ePublisher

ePublisher Platform

Documentation

Published date: 12/16/2023

webworks®

Technical Communication Software

Table of Contents

Advanced Format and Target Customizations

By defining online navigation, setting options and preferences in your project, and embedding additional information in your source documents, you can define the behavior and appearance of your online content. The following sections provide some additional information about techniques you can use to further refine and customize your design.

Note: These techniques are supported in all formats unless otherwise noted.

Understanding Customized Processing

ePublisher uses file and folder locations to provide a structure where you can create and store override files. These files customize how ePublisher transforms your content.

When you generate output for an output format, ePublisher uses the following process to identify the files to use to process your content and complete the task:

1. ePublisher checks the `Targets` folder hierarchy in your project for the files required to complete the task. ePublisher checks the folder hierarchy named for the target you are generating.
2. If the files are not found in the `Targets` folder hierarchy in your project, ePublisher checks the `Formats` folder hierarchy in your project for the files required to complete the task. ePublisher checks the folder hierarchy named for the output format of the target you are generating.
3. If the files are not found in your project folder hierarchy, ePublisher checks the installation folders.

This process allows you to override any default file in the installation folder hierarchy for one or more output formats or targets by placing a customized file with the same name at the correct location in the project folder hierarchy.

By recreating the file path in the `Targets` folder in your project, and storing a modified file in the correct location, you can change the processing for that specific target. This method allows you customize a specific target in a project that has multiple targets using the same output format.

By recreating the file path in the `Formats` folder in your project, and storing a modified file in the correct location, you can change the processing for all targets that use the same output format. This method is also helpful for projects with one target.

Note: Do not modify files in the installation folders. Store customized files only in the project folder hierarchy. You need to create the `Targets` and `Formats` folder hierarchies in your project folder, as needed for the files you want to override.

For more information, see “Stationery, Projects, and Overrides” and “ePublisher Pipeline and Transforms”.

Format and Target Overrides

ePublisher gives you complete control over the final output. However, some project modifications cannot be made through the ePublisher console. In these instances, you may need to override the default XSL files used by ePublisher to achieve the results you need.

When ePublisher generates output for the first time for a project, ePublisher reviews the `format.wwfmt` file in the folder for the appropriate format. This file tracks the actions required to generate the desired format. If there are no user customizations, all the files referenced by the `format.wwfmt` file are in the Format or Transforms folders in the installation folder hierarchy. By default, ePublisher first checks the project folder hierarchy for each required file before getting the files from the installation folder. With this process, you can override one or more default files by creating the same folder hierarchy in a project folder and storing a customized file with the same name at the correct location in the project folder hierarchy.

Note: You can override default files for all targets of a specific output format type in the project. You can also override default files for a specific target in a project without affecting other targets of the same output format type in that project. Overrides for a target override any customizations you specified in the override folders for a format.

Creating Format Overrides

This section defines how to override a default file for all targets of a specific output format type in the project. Do not modify the files in the installation folder hierarchy. These files are the default files and should remain as is. These files are also overwritten when you install new ePublisher releases. Instead, store and incorporate your customized files with your Stationery so your projects based on the Stationery get those customizations.

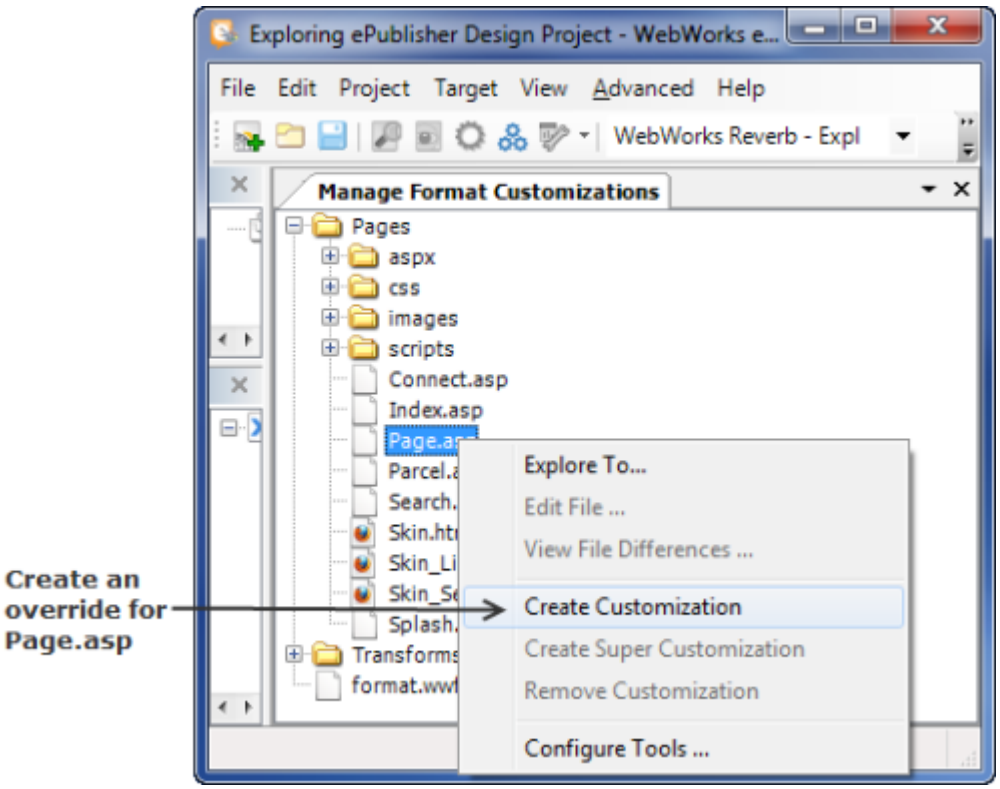
Note: Changes to `.xsl` and Format Trait Info `.fti` files are considered advanced customizations and are not supported by the Quadralay Product Support department. If you are familiar with XSL, you may find that customizations to `.xsl` and `.fti` files are a powerful way to achieve very specific results in your generated output or output deployment. However, you are responsible for maintaining any `.xsl` or `.fti` file overrides that you implement. The Quadralay Product Support department does not provide support for these advanced customizations.

File names used in ePublisher are case sensitive. Make sure the file and folder names you create exactly match the default file and folder names in the installation folder hierarchy. Do not copy any files into the project folder hierarchy that you are not overriding.

The following task assumes that ePublisher is installed in the default location and your ePublisher projects are stored in the `My Documents` folder. If you installed ePublisher to a non-default location, or if you store your projects in a folder other than the `My Documents` folder, adjust the paths in the following task.

To override a project format

1. In your Stationery design project, on the **Advanced** menu, click **Manage Format Customizations**.
2. In the displayed collection of folders and files, navigate to the file that you wish to override and highlight it.
3. To create an override for the selected file, right click and select the **Create Customization** menu. This will create an exact copy of the original file and place it into the appropriate location within the project directory. Now you can safely modify this file as desired as well as viewing the file differences between your override and the original.



4. Use either of the right-click mouse menu items: **Explore To...** or **Edit File ...** to access and modify your file override.

The next time you generate your project, the modified file automatically overrides the default file and the changes you made are incorporated into the output for all targets that produce that output format.

Note: When you save the ePublisher project as Stationery, the project format overrides you have created are saved with your Stationery. This Stationery can then be used to create future projects in ePublisher Express and ePublisher AutoMap.

Creating Target Overrides

This section defines how to override default files for a specific target in a project without affecting other targets of the same output format type in that project. Do not modify the files in the installation folder hierarchy. These files are the default files and should remain as is. These files are also overwritten when you install new ePublisher releases. Instead, store and incorporate your customized files with your Stationery so your projects based on the Stationery get those customizations.

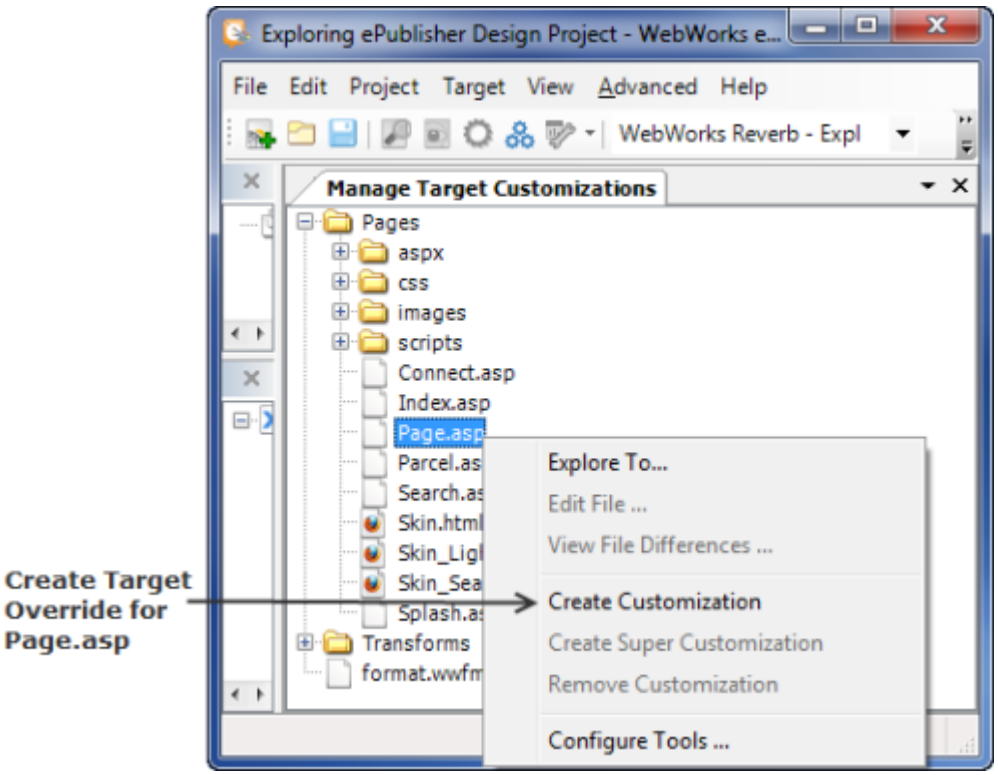
Note: Changes to `.xsl` and Format Trait Info `.fti` files are considered advanced customizations and are not supported by the Quadralay Product Support department. If you are familiar with XSL, you may find that customizations to `.xsl` and `.fti` files are a powerful way to achieve very specific results in your generated output or output deployment. However, you are responsible for maintaining any `.xsl` or `.fti` file overrides that you implement. The Quadralay Product Support department does not provide support for these advanced customizations.

Make sure the file and folder names you create exactly match the default file and folder names in the installation folder hierarchy. Do not copy any files into the project folder hierarchy that you are not overriding.

The following task assumes that ePublisher is installed in the default location and your ePublisher projects are stored in the `My Documents` folder. If you installed ePublisher to a non-default location, or if you store your projects in a folder other than the `My Documents` folder, adjust the paths in the following task.

To override a project target

1. In your Stationery design project, on the **Advanced** menu, click **Manage Target Customizations**.
2. In the displayed collection of folders and files, navigate to the file that you wish to override and highlight it.
3. To create an override for the selected file, right click and select the **Create Customization** menu. This will create an exact copy of the original file and place it into the appropriate location within the project directory. Now you can safely modify this file as desired as well as viewing the file differences between your override and the original.



