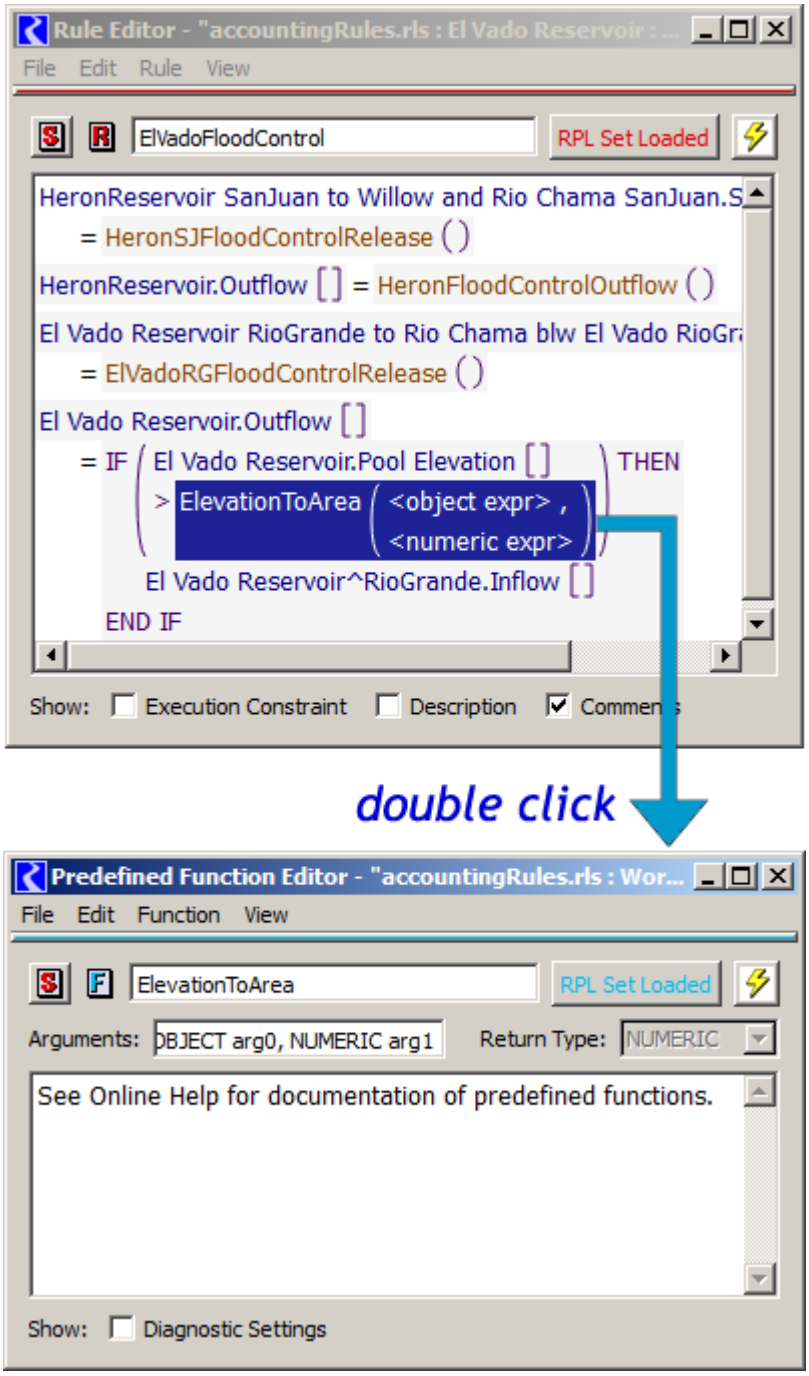
This proposal addresses adding help support for only: (a) RPL Predefined Functions, and (b) RPL expression types. Not addressed is help support for **RPL Statements.** Note that some of the Conditional and Iterative expression types look like certain RPL Statements (e.g. the IF … ELSE, and FOR …) – but they are not related.