4. Use either of the right-click mouse menu items: **Explore To...** or **Edit File ...** to access and modify your file override.

The next time you generate your project, the modified file automatically overrides the default file and the changes you made are incorporated into the output for that target. These customizations also override any override files you saved in the project folder hierarchy for the associated output format.

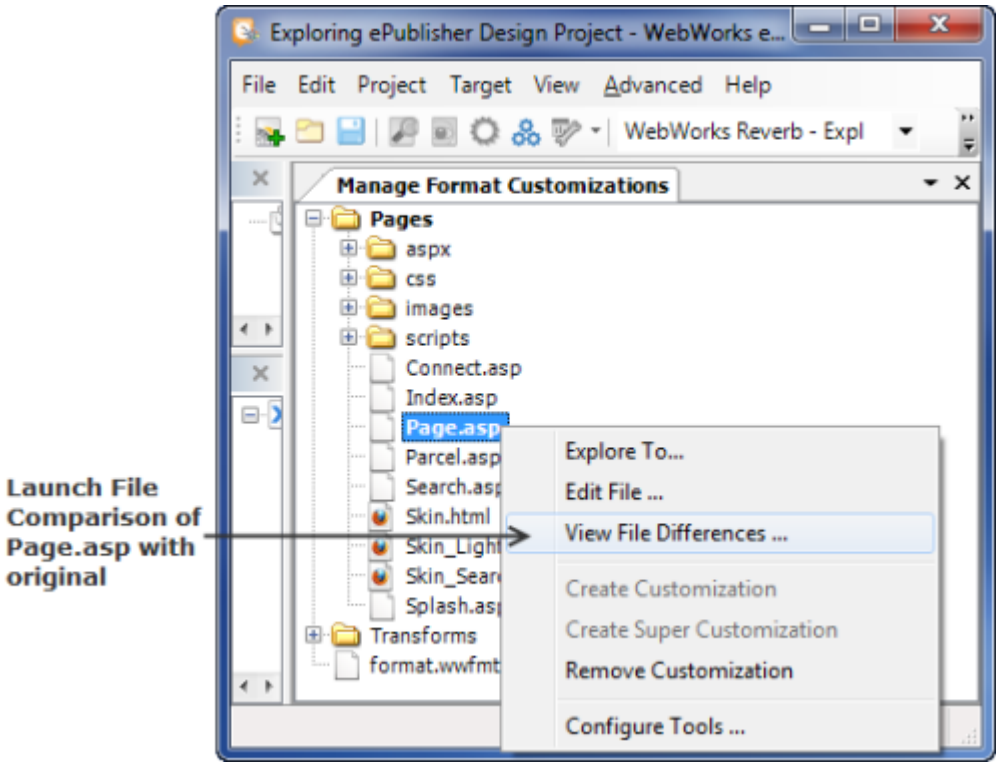
Note: When you save the ePublisher project as Stationery, the project target overrides you have created are saved with your Stationery. This Stationery can then be used to create future projects in ePublisher Express.

Managing Overrides

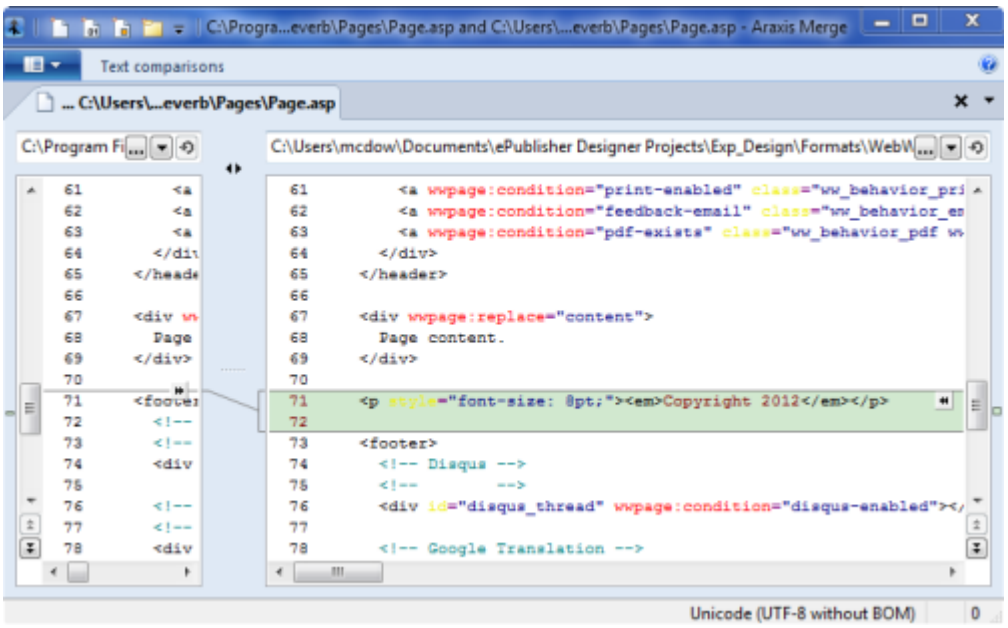
ePublisher assists you in the management of your project's overrides by graphically highlighting files that have been overridden as well as highlighting folders that contain files that have been overridden. In addition, you can also have ePublisher launch a 3rd-party *Diff* (also called a line-comparison) tool that will intuitively display what modifications have been made in your overridden file.

To View File Differences

1. In your Stationery design project, on the **Advanced** menu, click **Manage Format Customizations** (or Target for target overrides).
2. In the displayed collection of folders and files, navigate to the overridden file that you wish to compare and highlight it.
3. To compare the differences of the selected override, right click and select the **View File Differences...** menu. This will launch your configured *Diff* program and automatically display the differences between your file and the original.



4. You can examine how your override is different from the original. In addition, you can modify your overridden file directly from within the *Diff* tool.


C:\Progra...everb\Pages\Page.asp and C:\Users\...everb\Pages\Page.asp - Araxis Merge
Text comparisons
... C:\Users\...everb\Pages\Page.asp
C:\Program Fi
C:\Users\mcdow\Documents\ePublisher Designer Projects\Exp_Design\Formats\Web\
61 <a wwpage:condition="print-enabled" class="ww_behavior_pri
62 <a wwpage:condition="feedback-email" class="ww_behavior_es
63 <a wwpage:condition="pdf-exists" class="ww_behavior_pdf ws
64
65 </header>
66
67
68 Page content.
69
70
71 Copyright 2012
72
73 <footer>
74 <!-- Disqus -->
75 <!-- -->
76
77
78 <!-- Google Translation -->
Unicode (UTF-8 without BOM)
0

Customizing Page Templates (*.asp)

You can customize the appearance of your content page to meet your specific needs. ePublisher provides several customization options, such as where to display company information, browse buttons, and breadcrumbs. These options allow you to achieve a professional result in your generated output.

You may want to further customize your content page design. ePublisher provides this flexibility by allowing you to override the `Page.asp` file, which defines the appearance and behavior of your content page. However, if you customize this file, some customization options in the ePublisher console may no longer affect your content page design.

For example, you can override the `Page.asp` file to add a graphic bar across the top of the page with the product logo and name. You can also add a graphic bar across the bottom of the page with a copyright and a link to your company Web site and customer support area. You can also modify the formatting and layout of the default page design options, such as the company information options.

To override the Page.asp file

1. In your Stationery design project, on the **View** menu, click **Project Directory**. For more information about override files and locations, see “Stationery, Projects, and Overrides”.
2. ***If you want to override the processing for an output format***, create the `Formats\`*`formattype`*`\Pages` folder in your project folder, where *`formattype`* is the name of the output format you want to override, such as `Dynamic HTML`.
3. ***If you want to override the processing for a target***, create the `Targets\`*`targetname`*`\Pages` folder in your project folder, where *`targetname`* is the name of the target you want to override.
4. Copy the `Page.asp` file from the following folder to the override folder you created within your project folder:

   `Program Files\WebWorks\ePublisher Pro\Formats\`*`formattype`*`\Pages`
5. Open the `Page.asp` file you copied to your project override folder in a text editor.
6. Modify the `Page.asp` file as needed.
7. Save and close the `Page.asp` file.

Page Templates Reference

Page templates are files that form the skeleton structure of generated output files. Often users will create overrides for these files so that they can precisely control the look and function of generated output files. The most common type of page templates to customize are the `Page.asp` and `Splash.asp` files, which are used in most all of the output format types.

Namespace and Attributes

ePublisher defines the namespace: `wwpage` for page templates as follows:

```
xmlns:wwpage="urn:WebWorks-Page-Template-Schema"
```

The following shows the page template attributes that can be used along with their purpose.

Attribute	Purpose
`wwpage:condition`	Controls when elements should be preserved in the generated output.
`wwpage:content`	Use this attribute to replace the `innerHTML` or `innerXML` of an element.
`wwpage:replace`	Use this attribute to replace the `innerHTML`/`innerXML` as well as the `outerHTML`/`outerXML`.
`wwpage:attribute-<name>`	Use this attribute to emit a generated attribute value with name: `name`. In addition, you can specify a value of either: `relative-to-output` or `copy-relative-to-output` or `wwsetting:<target-setting-name>`. Specifying the value using `relative-to-output` is useful for file type attributes that require a pathname relative to the parent file. If the `copy` version is used, it additionally copies the file into the correct location within the generated output directory.
`wwpage:NoBreak`	Use this attribute to collapse all white space between two elements into nothing.
`wwpage:Import`	Use this attribute to indicate an that an import operation will take place using one of the `*-from-*` `wwpage` attributes.
`wwpage:content-from-file`	Use this attribute to import the contents of the specified file as the `innerHTML` of an element.
`wwpage:replace-from-file`	Use this attribute to import the contents of the specified file as the `innerHTML` as well as the `outerHTML` of an element.
`wwpage:content-from-lookup`	Use this attribute to import the contents of the specified item name generated by ePublisher as the `innerHTML` of an element.

Attribute	Purpose
`wwpage:replace-from-lookup`	Use this attribute to import the contents of the specified item name generated by ePublisher as the `innerHTML` as well as the `outerHTML` of an element.

Note: If you are using `wwpage:attribute-<name>` to express attribute that uses a namespace of its own, you can use a "-" character in place of the ":" character. For example: `wwpage:attribute-xml-lang` is used to generate the xml:lang attribute in the output.

The following shows a typical usage of `wwpage` attributes within a `Page.asp` or `Splash.asp` page template.

Attribute	Example usage in Page.asp page template
`wwpage:condition`	`<hr wwpage:condition="header-exists" align="left" />`
`wwpage:content`	`<title wwpage:content="title">Title</title>`
`wwpage:replace`	`<div wwpage:replace="content">Page content.</div>`
`wwpage:attribute-<name>`	`<link rel="StyleSheet" href="css/print.css" wwpage:attribute-href="copy-relative-to-output" type="text/css" media="print" />`

`wwpage:NoBreak`	`<a href="#">`
	
	`<wwpage:NoBreak />`
	
	`<img src="myimage.png">`
	
	`</a>`

`wwpage:Import`	`<wwpage:Import wwpage:replace-from-file="scripts/common.js" />`
`wwpage:content-from-file`	`<div wwpage:Import wwpage:content-from-file="scripts/common.js"></div>`
`wwpage:replace-from-file`	`<wwpage:Import wwpage:replace-from-file="css/webworks.css" />`
`wwpage:content-from-lookup`	`<div wwpage:Import wwpage:content-from-lookup="catalog-css"></div>`
`wwpage:replace-from-lookup`	`<wwpage:Import wwpage:replace-from-lookup="catalog-css" />`

Logical AND/OR/NOT in condition attributes

The `wwpage:condition` attribute supports both logical `AND` as well as `OR` to provide for multiple conditions in a single attribute expression. For a logical `AND`, use white space as a separator between conditions. For a logical `OR`, use a single comma as a separator between conditions. The following shows a few simple examples of using multiple conditions:

Logic Type	Example
AND (use space)	`<hr wwpage:condition="header-exists company-email-exists" />`
OR (use ",")	`<hr wwpage:condition="header-exists,footer-exists" />`
NOT (use "!")	`<hr wwpage:condition="!header-exists" />`
AND with OR and NOT	`<hr wwpage:condition="header-exists company-email-exists,footer-exists !company-email-exists" />`

Working with URL and File Paths in Page Templates

The `wwpage:attribute-<name> attribute can be used to specify several behaviors when working with URL and File paths. The following shows the list of available modifiers for this wwpage:attribute.`

Attribute Modifier	Example
relative-to-output	`<script type="text/javascript" src="scripts/common.js" wwpage:attribute-src="relative-to-output"></script>`
copy-relative-to-output	`<script type="text/javascript" src="scripts/common.js" wwpage:attribute-src="copy-relative-to-output"></script>`
relative-to-output-root	`<script type="text/javascript" src="scripts/common.js" wwpage:attribute-src="relative-to-output-root"></script>`
copy-relative-to-output-root	`<script type="text/javascript" src="scripts/common.js" wwpage:attribute-src="copy-relative-to-output-root"></script>`
absolute-to-output	`<external-graphic content-height="scale-to-fit" width="1in" height="1in" src="url('{wwformat:Pages/images/Logo.png}')" wwpage:attribute-src="absolute-to-output" />`

Attribute Modifier	Example
resolved-uri	`<external-graphic src="url({wwformat:Pages/images/rms_doc_logo.png})" wwpage:attribute-src="resolved-uri" content-width="100px" content-height="100px" left="0pt" top="0pt" />`
resolved-path	`<external-graphic src="{wwformat:Pages/images/rms_doc_logo.png}" wwpage:attribute-src="resolved-path" content-width="100px" content-height="100px" left="0pt" top="0pt" />`

Using Replacement Types ({}) in Page Templates

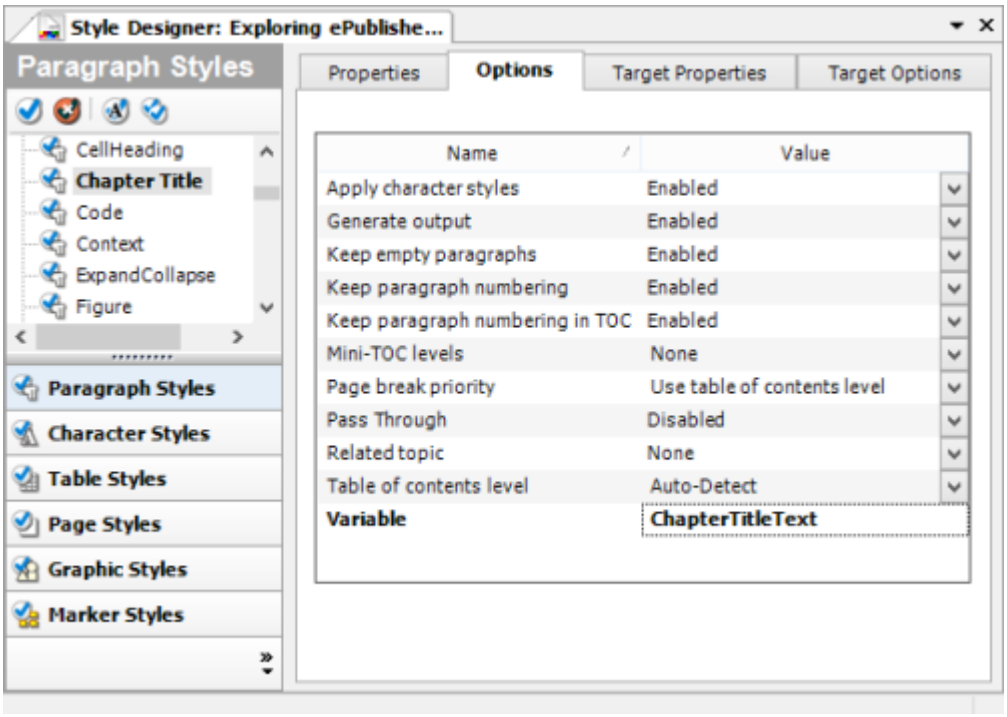
In page templates, you can use the "{}" characters to indicate a parameter to use in an attribute replacement. The "{}" characters are used to embed a path or parameter within an attribute value. For example, the following is a typical replacement for an HTML `img` tag, demonstrating how "{}" characters can be used to improve the result of the replacement.

```
<img src="url('{images/logo.png}')" wwpage:attribute-src="copy-relative-to-output" /
>.
```

Using ePublisher Style Variables in Page Templates

A powerful feature for capturing text in your source content based on their assigned style name is available by using the `wwvars:<Variable_Name>` in your page template file

To capture text from your source documents, you first need to configure the **Variable** style option within your ePublisher Designer project. The **Variable** style option is available for Paragraph, Character, and Marker styles.



In the ePublisher Style Designer, select the Variable option and assign it a value that will be used in your page template.

Once you have assigned a variable name to the style, any time ePublisher encounters that style it will set the value of that variable to the text of that style in the source content.

To access the variable in your page template, you would use the `wwvars:<Variable_Name>` attribute, similar to the following.

```
<div wwpage:content="wwvars:ChapterTitleText">

Chapter Title Appears Here

</div>
```

Using Markers in Page Templates

Another powerful feature for capturing text within your source content is through the use of markers. Within your authoring environment you can insert markers that have a name and value. Then in your page templates, you can access the value of these markers by their name. The page template will use the last encountered maker within the scope of the generated page of your output.

To implement this functionality in your page template file, use the syntax: `wwmarker:<Marker_Name>`.

For example, if you used a marker called: Reviewer the following is syntax that could be used in the page template file to capture the value.

```
<div wwpage:condition="wwmarker:Reviewer" wwpage:content="wwmarker:Reviewer">

Reviewer name appears here, if marker is available.

</div>
```

ePublisher Pipeline and Transforms

ePublisher provides the complete framework needed to solve an unlimited number of publishing issues. ePublisher provides a flexible processing model that allows you and third-party developers to customize the ePublisher workflow as needed. This workflow uses the open, standards-based XSL transformation approach to give you the flexibility and extensibility you need without locking your data in a proprietary format.

Terminology

To understand how ePublisher works, you should first understand some terminology. The following list defines several important product terms:

source document input formats

The types of source documents ePublisher processes, such as Markdown++, Adobe FrameMaker, Microsoft Word, and DITA-compliant XML files.

output formats

The types of output ePublisher can generate from your source documents. Each output format is a set of individual XSL transforms that process source documents to create the output. Quadralay provides many default output formats, such as WebWorks Reverb (1 & 2), WebWorks Help, HTML Help, Eclipse Help, Oracle Help, Sun JavaHelp, XML+XSL, Dynamic HTML, and many others. You can also create custom output formats. An output format is broken down into pipelines and stages.

format files

The files that define the output format and transforms. These files are stored in the `Formats` folder in the installation folder. You can override files to customize the transformation process and these override files are stored in a matching location within the project folder. When you save a Stationery, the override files are stored with the Stationery.

Stationery

A complete set of processing rules and styles that define all aspects of the output. The Stationery can define multiple targets, and each target can be the same or different output format. Writers use the Stationery created by the Stationery designer when they create projects and generate output. Writers can also override some default settings in their projects, such as variable values and conditions.

project

Identifies the Stationery to use, the source documents to process, and the output format and target settings to override when generating output. A project can include multiple targets.

target

Defines a deliverable for the project, based on an output format. A project can have multiple targets and each target can be the same or different output format. All targets in a project share the Style Designer properties and options, but each target has its own target settings, such as variable values and company information.

stage

The smallest discrete action possible in an output format, such as an XSL transform. ePublisher transforms source documents to a target by breaking the process into a series of steps, also known as stages. A stage is part of the transformation process that performs a specific action in the process. Stages are grouped into pipelines of related stages. In the future, non-XSL actions may also be possible. All stages define an output file type and zero or more input file types.

pipeline

A set of stages that are run in sequence. Pipelines can have dependencies on other pipelines, which can ensure that required input files are created before a pipeline runs.

Processing Workflow

A **Stationery designer** creates Stationery that identifies the supported output formats and defines the appearance and behavior of the output generated with projects based on that Stationery. A **writer** creates a project based on the Stationery and identifies the source documents to include in the project. Then, the writer uses the project to generate output for each target defined in the project. A **developer** can create custom behaviors and formats for Stationery designers to enable and use in the ePublisher Designer console.

ePublisher uses stages, pipelines, and the `format.wwfmt` file for each output format to structure and manage the processing workflow. A stage in a pipeline runs only once. The stage itself must determine the number of files to process and the number of files to emit. A stage can pass any message to another stage as long as it is emitted to a file first, and then the type and path of the file are emitted in the XML result.

ePublisher processes files as follows:

1. Apply conditions, cross-reference formats, and variables to source documents.
2. Export source documents to WIF.
3. Determine the execution order for format pipelines.
4. Select the next pipeline available for processing and execute all the stages defined in that pipeline.

To fill gaps left by XSL, ePublisher provides several XSL extension objects to support specific actions, such as processing image files, modifying the file system, and writing multiple documents from a single XSL transform.

Transformation Process

The first pipeline in the process prepares the source documents to be transformed by XSL. ePublisher uses XSL to transform documents to output formats. ePublisher also provides extensions to XSL to support image processing and other source document operations that otherwise would require additional technologies, such as FrameScript or VBA. ePublisher requires native access to source documents, a standard starting point to start XSL transformation, and a standard method for coordinating XSL transforms on files. This first pipeline applies conditions and variables, extracts native drawings and images, and exports the source documents to the WebWorks Intermediate Format (WIF), a WebWorks XML language that enables XSL to process the source documents in later stages.

When XSL processing begins, ePublisher processes files based on type rather than by name. Each stage defines an `.xsl` file that creates a portion of the output target. When every stage in every pipeline is finished, the transformation process is complete.

Using the transformation to WIF gives you the ultimate flexibility with multiple input and output format support. Each input format, such as Microsoft Word and Adobe FrameMaker, can be transformed to any and all output formats, such as Microsoft HTML Help and WebWorks Help. As new output formats become available, all input formats are automatically supported. In addition, a new input format automatically supports all output formats.

Adapters Transform Source Documents to WIF

To extract content from source documents, ePublisher defines an adapter interface. This adapter interface allows ePublisher to transparently support a variety of source documents, such as Microsoft Word, Adobe FrameMaker, and DITA.

When processing content, ePublisher is not aware of the type of adapter in use. ePublisher simply asks for an adapter that can process a source document, and then it sends requests to the adapter associated with that document type.