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

1. RPL Palette / Palette Buttons Tab (for expression type help).
2. RPL Palette / Predefined Functions Tab (for "functions" help).
3. RPL Frame (editor panels) in which a predefined functions, expressions 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 (intended 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 expression types 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.

> Note: Our decision to use Qt 4's general QDesktopServices library for showing external documents certainly has benefits. But for viewing PDF files, that approach doesn't allow us to pass a "named destination" command line parameter to the Adobe Reader program, nor even specify the viewer application to be used.

If an HTML version of any of the PDF documents were made available, it would be possible to *reliably* navigate to a particular section within that document – assuming the "anchor" (or "id" element attribute text value) for the desired section is known. Unfortunately, depending on the default browser on the user's system, a new instance of the browser -- or a new tab within an active instance -- may be created *each time* RiverWare attempts to show a section of the documentation.

(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 (assuming that the XML actually has semantic structure rather than just being an XML representation of display-oriented content). 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 certain 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 5.3 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.

(2c) Support for Formal Argument Names

In the RiverWare user interface, RPL Predefined Function arguments (parameters) are often designated with a value type (e.g. NUMERIC) and a numbered argument name, i.e. “arg0”, “arg1”, “arg2”, etc. It would be desirable to:

1. Assign meaningful names (*formal argument names*) to each of the arguments in all of the RPL Predefined Function.

2. Use those formal argument names in place of the arbitrary numbered argument names wherever they appear in the RiverWare GUI and RPL diagnostic messages.

Some thought should go into naming conventions for formal argument names among the existing 191 RPL Predefined Functions. We need to consider the uses of these names in the RiverWare user interface, e.g. with regard to how terse these argument names should be. Note that it may be inconvenient to change these names in the future.

Formal argument names should start with a lower-case letter. They cannot contain spaces – but they could be *camelCased.* And they should not be very long.

See the subsequent section, “Design Level 2: Improvements to RPL documentation,” for further discussion of this feature.

(3) Design

There would be quite a bit of value in providing access from RiverWare to the RPL Predefined Function and RPL Statements and Operator documentation in basically its current form – that is, without significant content revision.

In particular, the feature of presenting well-formulated *formal argument names* for RPL Predefined functions is significantly independent from importing RPL documentation content into RiverWare, or otherwise providing access to that documentation from RiverWare.

Designs to support two levels of functionality are proposed separately in these two sections:

- (3a) Design Level 1: Access to existing RPL documentation
- (3b) Design Level 2: Improvements to RPL documentation

(3a) Design Level 1: Access to existing RPL documentation

The following RPL online help topics can be displayed by RiverWare:

1. RPL Predefined Functions (191 functions).
2. RPL Expression Types / Buttons in the RPL Palette (60, in 8 categories).

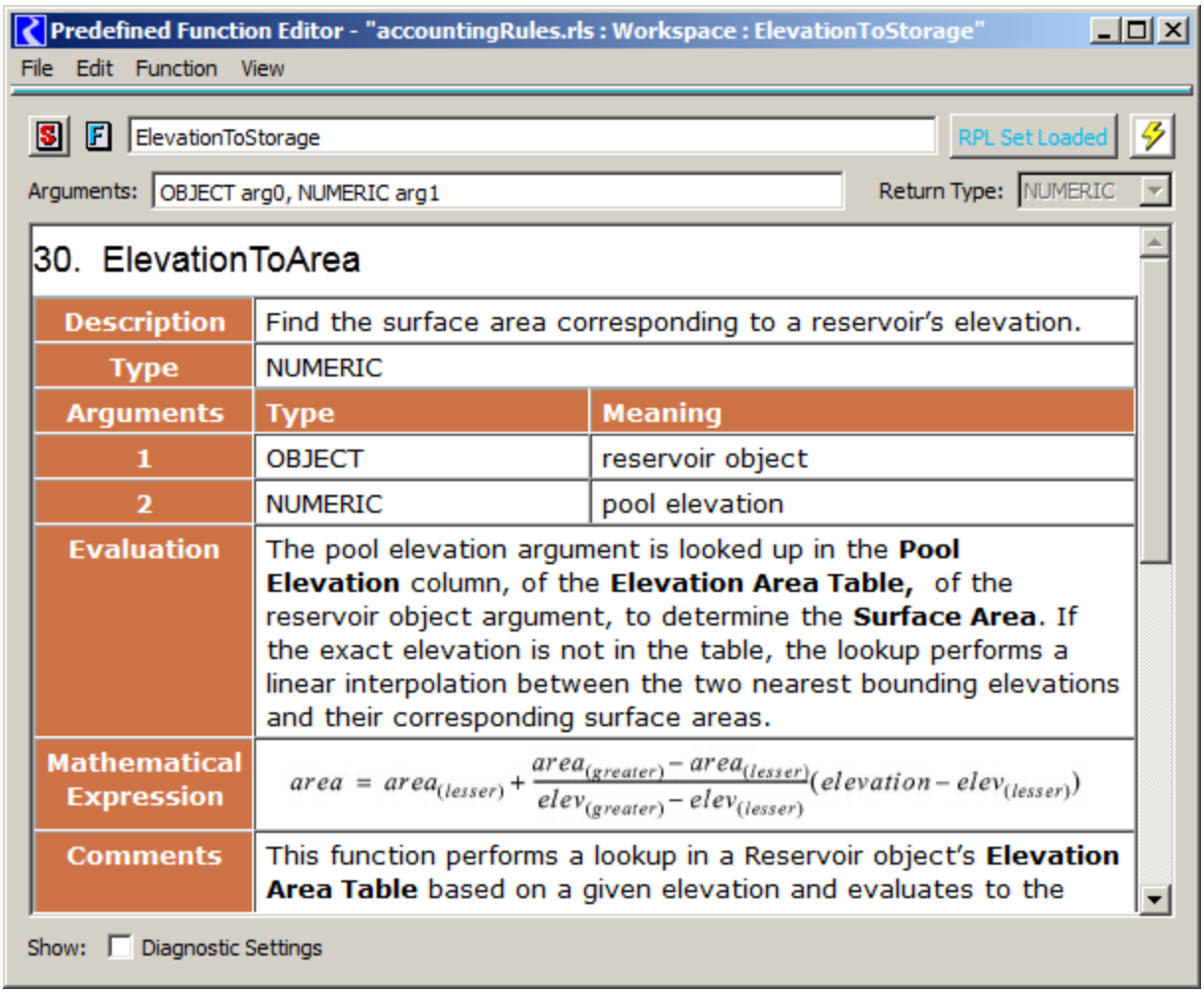
Content for these two types of information are provided to RiverWare via two HTML files having some semantic structure and markup, plus about 80 image files.

The RPL Predefined Functions HTML document has an overall “div” element *for each RPL Function.* Those function elements contain other elements having attributes to distinguish a function’s description, return data type, “evaluation” description, mathematical expression, additional comments, and for each argument: the argument data type, and “meaning” description.

The two HTML files and associated image files are incorporated into RiverWare using Qt's resource system (i.e., included in the build tree, referenced by a .qrc file, compiled using rcc as part of the build process). The overall publishing process is described in a latter section.

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).

Additionally, the “description” sentences for RPL Predefined Functions are displayed on the RPL Palette’s Predefined Functions tab in an optionally shown “Description” panel below the function list (similar to that panel on the User Defined Functions tab).



Note [TENTATIVE]: The Predefined Function Editor's "Arguments" and "Return Type" display widget line is now technically redundant with that information in the displayed help content. But that is still shown so that there is a consistent place within the dialog where these values are shown. Note that, within the displayed help content, those details may be scrolled out of view. We might, however, choose to move the Return Type field (*see above*) to the left side.

RPL Expression Type 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. This window will show the entire HTML version of the RPL Data Types and Palette document, scrolled to the section containing the relevant expression type.