Adapters handle the following tasks:

1. Reporting book files.
2. Scanning for styles, conditions, cross-references, and variables.
3. Applying specified conditions, cross-reference formats, and variables to source documents.
4. Extracting native drawings and images.
5. Exporting source documents to WIF.

This final step is where the source document is effectively brought into the world of XML/XSL processing.

WebWorks Intermediate Format (WIF)

WIF is a Quadralay Corporation standard used as an intermediate format. Quadralay chose to establish WIF rather than use another XML format because other XML formats do not provide all the functionality needed to solve conversion-related issues. Other XML standards are great for authoring structured content, but they deliberately exclude formatting information and they are not designed to define source that lacks structure. To address many conversion-related issues, such as unstructured Microsoft Word source documents with one style (Normal) used throughout the documents and customized as needed, Quadralay needed a more flexible and powerful intermediate format.

The most flexible way to resolve these issues was to create WIF, which provides a stable XML schema that can grow and change as needed for future input and output format requirements. WIF is designed to address the following goals:

- Represent source documents in a standard schema for XSL processing.
- Preserve individual word processor features with high fidelity.
- Strongly favor XSL processing over ease of authoring.

Processing Files by Type

Once source documents are transformed into WIF, XSL processing can begin. The next question is about how to organize the XSL processing. ePublisher gives you the flexibility you need to define a workflow without the drudgery. While some processes and standards require intimate knowledge of an XSL transform's input and output *file names*, ePublisher uses knowledge of an XSL transform's input and output *file types*. This distinction is subtle but powerful.

Consider processing XSL with exact file names:

- `xslt doctoc.xsl alpha.xml > alpha_toc.xml`
- `xslt doctoc.xsl delta.xml > delta_toc.xml`
- `xslt grouptoc.xsl alpha_toc.xml delta_toc.xml > group_toc.xml`

Now consider using file types in place of file names:

- `xslt doctoc.xsl Source > Doc_TOC`
- `xslt grouptoc.xsl Doc_TOC > Group_TOC`

Using file types in place of file names simplifies the specification, where the file types are defined as follows:

File Type	File Names
Source	alpha.xml delta.xml
Doc_TOC	alpha_toc.xml delta_toc.xml
Group_TOC	group_toc.xml

With the file type approach, you need to specify `xslt doctoc.xsl` *Source* > *Doc_TOC* only once. This specification runs for every *Source* type document found and generates the corresponding *Doc_TOC* output files.

Identifying Files to Process

Processing files based on type rather than individual file names allows developers to define workflows for any number of input and output files. To identify the files each XSL transform should process or create, ePublisher defines an XML schema to store file information. A simplified form of this schema looks like the following example:

```
<Files>
 <File name="alpha.xml" type="Source" />
 <File name="delta.xml" type="Source" />
</Files>
```

This XML document is fed as input to each XSL transform, and every XSL transform emits the list of generated files using this same schema. Therefore, running `doctoc.xsl` with the previous example input file list returns the following output:

```
<Files>
 <File name="alpha_toc.xml" type="Doc_TOC" />
 <File name="delta_toc.xml" type="Doc_TOC" />
</Files>
```

In this example, the final transform, `grouptoc.xsl`, runs with the following input file list:

```
<Files>
 <File name="alpha.xml" type="Source" />
 <File name="delta.xml" type="Source" />
 <File name="alpha_toc.xml" type="Doc_TOC" />
 <File name="delta_toc.xml" type="Doc_TOC" />
</Files>
```

This final transform returns the following output:

```
<Files>
 <File name="group_toc.xml" type="Group_TOC" />
</Files>
```

The *`Source`* documents were fed into the `grouptoc.xsl` transform because all previous output files are available to all downstream XSL transforms. ePublisher allows developers to define as many XSL stages as needed, and each stage can process, reprocess, examine, and manage, any preceding file generated.

TOC Processing Example

The previous sections describe a TOC processing example to illustrate how ePublisher processes files. The following example shows the corresponding `format.wwfmt` file for this TOC processing example:

```
<Format>
<Pipeline name="TOC">
   <Depends pipeline="Locale" />
   <Stage type="xsl" action="doc_toc.xsl">
      <Parameter name="ParameterDependsType" value="Source" />
      <Parameter name="ParameterType" value="Doc_TOC" />
   </Stage>
   <Stage type="xsl" action="group_toc.xsl">
      <Parameter name="ParameterDependsType" value="Doc_TOC" />
      <Parameter name="ParameterType" value="Group_TOC" />
   </Stage>
</Pipeline>
</Format>
```

This representation defines the XSL transforms and file types to process.

Stationery, Projects, and Overrides

The following sections provide an overview of the ePublisher files used in the transform process. You can customize these files and store them in an override location to customize how ePublisher transforms your content. For more information about terms used in the following sections, see “Terminology”.

File Locations

ePublisher Designer allows you to create Stationery that writers can base their projects on. The Stationery stores the style settings and customized files used to define how ePublisher transforms your content. When writers create projects based on Stationery, their projects copy the customized files and settings from the Stationery and use those customized files when processing the content included in their projects. In this way, customized files in the projects override the default ePublisher files. The Stationery also allows you to quickly roll out changes to your customized processes and settings. Once you update your Stationery, writers are notified when they open a project based on the Stationery to synchronize with the Stationery, which incorporates your latest changes.

ePublisher uses file and folder locations to provide a structure where you can create and store override files. An **override file** is a customized file stored in a parallel location in a project or Stationery that is used to process the source documents instead of the default ePublisher file stored in the installation folders. These files customize how ePublisher transforms your content. ePublisher uses the following locations to store its files:

Your project folders

By default, each project is saved in the `My Documents\`*`componentname`*` Projects\`*`projectname`* folder, where *`componentname`* is the name of the ePublisher component used to create the project, such as `ePublisher Designer` or `ePublisher Express`, and *`projectname`* is the name of the project itself.

ePublisher installation folder

By default, ePublisher components are installed in the `Program Files\WebWorks` folder. The default transformation files are stored with ePublisher Designer in the `Program Files\WebWorks\ePublisher\`ePublisher Designer folder.

When you generate output for an output format, ePublisher first checks the project folders for the files required to complete the task. If the files are not found in the project folders, ePublisher checks the installation folders. This process allows you to override any default file in the installation folder hierarchy for one or more output formats by placing a customized file with the same name at the correct location in the project folder hierarchy.

Note: Do not modify files in the installation folders. Store customized files only in the project folder hierarchy.

The files used to transform a source document are located in several main folders in the ePublisher installation folder hierarchy:

Formats

Contains format-specific XSL files. The default files that you can customize and store in your project folder hierarchy are stored in this folder. The `Formats\Shared` folder contains XSL files used by multiple formats.

Helpers

Contains command-line programs to perform specific actions, such as compiling HTML Help `.chm` files or generating WebWorks Help search indices.

File Processing

ePublisher divides the transform process into a series of pipelines, or groups of similar stages. Each stage defines an `.xsl` file to run that creates a portion of the target. ePublisher uses a combination of the format pipelines, the `files.info` GlobalFiles record file, XSLT parameters, and the root match template in a given XSL file to perform the action identified in a stage. These stages are defined in the `format.wwfmt` file located in each format folder. The `format.wwfmt` file defines all the pipelines and stages necessary to create a specific output format, and the relationships that each of these pipelines and stages have to one another. There is no preconceived order in which pipelines run. ePublisher calculates at run time the order in which each pipeline runs, based on pipeline dependencies.

ePublisher processes files based on type rather than by name. Each stage defines an `.xsl` file to run that creates a portion of the target. All ePublisher XSL style sheets define the following global parameters by default:

Parameter	Description
GlobalFiles	Provides the `files.info` file for the project, which includes all previously generated files in the files XML schema.
GlobalProject	Provides the `.wep` project file, which defines the project as XML.
GlobalPipelineName	Specifies the name of the pipeline in which the current stage is defined.
GlobalInput	Identifies the files selected in Document Manager. This parameter provides all previously generated files derived from the `Generate Selected` command in the files XML schema.

In addition, ePublisher passes the XSL style sheets all parameters defined for the current stage in the `format.wwfmt` file. Parameters passed to each XSLT file are usually used to load the various XML node sets needed for the current stage. For example, a stage may need to take information recorded in the **Behaviors** pipeline and merge it with information in the **Links** pipeline. The location of information generated in each stage is stored in the `files.info` file. Future stages can use the `files.info` file to learn the location of previously generated information needed to complete that particular stage. When that stage finishes, it notifies the `files.info` file that the information it created is available for future stages to use, if necessary.

Processing files based on type rather than by individual names allows you to define workflows for any number of input and output files. This flexibility allows you to define as many XSL stages as you need, and each stage can use any preceding files generated. You can create or delete pipelines, add or delete stages, or insert elements that dictate when to run a stage or pipeline.

ePublisher File Types

This section provides specific information about the files most commonly used to customize ePublisher and its transformation processes.

Format Trait Info (*.fti) Files

Format Trait Info files have the `.fti` file extension. Format Trait Info files are located in both the ePublisher `Formats\Shared` folders, and the `Format` folders of each target.

In every project, the user has a series of format and style options to customize the appearance of his output. In the ePublisher consoles, these options are accessed and manipulated through target settings and Style Designer. For ePublisher, these customization options are defined in the Format Trait Info files. Manipulating these files allows you to customize the location of options, create or delete options and parameters, adjust their valid values, or choose new default values.

For ePublisher to process a Format Trait Info file, the `.fti` file must have the same file name, not including the `.fti` file extension, as an XSL file in the current format folder hierarchy. For example, ePublisher processes the table of contents using the toc.fti and toc.xsl files in the same folder. The information displayed in ePublisher represents all Format Trait Info files associated with a specific format. Two or more Format Trait Info files may define the same <Setting /> element, but the ePublisher console displays that setting only once.

When you generate output, ePublisher reviews the settings specified in target settings and Style Designer, stores the settings in the `.wep` project file, and incorporates the settings in the generated output.

format.wwfmt Files

A `format.wwfmt` file is stored in each of the specific format folders. The `format.wwfmt` file defines all the stages required to create output, and the relationships between each of these stages. These stages are grouped in pipelines of related stages. When every stage of every pipeline has finished, ePublisher is done generating the output. There is no preconceived order in which pipelines run. ePublisher calculates at run time the order in which each pipeline runs. You can create or delete pipelines, add or delete stages, or insert a <Depends /> element that dictates when a stage or pipeline runs.

files.info Files

ePublisher creates and stores the `files.info` file for each target with the project files. Each stage completes a small portion of the transformation. A completed stage creates new information that contributes to the appearance or functionality of the transformed content. When a stage finishes, ePublisher writes the location of the processed information in the `files.info` file for use by other stages. Each stage can use information in the `files.info` file and record information in the `files.info` file for use by another stage.

Stationery Design Project .wep File

The Stationery design project `.wep` file contains all the configuration information required to define the Stationery and generate output. This file includes the sample source document names, settings, options, and customizations. You can use ePublisher Designer to modify a Stationery design project, and to save it as Stationery.

Project .wrp File

The project `.wrp` file contains all the information required to generate output. This file includes all the source document names, settings, options, and customizations.

Stationery .wxsp File

A Stationery `.wxsp` file is based on settings and options defined in a Stationery design project and saved as Stationery. By saving a project as Stationery, all the settings and customizations that you captured in your project, including project format overrides, are automatically implemented in any new projects based on the Stationery.