> **Alternative:** Instead of displaying RPL Expression Type help content in a new RiverWare Help window, we could show an HTML version of the RPL Data Types and Palette document, scrolled to the section (one of eight) describing the item, in a web browser. Since an HTML URL can include an anchor name (or element ID attribute value), this can be done reliably. However, depending on the user's default web browser, a *new browser window* or a *new tab* in the browser may be created *each time* RiverWare shows this help content.

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. In the current level of documentation support, hyperlinks in help text displayed in RiverWare are treated as follows:

- Links to RPL Predefined Functions are supported by showing the Predefined Function Editor for the linked function.
- Links to sections in the RPL Data Types and Palette document are supported by showing that document in the new RiverWare Help window, scrolled to the designated section.
- All other links are removed. Note: we will need to change the presentation of links within the two relevant FrameMaker documents to use the word “See …” following by a link on the name of the linked section having the actual link markup, rather than having the link on words like “CLICK HERE”. In this way, semantics will be preserved when just removing the unsupported link (anchor) tag.

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

1. From a RPL Frame (editor):
    - double clicking on the call to a RPL Predefined Function.
    - Clicking on a new question-mark-circle icon button, which shows help on the selected expression. This button is enabled only when help for that expression is available.
2. From the RPL Palette's "Predefined Functions" tab (supporting single-item selection in a list or tree of the available predefined functions):
    - Function >> "Show Help..." menu operation.
    - "Show Help..." context-menu (right-click) operation
    - 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 Expression Type)** can be shown from these places within RiverWare, as indicated:

- From a RPL Frame (editor):
    - Clicking on a new question-mark-circle icon button, which shows help on the selected expression. This button is enabled only when help for that expression is available.
- 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.
>
> Another provision we might consider to alert the user that the primary actions of the RPL Palette have an edit effect in another dialog would be the following. We could add a line to the top of the RPL Palette dialog (visible when any of the three tabs are selected) indicating the entity (Rule, User-Defined RPL Function, or RPL Expression Slot) which is currently being edited. Also the corresponding RPL Object or Slot icon could be presented as a small button which raises that active editor dialog. For example:
>
> EDITING: [F] Function: DeltaTimeToMarch

Technical Design Issue

> The help content is "logically" provided to RiverWare as individual HTML and image files. But instead of distributing these many files with RiverWare, they 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.

(3b) Design Level 2: Improvements to RPL documentation

Although modifications of the two relevant FrameMaker source documents will be required for technical reasons to implement the design described above, the task of supporting new formal argument (parameter) names for RPL Predefined Functions is a significant isolatable task which could reasonably be addressed as a separate enhancement.

> Note: This enhancement may not even be entirely desirable. If formal parameter names are not kept especially short, the few contexts in which they appear may be visually "overwhelmed" by long names – the latter ones may even typically be *out of view*. I [Phil] do recommend that we implement the foregoing functionality first, and get a feel for what that looks like before proceeding with this additional enhancement.

There is also the idea that we want to either remove, or combine with the "description" cell, the leading sentence which appears above the main display table in most RPL Predefined Functions' help content. That change is not technically required, and can readily be handled as a separate task involving only edits to the FrameMaker source. (Note however that we will not attempt to

mark those leading sentences with semantic tags for use by RiverWare, since this content change is likely to be done at some point).

The "Arguments" rows in a RPL Predefined Function documentation table, in the legacy implementation, contain these columns:

- Column 1: Argument Number (integer, 1 or greater).
- Column 2: Argument Type (e.g. OBJECT or NUMERIC)
- Column 3: Meaning – sometimes one or two words, and often one or more sentences.

Three options can be considered for adding a formal argument name.

Option 1: The first column could include both the argument number and the argument name, separated with a colon, for example:

2: poolElevation

Option 2: The formal argument name could be placed at the beginning of the Meaning column, followed by a colon, for example:

col: column index of the table slot (0-based).

Option 3: This new column could be added for a formal argument name. There would be four columns:

- Column 1: Argument Number (integer, 1 or greater).
- Column 2: Argument Name, one word, possibly camel cased, e.g. "poolElevation".
- Column 3: Argument Type (e.g. OBJECT or NUMERIC)
- Column 4: Meaning – sometimes one or two words, and often one or more sentences.

Not adding a fourth column (i.e. not choosing Option 3) would have the benefit of the content displaying better in a window or panel having limited width.

The places in the RiverWare GUI where formal argument names would be used in place of the numbered argument names (e.g. "arg0") include:

1. RPL Predefined Function editor dialog, in the Arguments field (unless we decide to eliminate that because it is technically redundant with the displayed help text).
2. RPL Palette's Predefined Functions tab, in the Arguments column of the function list.
3. Possibly as a tooltip on an actual parameter in a RPL frame, i.e. in the context of the use of a RPL Predefined Function. This could potentially be supported whether or not the formal parameter was specified, e.g. even if the parameter is shown as "<object expr>".
4. In diagnostics where particular RPL Predefined Function arguments are cited.

On RiverWare startup, RPL Predefined Function formal argument names would be read from the help data and effectively placed on PredefinedFunction objects.

(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.

Native RiverWare help for RPL will be based on two HTML files, one for predefined functions and one for expression types, with additional image files (approximately 80). These files will be incorporated into RiverWare using Qt's resource system (i.e., included in the build tree, referenced by a .qrc file, compiled using rcc as part of the build process).

These HTML files:

- Need to be created from the Framemaker source document so that we are not maintaining documentation in two places.
- Need to contain non-displaying content (semantic mark-up) that indicate which part of the content is relevant to which function or statement/operator.
    - Each RPL expression type needs a well-known anchor for the corresponding document section, discernible by RiverWare.
    - All the content for a particular RPL Predefined Function should be contained within a high-level element (an HTML “div”). (This is not relevant for expression types, as that is not being separated out as independently displayable content).
    - The individual content *components* of each function should be distinguishable by RiverWare: a function’s name, description, return data type, “evaluation” description, mathematical expression, additional comments, and for each argument: the argument data type, formal name (when supported) and “meaning” description.

The FrameMaker "publish" functionality produces HTML, but it does not have the desired semantic mark-up required by RiverWare. It can't have the mark-up because the source document does not have the relevant information.

Whenever we do a release with new on-line help (not generally for patch releases), we add a couple of steps after the release source code branch is created and after on-line help has been updated for the release:

- In Framemaker, publish the two relevant RPL documents (operators, predefined functions) to HTML.
- Run the Python script to generate two improved html files with image files.
- Copy these files to the builds tree, recompile, commit.

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.

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. RPL Predefined Functions are not contained within their own single HTML element.
2. Text style properties are not optimal (e.g. larger font for non-font-size related differences).
3. Bullets (in lists of items) are showing up as garbled data.
4. Table cell background color is either not generated, or applies to the whole table instead of individual cells.
5. Name-anchor ("named destination") strings are not based on content strings (so, as is, RiverWare would have no way of associating a location within the document with a particular function or expression type).
6. 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. Also, even though styled elements have a useful class name, the inlined CSS style attributes would prevent overriding, as they have higher priority than class-based styles.
7. 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 are not proposing a solution to this limitation. Raster images will be presented within RiverWare for this content].

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**).
- Translate name-anchors and image file names to content-based strings. In the course of changing image names, the script would also copy the images generated from

FrameMaker to new image files with the new name (e.g. based on the RPL Predefined Function name).
- 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 expression type topic documents used within RiverWare.

Some experimental work was done to develop a Python-language script to transform HTML generated from FrameMaker into HTML having the required structure (e.g. RPL Predefined Functions encapsulated within their own high level element) and semantic markup (e.g. to identify content for individual function arguments). The basic structure of the original HTML document is retained. Beyond removing inlined CSS style attributes, markup not explicitly transformed by the script is retained as is. This script makes use of the Python 3.4 standard library's XML DOM (*Document Object Model*) processing "minidom" module.

(5) Development Tasks

(A) RPL Document Processing

The goal is to devise and document a process and create supporting tools to export HTML from the two relevant FrameMaker source files (using FrameMaker 12) -- and from that, (primarily, with a Python script) generate HTML and supporting image 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.