XSL Match Templates

Style sheets are made of a number of templates, each of which defines what the XSLT processor should do when it matches a particular node in the XML source document. The XSLT processor populates the result document by instantiating a sequence of templates. Instantiation of a template means that the XSLT processor performs the following tasks:

- Copies any literal data from the template to the target
- Executes the XSLT instructions in the template

Templates are defined using the `<xsl: template>` element. The `match` attribute in the `<xsl:template>` element indicates which parts of the source document should be processed with the particular template.

Root Match Templates

Each XML document has a single root element. This element encloses all the following elements and is therefore the parent element to all the other elements. When the XSLT processor applies a style sheet to an XML document, it begins processing with the root element of the XML source document. To process the root element, the XSLT processor searches the style sheet for a template rule that matches the root element. A template rule matches the root element when the value of the template `match` attribute is `/` (slash).

If the user explicitly defines a template rule that matches the root element, the XSLT processor finds it and applies that template to the entire XML document. If the XSLT processor does not find an explicitly defined template rule that matches the root element, the processor implements the default template that matches the root element. Every style sheet includes this default template.

Root Match Templates in ePublisher

The root match template has a number of responsibilities in ePublisher:

- Recording files and dependencies
- Loading node sets
- Progress recording for the extension object that lives in the `wwprogress` namespace.
- Up-to-date checking on the projects output files.
- Writing output

Extension Objects

While extremely powerful, XSL has shortcomings when used to perform system-level scripting tasks. Some operations, such as working with files and advanced string operations, are not included in the standard XSL language.

XSL provides a generic extension mechanism to add new features to the base language. ePublisher provides a standard set of extension objects that enable XSL to generate all the required formats.

XSL extensions live in a specific namespace. You can define your own prefix to associate with a given namespace, but using a consistent naming convention makes life easier.

Creating Super Overrides

An XSL super override is simply a way for the stationery designer to add an XSL override to a file in the Transforms directory without having to copy the entire template structure of the file.

> **Note:** Support XSL overrides is only available through Study Hall and not Standard Support. For more information regarding Study Hall, please go to http://www.webworks.com/eschool/study_hall/.

Here is an example from the WebWorks wiki of `XSL` that you can use as a starting point for learning about this feature:

```
<!-- Override the mode wwdoc:Table from the base content.xsl -->
 <!--                                                          -->
 <xsl:template match="wwdoc:Table" mode="wwmode:content">
  <xsl:param name="ParamTable" select="." />
  <!-- Parameters passed in to this processing context.  We need to pass them along.
 -->
  <!--
 -->
  <xsl:param name="ParamSplits" />
  <xsl:param name="ParamCargo" />
  <xsl:param name="ParamLinks" />
  <xsl:param name="ParamTOCData" />
  <xsl:param name="ParamSplit" />

  <!-- Manually set all table formatting in this template. -->
  <!--                                                      -->
  <html:table cellspacing="0" cellpadding="5" border="1" style="margin-top: 0.6em;
 margin-bottom: 0.6em;">
   <xsl:for-each select="$ParamTable/wwdoc:Caption/wwdoc:Paragraph[1] | $ParamTable/
preceding-sibling::wwdoc:Paragraph[@stylename = 'Caption'][1]">
    <html:caption>
     <html:p>
      <html:b>
```

```
      <xsl:for-each select="./wwdoc:TextRun/wwdoc:Text">
       <xsl:value-of select="@value" />
      </xsl:for-each>
     </html:b>
    </html:p>
   </html:caption>
  </xsl:for-each>

  <xsl:for-each select="$ParamTable/wwdoc:TableHead | $ParamTable/wwdoc:TableBody |
$ParamTable/wwdoc:TableFoot">
   <xsl:variable name="VarSection" select="." />

   <xsl:variable name="VarTagName">
    <xsl:choose>
     <xsl:when test="local-name($VarSection) = 'TableHead'">
      <xsl:text>thead</xsl:text>
     </xsl:when>
     <xsl:when test="local-name($VarSection) = 'TableBody'">
      <xsl:text>tbody</xsl:text>
     </xsl:when>
     <xsl:when test="local-name($VarSection) = 'TableFoot'">
      <xsl:text>tfoot</xsl:text>
     </xsl:when>
    </xsl:choose>
   </xsl:variable>

   <xsl:element name="{$VarTagName}" namespace="'http://www.w3.org/1999/xhtml'">
    <xsl:for-each select="$VarSection/wwdoc:TableRow">
     <xsl:variable name="VarRow" select="." />

     <html:tr>
      <xsl:for-each select="$VarRow/wwdoc:TableCell">
       <xsl:variable name="VarCell" select="." />

       <html:td>
        <!-- Pass control back to the base processing flow. -->
        <!--                                                 -->
        <xsl:apply-templates select="$VarCell/wwdoc:Paragraph"
mode="wwmode:content">
         <xsl:with-param name="ParamSplits" select="$ParamSplits" />
         <xsl:with-param name="ParamCargo" select="$ParamCargo" />
         <xsl:with-param name="ParamLinks" select="$ParamLinks" />
         <xsl:with-param name="ParamTOCData" select="$ParamTOCData" />
         <xsl:with-param name="ParamSplit" select="$ParamSplit" />
```

```
          </xsl:apply-templates>
         </html:td>
        </xsl:for-each>
       </html:tr>
      </xsl:for-each>
     </xsl:element>
    </xsl:for-each>
   </html:table>
  </xsl:template>
```

To use the super override

1. In your Stationery design project, on the **Advanced** menu, click **Manage Format Customizations** (or Target for target overrides)
2. In the displayed collection of folders and files navigate to the **Transforms** directory, and click on the XSL file you wish to modify
3. Right click the XSL file, and click Create Super Customization
4. In the text editor, you will see the XSL that indicates:

```
<!-- Write templates here -->

<!-- -->
```

5. Add the custom XSL, please note that the code will have to reference the original XSL file in order for ePublisher to process it correctly. For more information, please refer to this wiki page.
6. Save XSL override, save ePublisher project, reopen and generate output for ePublisher to process this override

Introduction

Welcome to the Quadralay's Developer's Guide for WebWorks ePublisher Designer. This document serves as an overview of how ePublisher Designer works and provides details on how the user can make changes to files to customize both the process of converting a document, and the document's final appearance.

Audience

This document is aimed at users of ePublisher Designer who want to make specific customizations to their projects. Users should have at least basic knowledge of XML, XSLT, and how the two interact. This guide presents a brief overview of XML and XSL. If the concepts there present confusion to the user, the references section provides resources for learning critical features of the languages.

Help

User changes to ePublisher Designer files are not supported through Quadralay's Product Support department. Modification of files is only supported through Quadralay's Services department. Modifying files in the installation directories is not recommended. Files should be modified in a project itself or in user-created formats only.

Conventions

This Reference Guide uses the following conventions.

Formatting

Bold:

File names are listed in bold

Code:

```
<Classes>
 <Class name="boolean">
  <Item value="true" stringid="boolean-true" />
  <Item value="false" stringid="boolean-false" />
 </Class>
 <Class name="color-name">
  <Item value="transparent" />
  <Item value="aqua" />
  <Item value="black" />
  <Item value="white" />
  <Item value="yellow" />
 </Class>
</Classes>
```

Highlights markup text used in XML.

Terminology

GUI:

A graphical user interface is a method for issuing commands to a computer through direct manipulation of graphical images and widgets in addition to text. The buttons and menus used in Microsoft Word are an example of a GUI.

Output Target:

The chosen format to which a source document will be transformed. There may be more than one output target.

Source Document:

The original document, created in Word or Framemaker, that will be transformed by ePublisher Designer to an output target.

Transform:

The process of converting a source document into a new chosen format.

Organization

This book is separated into seven parts:

Introduction

Architecture Overview:

Details regarding how ePublisher Designer works

XSL Match Templates:

ePublisher Designer's unique approach to template rule matching

Extension Objects:

Details on extending the functionality of XSL with Microsoft and ePublisher Designer extension objects

File Reference:

Details on files the user may modify

Project Format Overrides:

How to get started

Appendix:

Extension Objects: General, Microsoft, and ePublisher Designer extensions

About XML and XSL

XML is a meta-language. That is, it is a language used to create other markup languages. XML provides a basic structure and a set of rules to which any markup language must adhere. XML is based on three basic building blocks:

- elements
- attributes
- values

Elements describe or contain a piece of information and form the basis of all XML documents. Elements take the form of tags, as in HTML. Attributes are pieces of descriptive information that appear within an elements opening tag. An attribute consists of an attribute name and a corresponding value, separated by an equal symbol (=). The values of an attribute appear to the right of the equal symbol and must appear within quotes.

Together, a group of elements, attributes, and values make up an XML document.

XML allows users to create elements, attributes and values. There are no fixed elements as in HTML. In HTML `<table>` means only information grouped together in rows and columns. In XML, an element with the name `<table>` could refer to an HTML type table, or a piece of furniture, or whatever the author would like. In order to bring order to these seemingly randomly-named elements, XML needs a document that explains what each of these building blocks mean. The document that defines these building blocks is called an XSL style sheet. A rough comparison can be made to Custom Style Sheets, used in HTML.

ePublisher uses two different aspects of XSL to generate an output:

XSLT:

XSLT describes how to transform a source document from one markup language to another.

XPath:

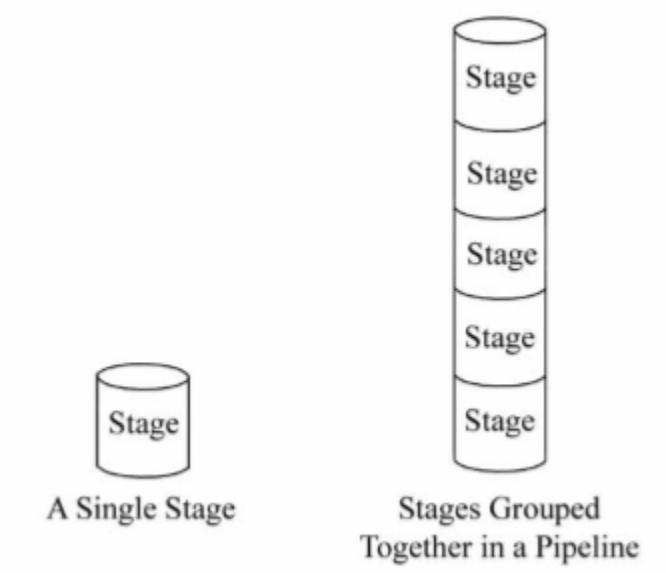
XPath is used by XSLT to select parts of XML to process and perform calculations.

An XSLT processor executes a stylesheet to give the user a particular result. In the transformation process, XPath defines parts of the source document for XSLT to match against one or more predefined templates. When a match is found, XSLT will transform the matching part of the source document into the result document.

Architectural Overview

Real World Example

ePublisher Designer converts a source document to an output target by breaking the process into a series of steps, or “stages.” Each stage performs a specific action in the process. These steps are grouped in “pipelines” of related stages. See Figure 1 for an illustration.



The first pipeline in the process prepares the source document to be converted by XSL. ePublisher Designer uses XSL to transform documents to target formats. XSL, however, cannot extract XML from Word or Framemaker documents, nor can it render images. This first stage applies conditions and variables, extracts native drawings and images, and exports the source document to WIF (WebWorks Intermediate Format), a WebWorks proprietary XML language that enables XSL to process the source document in later stages.