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 sufficiently meaningful "paragraph" styles which can be discerned in the generated HTML file. If needed this may require going through all of the 191 functions and 60 expression types to apply new semantic-level FrameMaker styles.
    1. RPLPredefinedFunctions.pdf -- RPL Predefined Functions
    2. RPLTypesPalette.pdf -- Palette Buttons

2. 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 expression types).
    3. Image file processing. This is just copying image files generated from FrameMaker to files with the required names; no actual modification to image data is involved.
    4. Generate new HTML output for each function, referring to our own CSS styles. (See Appendix B example).

3. Palette Button (Expression Type) Processing
    - Similar steps to previous item, but reusing some common code.

4. Add generation of Qt Resources index file (.qrc). This is used in the RiverWare build to bind the generated HTML and image files.

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

(B) RiverWare Development / Level 1: Access to existing RPL documentation

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 expression types.

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. Only links within the two relevant documents are supported. Links to RPL Predefined Functions bring up the function's editor dialog. Links to expression type help shows that document in the new Help Viewer (see task 5, below), scrolled to the appropriate section.

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 expression types: the 60 buttons). This includes a QWebView (similar to above). The only active component is a history combo box to jump to recently viewed topics.

6. In RplFrame, double clicking on a supported expression, show the HTML version of the RPL Data Types and Palette document in the Help Viewer, scrolled to the appropriate section.

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.

(C) Design Level 2 Development: Improvements to RPL documentation

1. Condense leading sentence of some RPL Predefined Functions with their formal Description text. In the course of this task, we may want to make limited revisions to accompanying function content.

2. Provide formal argument names to the arguments of the 191 RPL Predefined Functions. An analysis should be done in a couple passes such that a consistent and desirable naming convention is devised and documented. See the three proposed options for adding this information, in the "Design Level 2" section. Also, descriptive content should be rewritten to refer to the new formal parameter names (where desirable).

3. Enhance post-processor script to appropriately tag formal argument names.

4. Develop RiverWare mechanism to assign formal parameter names to PredefinedFunction objects, and to retrieve those names for those objects.

5. Display formal parameter names, in place of "arg0", "arg1", etc., in simple GUI contexts:
    - RPL Predefined Function editor dialog, in the Arguments.
    - RPL Palette's Predefined Functions tab, in the Arguments column of the function list.

6. Implement tooltip on relevant visual tokens in the RPL Frame, incorporating formal parameter names where relevant. Note that no tooltip was previously supported here.

7. Use formal argument names in relevant RPL diagnostics. This will involve a good amount of analysis and code changes, as certain lower-level utilities which generate diagnostic messages do not currently have access to the name and argument index of the relevant RPL Predefined Function. There would be value in this sort of support for only the most likely error conditions.

8. Completion testing and documentation revisions.

(6) Development Estimate

Estimate Revision: 4-9-2014, Phil.

Task	Est. Hours	Task Description
(A) RPL Documentation Processing (total: 48)		
A1	8	Initial Document Preparation / Semantic Styles (where needed)
A2	16	Predefined Function Topic Processing (additional work beyond experimental work already completed).
A3	16	Palette Button Topic Processing
A4	4	Generation of Qt Resources index file (.qrc)
A5	4	Process Documentation
(B) RiverWare Development (total: 84)		
B1	12	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	8	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
(C) Design Level 2 Development: Improvements to RPL documentation (total: 88)		
C1	8	Condense function leading sentence / Description text
C2	24	Provide formal argument names, revise related content accordingly
C3	4	Enhance post-processor script to appropriately tag formal argument names.
C4	8	Mechanism to assign formal parameter names to PredefinedFunction instances, and to retrieve those names for those functions.
C5	4	Display formal parameter names in simple GUI contexts
C6	8	Add tooltips to visual tokens in the RPL Frame, use formal parameter names.
C7	24 ???	Use formal argument names in relevant RPL diagnostics.
C8	8	Completion testing and documentation revisions
	220	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 Expression Types / 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] (2 pages).
- 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) ---