Once source documents are transformed into WIF, XSL processing can begin. ePublisher Designer processes files based on type rather than by name. Each stage defines an .xsl file to be executed which creates a portion of the output target. When every stage in every pipeline has been executed, the transform is complete.

Real World Example

For the sake of demonstrating the ePublisher Designer transform process let us assume a fictional pizzeria in New York City. Further, we'll say that a bag of pizza ingredients in the refrigerator is our source document. While that bag of ingredients is perfectly nice, it isn't particularly useful to anyone. The chef needs to transform the ingredients into something useful to him, a cooked pizza. In ePublisher Designer, the ingredients are our source document, the pizza oven is XSL, and the cooked pizza is our output target.

Unfortunately, it isn't possible simply to throw the ingredients into the oven and then remove a pizza ten minutes later. The bag of ingredients must first be prepared so the oven can deal with the ingredients in a way that is useful to us. In ePublisher Designer, the process of rolling out the dough, and spreading the sauce and cheese is the first stage where a source document is prepared for XSL (the oven) to do it's job.

Experienced chefs understand that the cooking process is a series of chemical reactions of food to heat. The dough becomes crisp, the cheese melts, etc. The process creates what any reasonable person would define as a pizza. In ePublisher Designer, XSL (the oven) is applying a series of steps (cooking) that are changing our source document (uncooked pizza) into the output target (something the customer is willing to consume).

Just as the pizza chef can customize the pizza ingredients any number of ways, such as cooking the pizza longer or at a different temperature to get different results, the ePublisher Designer user may customize the XSL process to affect his source document.

File Reference

This section provides basic information on the ePublisher Designer files used in the transform process. This is where the user can make changes to the process to customize his output. In our real life example, this is where the chef can make his crust extra crisp, or add pepperoni or anchovies.

File Locations

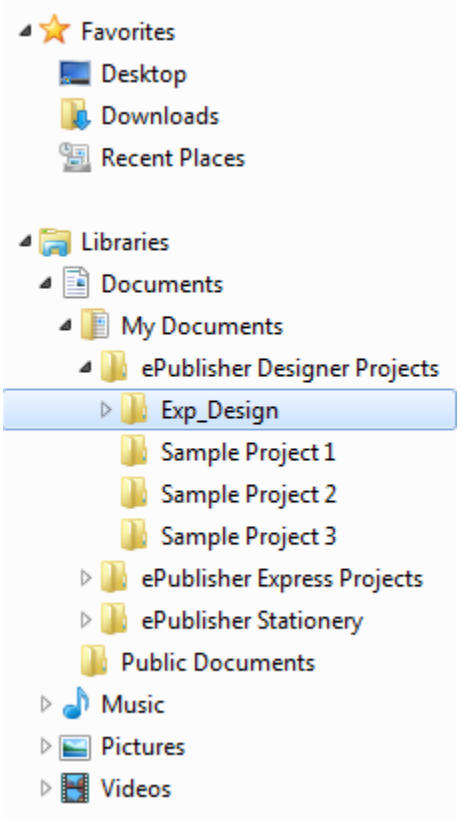
There are two locations to keep in mind when planning to modify a file:

ePublisher Designer installation directory:

By default, ePublisher Designer is installed in C:\Program Files\WebWorks\ePublisher Designer.

Project directory:

By default, a project is saved in “My Documents > ePublisher Designer Projects” in a directory with the same name as the project itself. See Figure 2.



When output is generated for a given output format, the ePublisher Designer engine first looks in the Project directory for the files required to complete its task. If the necessary files are not found in the Project directory, the engine will look in the installation directory. This means that any file contained in the installation directory hierarchy may be overridden in a given format by placing a file with the same name at the correct location in the Project directory. See the procedure detailed in “Creating Project Format Overrides” for more information.

Note: Modifying files in the installation directories is not recommended. Files should be modified in the project itself only.

The files used to transform a source document are located in three main folders in the ePublisher Designer installation directory:

Transforms:

This folder contains XSL documents which are used by all formats

Formats:

This folder contains format-specific XSL documents. Files to be modified by the user are located here.

Helpers:

This folder contains command line programs to perform some action (i.e., compiling .CHMs or generating WWHelp search indices)

Real World Example

Let us revisit our pizzeria in New York City. For the sake of this example, let's pretend that once the chef makes a pizza, he completely forgets the process of how to do it. Luckily, the owner has placed the directions on the wall where the chef rolls out the dough. The chef quickly learns to check the wall to see what he is supposed to do next.

Now, sometimes, the customer wants thin crust. Or extra cheese. The instructions on the wall only tell the chef how to make one kind of crust, and only a certain amount of cheese. In order to make sure the chef makes the pizza the way the customer wants it, after receiving the order, the owner places instructions on the bag of ingredients the chef retrieves from the refrigerator. The owner tells the chef to always check the bag for instructions first. If there are no instructions on the bag, the chef should simply follow the directions posted on the wall.

In ePublisher Designer, the Project directory is the instructions posted on each bag. The pizza chef knows to check the bag (the project folder) first for instructions, then to check the instructions on the wall (the installation directory) if there are no instructions on the bag.

File Processing

As mentioned previously, ePublisher Designer breaks up the transform process into a series of pipelines, or groups of similar stages. Each stage defines an **.xsl** file to be executed which creates a portion of the output target. ePublisher Designer uses a combination of the Format's Pipelines, the GlobalFiles record (**files.info**), XSLT parameters and the root match template in a given XSL file to perform the action to be executed in a stage. These stages are defined in the **format.wwfmt** file located in each Format directory. The **format.wwfmt** file defines all of the pipelines and steps necessary to create a given output, and the relationships that each of these have to one another. There is no preconceived order in which pipelines fire; the ePublisher Designer engine calculates at run time the order in which each pipeline will fire, based on pipeline dependencies.

ePublisher Designer processes files based on type rather than by name. Each stage defines an .xsl file to be executed which creates a portion of the output target. All ePublisher Designer XSL stylesheets define four parameters by default. These are as follows:

GlobalFiles –

A project's **files.info**

GlobalProject -

The project file (***.wep**)

GlobalPipelineName -

The pipeline of the current stage.

GlobalInput -

The files selected in the Document Manager.

In addition, XSL stylesheets are passed all parameters defined for the current stage in **format.wwfmt**. Parameters passed to each XSLT file are usually used to load the various XML node sets needed for the current stage. For example, a stage may need to take information recorded in the Behaviors pipeline and merge it with information in the Links pipeline. The location of information generated in each stage is stored in the **files.info** file. Future stages may consult **files.info** to learn the location of previously generated information needed to complete that particular stage. Similarly, when that stage is complete, it will send notice to the **files.info** file that the information it has created is available for future stages to use if necessary.

What This Means For The User

Processing files based on type rather than by individual names allows developers to define workflows for any number of input-output files. This allows users to define as many XSL stages as they like and each stage may make use of any preceding file generated. Users may create or delete pipelines, add or delete stages, or insert elements that will dictate when a stage or pipeline is executed.

File Types

The section provides specific information about the files most commonly used in customization of ePublisher Designer.

Format Trait Info (*.fti)

Format Trait Info files end in the .fti extension. Format Trait Info files are located in both the ePublisher Designer Transforms directory, and the format directories of each output target.

Explanation

In every project, the user has a series of format and style options to customize the appearance of his output. In the GUI, these options can be accessed and manipulated through Target Settings and Style Designer on the ePublisher Designer toolbar. For the ePublisher Designer engine, these customization options reside in the Format Trait Info (***.fti**) files. Manipulating these files allows for customizing the location of options, creating or deleting options and parameters, or adjusting their values, or choosing a new default value.

Format Trait Info files must have the same base file name (that is, the file name without the file extension) as an XSL file in the current format in order to be processed by ePublisher Designer. The information displayed in the ePublisher GUI represents all Format Trait Info files associated with a given format. So, two or more Format Trait Info files may define a `<Setting />` but would only be displayed once in the GUI.

When ePublisher Designer is generating output, the settings specified in Target Settings and Style Designer are consulted, stored in the project file (**.wep**) and incorporated in the final generation of output.

Components

There are four elements in an Format Trait Info (**.fti**) file.

<Classes>

The only child element in <Classes> is <Class>. <Class> has a child element of <Item>. See Example 1 for sample code.

To facilitate organization, values for a given option may be grouped together as <Items> in a <Class>. The <Class> element can then be applied in the <Settings> and <RuleTraitsSet> elements to declare the options available for that specific option.

```
<Classes>

 <Class name="boolean">

  <Item value="true" stringid="boolean-true" />

  <Item value="false" stringid="boolean-false" />

 </Class>

 <Class name="color-name">

  <Item value="transparent" />

  <Item value="aqua" />

  <Item value="black" />

  <Item value="white" />

  <Item value="yellow" />

 </Class>

</Classes>
```

Example 1: Sample Code Illustrating Child Elements, Attributes, and Values in the <Classes> Element

<Groups>

The only child element to <Groups> is <Group>. The <Groups> element defines items that can be used to group properties available in the Style Designer. Users can create groups and apply the group element tag in the appropriate location in the <Settings> and

`<RuleTraitsSet>` elements of the Format Trait Info (**.fti**) file. This will assemble all `<Settings>` with a similar group together in the appropriate section of the GUI.

<Settings>

The only child element of the `<Settings>` element is `<Setting>`. The `<Setting>` element defines the parameters for the GUI's Target Settings dialog. A user may customize the target settings by making changes in this portion of the Format Trait Info (**.fti**) file.

<RuleTraitsSet>

The only child element of the `<RuleTraitsSet>` element is `<Options>`. `<Options>` contains the child element `<option>`. See Example 2 for sample code.

The `<option>` element defines the parameters for the GUI's Properties and Options Tabs in the Style Designer. A user may customize the Style Designer by making changes in this portion of the Format Trait Info (**.fti**) file.

```
<RuleTraitsSet>

 <RuleTraits category="Graphic">

  <Options>

   <Option name="file-extension" group="options" default=".jpg">

    <OptionClass name="file-extension" />

   </Option>

   <Option name="format" group="options" default="jpeg">

    <OptionClass name="image-format" />

   </Option>

   <Option name="color-depth" group="options" default="24">

    <OptionClass name="color-depth" />

   </Option>

  </Options>

 </RuleTraits>

</RuleTraitsSet>
```

Example 2: Sample Code Illustrating Child Elements, Attributes, and Values in the `<RuleTraitsSet>` Element

Relationships

Format Trait Info appears in the ePublisher Designer user interface. When a user effects changes to these setting through the user interface, those changes are written to the project file (***.wep**). A format's XSL transformations then have access to these values by reading the project file.

format.wwfmt

format.wwfmt files are located in the specific format directories.

Explanation

The **format.wwfmt** file defines all of the steps necessary to create output, and the relationships that each of these steps has to one another. These steps, called “stages” are grouped in “pipelines” of related stages. When every stage of every pipeline has been executed, the ePublisher Designer engine output is complete. There is no preconceived order in which pipelines fire; the ePublisher Designer engine calculates at run time the order in which each pipeline will fire. Users may create or delete pipelines, add or delete stages, or insert a `<Depends />` element that will dictates when a stage or pipeline is executed.

Components

The root element of **format.wwfmt** is `<Format />`. The child elements are `<Pipelines>` and `<Capabilities>`. See example 3 for sample code.

<Pipelines>

The `<Pipelines>` element is a container element for the `<Pipeline>` elements that define the steps needed to generate the format. The only child element of the `<Pipelines>` element is `<Pipeline>`.

`<Pipeline>`:

The `<Pipeline>` element is a container element for the `<Depends>` and `<Stage>` elements which comprise a given segment of output files needed to generate a format. This element requires a name attribute so that it can be identified by `<Depends>` elements within other `<Pipeline>` elements. The `<Pipeline>` element contains the following child elements:

`<Depends>`:

This element specifies which other `<Pipeline>` elements the current `<Pipeline>` requires to complete its task. Including a `<Depends>` elemen t in a pipeline is the only way to ensure a Pipeline does not execute before another Pipeline on which it depends. The ePublisher Designer engine calculates at run time the order in which each Pipeline will run.

`<Stage>`:

This element identifies an action to perform and specifies a configuration for the action. The action is identified via the type and action attributes, usually an XSL stylesheet, and the configuration is defined with `<Parameter>` elements. These `<parameter>` elements define what is to be worked on, what will be created, and what else should be done with the result.

Note:

With the exception of Global Files and GlobalProject, all parameters passed to XSL transforms are strings. Global Files and Global Project are node-sets which have already been loaded.

<Capabilities>

The `<Capabilities>` element contains only the child element `<Capability>` which defines information regarding what types of technologies or features a format supports.

```
<Capabilities>

 <Capability name="merge-context" value="false" />

   ...
```

```
 </Capabilities>
<Pipelines>
 <Pipeline name="CompanyInfo">
  <Stage type="xsl" action="wwtransform:common/companyinfo/companyinfo.xsl">
   <!-- Pull in Company Info .fti file -->
   <!--                                 -->
  </Stage>
 </Pipeline>
 <Pipeline name="DocumentBehaviors">
  <Stage type="xsl" action="wwtransform:common/behaviors/document.xsl">
   <Parameter name="ParameterDropDowns" value="false" />
   <Parameter name="ParameterPopups" value="false" />
    ...
  </Stage>
  <Stage type="xsl" action="wwtransform:common/behaviors/pullup.xsl">
   <Parameter name="ParameterDropDowns" value="false" />
   <Parameter name="ParameterPopups" value="false" />
    ...
  </Stage>
 </Pipeline>
</Pipelines>
```

Example 3: Sample Code Illustrating Child Elements, Attributes, and Values in the `format.wwfmt` file

Relationships

The locations of files generated by the action elements in the stages of the **format.wwfmt** file are stored in **files.info**. The **format.wwfmt** draws the same information from **files.info** as needed.

files.info

This file is created and stored with the project files.

Explanation

Each stage completes a small portion of the conversion. A completed stage creates new information that will contribute to the appearance or functionality of the new transform. Upon completion of the stage, the ePublisher Designer engine will write the location of the processed information in the **files.info** file for use by other stages attempting to complete their pipeline. Each stage may draw information from **files.info** and deposit information for use by another stage.

Components

The root element of files.info is `<Files />`. The child elements are `<File>` and `<Depends>`.

The `<File>` element holds the location of the data created by a stage.

The `<Depends>` element notes the location of data on which that file is dependent.

Relationships

This file stores the dependencies and locations of files created by the stages defined in **format.wwfmt**. The ePublisher Designer engine will provide this list of locations in order to process other stages in other pipelines.

Designer Project File (.wep)

The project file (**.wep**) file is a collection of all the necessary information required to create output. This includes all the source document, settings, options and customizations created by the user or through the GUI.

Stationery File (.wsxp)

.wsxp is the extension used for stationery files. Stationery is a file based upon configurations that have been made to a previous ePublisher Designer project. By saving a project as stationery, all of the settings and customizations that you captured in your project, including project format overrides, are automatically implemented in any new projects based on the stationery.

Express Project File (.wrp)

The project file (**.wrp**) file is a collection of all the necessary information required to create output and is ideally suited for repeated publishing tasks. Unlike the Designer Project (**.wep**), the Express project uses a Stationery to update (import) all of its settings, options, and other customizations.

XSL Match Templates

Root Match Templates

Stylesheets are made up of a number of templates, each of which defines what the XSLT processor should do when it matches a particular node in the XML source document. The XSLT processor populates the result document by instantiating a sequence of templates. Instantiation of a template means that the XSLT processor

- Copies any literal data from the template to the target
- Executes the XSLT instructions in the template

Templates are defined using the `<xsl:template>` element. The match attribute in the `<xsl:template>` element indicates which parts of the source document should be processed with the particular template.

Root Match Templates

Each XML document has a single root element. This element encloses all following elements and is therefore the parent element to all the other elements.

When the XSLT processor applies a stylesheet to an XML document, it begins processing with the root element of the XML source document. To process the root element, the XSLT processor searches the stylesheet for a template rule that matches the root element. A template rule matches the root element when the value of the template's match attribute is "/".

If the user explicitly defined a template rule that matches the root element, the XSLT processor finds it and implements that template to the entire XML document. If the XSLT processor does not find an explicitly defined template rule that matches the root element, the processor implements the default template that matches the root element. Every stylesheet includes this default template.

Root Match Templates in ePublisher Designer

The root match template has a number of responsibilities in ePublisher Designer. They are listed here:

- Recording Files and Dependencies
- Loading Node Sets
- Progress recording for the extension object that lives in the wwprogress namespace
- Up-to-Date checking on the projects output files
- Writing output

Recording Files and Dependencies

All root match templates in ePublisher Designer XSL files begin and end with the `<wwfiles:Files></wwfiles:Files>` elements. Within each root match template, usually near the bottom are one or more `<wwfiles:Files>` elements which include one or more `<wwfiles:Depends />` elements. The attributes in these elements provide information about the file such as the path, and type. Upon completion of a stage, the ePublisher Designer engine will write the location of the processed information in the **files.info** file for use by other stages attempting to complete their pipeline. Each stage may draw information from **files.info** and deposit information for use by another stage.

Loading Node Sets

Another responsibility of the ePublisher Designer XSL root match template is to load the node sets needed for the transformation. This is usually done by accessing one or more `<wwfiles:Files />` elements from files.info, and using the path attribute on each `<wwfiles:Files />` element together with the `document()` or `wwexsldoc:LoadXMLWithoutResolver` extension object, to load the node set for the transform. In most cases, the `<wwfiles:Files />` elements are selected using an XSL parameter from the **format.wwfmt** file, or Stage in the current Pipeline.

wwprogress

The `wwprogress` extension object sends messages to the ePublisher Designer progress indicator. If a root match template is being applied over several documents in several groups, then the number of documents or groups to be processed is recorded using the Start method in the `wwprogress` namespace.

Notice that an operation registered with the `wwprogress` extension object has concluded is initiated by using the End method.

Up-to-Date

The ePublisher Designer's XSL file's root match template also ensures whether files to be processed are up to date. This is accomplished by passing attributes accepted by the `wwfilesext:UpToDate` extension method.

Writing Output

If a file is determined to be not up to date, then a XSL template is called, with the result then passed along with information regarding what type of output file is to be written, to the `wwexsldoc:Document()` method for processing. If the output is to contain XML or HTML elements, then it is necessary to use the `msxsl:node-set()` method to make the contents of the variable behave as a new XML fragment.

Real Life Example

Let's return to our pizzeria. Our pizza chef, who forgets how to make a pizza whenever he sets out to do so, has a set of rules he follows posted on the wall. Sometimes, the owner pastes special instructions on the bag of ingredients. But there are some rules that are never altered. The chef may be instructed every time to wash his hands. Or not to smoke a cigarette while making a pizza. These rules that he must make sure to follow every single time during the entire pizza production process, are the equivalent of the root match template.

Extension Objects

While extremely powerful, XSL has many shortcomings when used to perform system level scripting tasks. Operations such as working with files and advanced string operations are not included in the standard XSL language.

XSL does provide a generic extension mechanism to add new features to the base language. ePublisher Designer provides a standard set of extension objects which enable XSL to generate all required formats.

XSL extensions live in a specific namespace. Users can define their own prefix to associate with a given namespace, but in general sticking with a consistent naming convention makes life easier.

Output Customizations

Presenting information in print form can involve very different presentation decisions than displaying information in a web-friendly form. Display can vary from one platform to another, or one format may have features unavailable in another. In a transform performed by ePublisher Designer, the user can make several customization choices through various GUI options. For example, Target Settings may allow a user to specify whether a certain piece of information is displayed, or how or where it is located on a page. The Style Designer allows the user to customize any style in the document to appear a certain way. By creating specific rules and conditions for how the document should appear when the transform is complete the user can add functionality or information for specific audiences, or make use of specific features of a given format.

By using XSL, ePublisher Designer allows the user to customize the process even further. The user may make additions, deletions, or modifications to the XSL files in ePublisher Designer to further customize the output. The user may even add, remove, or modify options available in Target Settings and Style Designer.

Transform Overrides

Users make decisions on a Project's final output through the ePublisher Designer GUI. Some project modifications, however, cannot be made through the GUI. In these instances, the user may access the XSL files used by the ePublisher Designer engine, make changes, and thereby generate the desired output. Users may even add, modify or delete options in the GUI via Format Trait Info (**.fti**) overrides.

When output is generated for the first time, ePublisher Designer consults the **format.wwfmt** file in the format directory that tracks which actions are to be executed to generate the desired format. If there are no user customizations, all the files consulted by **format.wwfmt** are in the Format or Applications Transforms directories. By default, WebWorks ePublisher Designer will check a project's local directory for files before seeking the files from the installation directory. Users can override files in the default format files by creating a mirror directory structure within a given Project's Format or Target Override directory. By placing a file of the same name at the correct location in the Project Format/Target Override directory, any file in a given format within the installation directory may be overridden.

Creating Transform Overrides

This procedure details the steps necessary to override a specific file and ensure that all previous functionality remains intact.

Note: Changes to **.xsl** and Format Trait Info (**.fti**) files are not supported through Quadralay's Product Support department. Assistance with modifying these files may be purchased through WebWorks Services.

Information about Overriding files

Modifying files in the installation directories is not recommended. Files should be modified in the project itself only. See the following procedure for the steps required to do this.

The file name nomenclature used in ePublisher Designer is case sensitive. Care should be taken to ensure directories and files created by the user match the original installation directory or file name exactly.

It is not necessary to copy any files into the project Formats directory that you are not explicitly overriding.

Procedure

These steps may be taken with any of the files you need to modify inside the Formats directory. This procedure assumes that ePublisher Designer has been installed in the default directory, and that your ePublisher Designer projects are located in the “Documents” folder. If the user has installed ePublisher Designer somewhere else, or stores projects somewhere else, remember to substitute those locations when following the procedure.

Note: This is the manual way of doing an override, but you can always use the ePublisher Designer Advanced Menu (specially if you are overriding the Shared folder).

1. In Windows, open your project folder. Create a new folder named "Formats".
2. Open this new Formats directory.
3. Duplicate the folder hierarchy of the file you wish to modify.

   See Figure 2 for the folder hierarchy necessary to modify the pages.xsl file in WebWorks Reverb.

   Substitute the correct folder and file name inside the Formats directory.

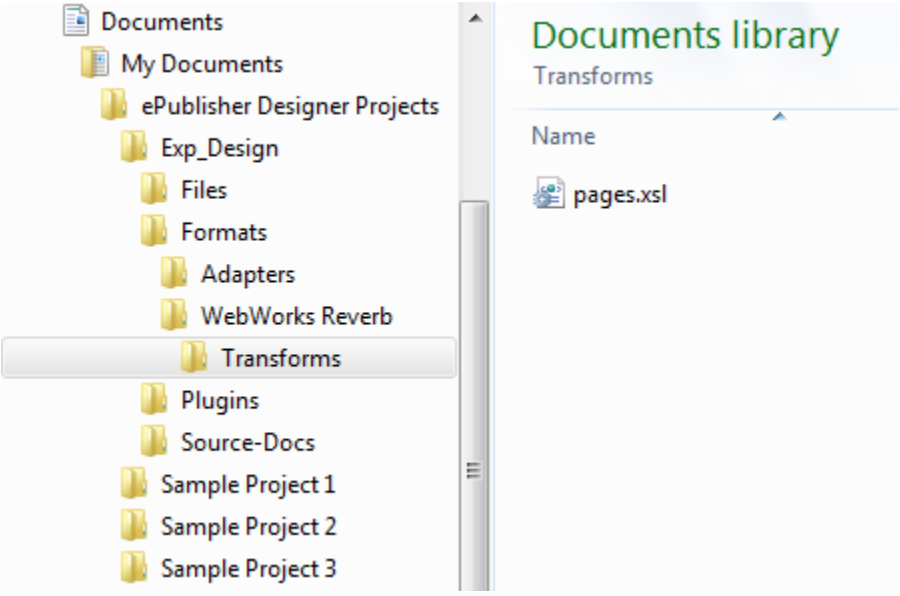


Figure 2: Folder Hierarchy Required To Modify pages.xsl in WebWorks Reverb

4. Navigate to the file you wish to modify in the ePublisher Designer installation directory.
5. Select and copy the file(s) you wish to override.
6. Navigate back to the Formats folder in the Projects directory created in steps 1-3.
7. Paste the file you copied from the installation directory into the folder created in step 3.
8. Modify files as needed.

After you have made the modifications you wish to make to the file in your project directory, save it, and the next time you click Generate in your ePublisher Designer project, this file will automatically override the default file and the changes you made will be incorporated into the output.

Note: When you save the ePublisher Designer project as Stationery, the project format overrides you have created will be saved with your Stationery. This stationery can then be used to create future projects.

XSLT Reference

XSLT is the primary language used in transforming source XML into the various types of generated output. This section is an XSLT Reference for use with ePublisher.

XSLT Documentation

Below is a collection of useful resources for better understanding XSLT:

Useful Videos from WebWorks:

- Basics of XML, XSL, and ePublisher Conversions
- ePublisher Introduction to XSL and Extension Methods
- ePublisher XSLT, .Net, and Custom Formats
- ePublisher Document and WIF Processing
- Creating ePublisher Transform Overrides

From the WebWorks Wiki:

- Debugging the XSL of an ePublisher Designer project
- XSL-FO Page template
- XML Transform Changes
- Locales
- Developer Documentation Topics

Microsoft XSLT and XPath Reference

- XSLT Elements
- XSLT Syntax Patterns
- XSLT Functions
- XPath Handling and Special Characters
- XPath Node Collection Indexing, Finding, and Grouping
- XPath Logical Operators
- XPath Comparisons

Microsoft XPath Functions Reference

- XPath Node-Set Functions
- XPath String Functions
- XPath Boolean Functions
- XPath Number Functions

XSLT Quick External Reference

- Character Set used for HTML, XSLT, XML

Good to Know

The below tools are useful for debugging and testing custom code in a controlled environment:

- For XSLT: XSLT fiddle
- For C#: .NET Fiddle
- For JavaScript/HTML/CSS: JSFiddle

Tool ePublisher uses to clean HTML documents: Tidy. If you need to customize Tidy, refer to the Tidy Quick Reference

Using Extension Objects

The WebWorks ePublisher engine uses XML, XSL, and XPath as the foundation for all processing. While extremely powerful, XSL has many shortcomings when used to perform system level scripting tasks. Operations such as working with files and advanced string operations are not included in the standard XSL language.

XSL does provide a generic extension mechanism to add new features to the base language. WebWorks ePublisher provides a standard set of extension objects which enable XSL to generate all required formats.

General XSL Extensions

XSL extensions live in a specific namespace. Users can define their own prefix to associate with a given namespace, but in general sticking with a consistent naming convention makes life easier.

For example, the XSL namespace is:

To define a prefix for it, one adds an XSL namespace declaration to your XSL stylesheet:

```
<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

 ..

</xsl:stylesheet>
```

Now the desired namespace can be referenced just using a prefix rather than the actual namespace. Therefore, the following two lines are equivalent:

```
<xsl:variable name="VarNew" select="'Hello'" />

<variable xmlns="http://www.w3.org/1999/XSL/Transform" name="VarNew"
 select="'Hello'" />
```

Microsoft's XSL transform implementation does not support extension elements at this time. Therefore, methods must be called from within "select" contexts.

```
<xsl:value-of select="wwexsldoc:Document($VarResult, $VarPath)" />

<xsl:variable name="VarDocumentWrite" select="wwexsldoc:Document($VarResult,
 $VarPath)" />
```

Microsoft Extensions

Extension objects that are defined and implemented by Microsoft as part of the .NET XSL transform runtime.

Purpose

Microsoft implemented additional methods not part of the XSLT 1.1 standard. This work was done prior to the XSLT 2.0 and EXSLT specifications being finalized.

Namespace

urn:schemas-microsoft-com:xslt

Prefix

msxsl

Method

msxsl:node-set(string textualXML):

Description

Converts textual XML into a node set in a new XML document. Most often used to convert intermediate XML back into a working node set for additional processing.

Returns

A node set equivalent to the provided textual XML.

Note: The Microsoft XSL transform engine also supports custom extensions defined with script blocks.

Using ePublisher XSLT Extensions

ePublisher implements a variety of extension objects to enable full processing of desired output within the language of XSL. In order to use them in XSL transforms they must be declared.

Here is an example of how to define namespace in you XSL file:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns="urn:WebWorks-Images-Schema"

                              xmlns:xsl="http://www.w3.org/1999/XSL/
Transform"

                              xmlns:wwlog="urn:WebWorks-XSLT-Extension-Log"

                              exclude-result-prefixes="xsl wwlog"

>



</xsl:stylesheet>
```

ePublisher Platform XSLT Extensions

ePublisher implements a variety of extension objects to enable full processing of desired output within the language of XSL.

Namespace	Prefix	Class Name	Description
urn:WebWorks-XSLT-Extension-Adapter	wwadapter	**Adapter**	XSLT extension functions for use in ePublisher Adapter stylesheets.
urn:WebWorks-Adapter-Configuration-Extension	wwadapterconf	**AdapterConfiguration**	Provides access to the adapter configuration.
urn:WebWorks-XSLT-Extension-DateTimeUtilities	wwdatetime	**DateTime**	Allows XSL pages access to various datetime methods
urn:WebWorks-XSLT-Extension-Environment	wwenv	**Environment**	Enable XSL transforms to query the current system environment for the location and state of programs and variables.

Namespace	Prefix	Class Name	Description
urn:WebWorks-XSLT-Extension-Execute	wwexec	**Exec**	Allows XSL stylesheets to execute external programs and process results. Provides the return code, stdout, and stderr results from the running proces
urn:WebWorks-XSLT-Extension-ExecPython	wwpython	**ExecPython**	Allows XSL stylesheets to execute python programs and process results.
urn:WebWorks-XSLT-Extension-Sass	wwsass	**Sass**	Allows XSL stylesheets to manage SASS files
urn:WebWorks-XSLT-Extension-Document	wwexsldoc	**ExslDocument**	Allow multiple output files from a single XSL transform. Also provides routines to quickly load XML files without invoking XML validators as well as utility methods to enable correct output formatting.
urn:WebWorks-XSLT-Extension-Files	wwfilesext	**Files**	Enables incrementa build support.
urn:WebWorks-XSLT-Extension-FileSystem	wwfilesystem	**FileSystem**	Allow XSL transforms to quer and manipulate file and directories. Also handles syste path parsing and processing.
urn:WebWorks-XSLT-Extension-Fonts	wwfonts	**Fonts**	Answer questions about fonts that

Namespace	Prefix	Class Name	Description
			might affect forma output.
urn:WebWorks-Imaging-Info	wwimageinfo	**Imaging**	Enable processing images within XSL transforms. Return information about a particular image file, including width and height, image format, bit-depth, path on system, et
urn:WebWorks-XSLT-Extension-Log	wwlog	**Log**	Enables XSL transforms to report messages, warnings, and erro to the generation log.
urn:WebWorks-XSLT-Extension-MultiSearchReplace	wwmultisere	**MultiSearchReplaceExt**	Replaces multiple strings in a single operation.
urn:WebWorks-XSLT-Extension-NodeSet	wwnodeset	**NodeSet**	Miscellaneous node set functions.
urn:WebWorks-XSLT-Extension-Progress	wwprogress	**Progress**	Reports progress during long lived XSL transforms.
urn:WebWorks-XSLT-Extension-Project	wwprojext	**Project**	Query for information about the currently running project.
urn:WebWorks-XSLT-Extension-StageInfo	wwstageinfo	**StageInfo**	Allows XLST processing to store and retrieve key/ value pairs as needed to track state.
urn:WebWorks-XSLT-Extension-StringUtilities	wwstring	**StringUtilities**	Extend the availabl string processing methods to XSL to include message formatting,

Namespace	Prefix	Class Name	Description
			specialized text escaping, regular expression operations, etc.
urn:WebWorks-XSLT-Extension-Units	wwunits	**Units**	Utility methods for extracting units and value from raw strings along with unit-to-unit conversion routines
urn:WebWorks-XSLT-Extension-URI	wwuri	**URI**	Utility methods which convert to and from file paths and create absolute or relative URIs.
urn:WebWorks-XSLT-Extension-Zip	wwzip	**ZipExtension**	Allow XSL transforms to hand zip archives.

Class Documentation

Adapter

urn:WebWorks-XSLT-Extension-Adapter

Functions

void AddToPDFPageNumberOffset (int addToPageNumberOffset)

Adds to the PDF page number offset.

bool GeneratePDF (string originalDocumentPath, string conversionPDFDocumentPath, bool singleFile, XPathNodeIterator tocStyleNodesIterator, XPathNodeIterator groupFileNodesIterator, string pdfJobSettings, string pdfFilePath)

Generates a PDF.

bool GeneratePDFWithSaveAs (string originalDocumentPath, string conversionPDFDocumentPath, bool singleFile, XPathNodeIterator tocStyleNodesIterator, XPathNodeIterator groupFileNodesIterator, string pdfJobSettings, string pdfFilePath)

Generates a PDF using FrameMaker save as.

bool GeneratePostScriptForImage (object input, string postScriptPath)

Generates postscript for image.

long GeneratePostScriptForPDF (string originalDocumentPath, string conversionPDFDocumentPath, bool singleFile, XPathNodeIterator tocStyleNodesIterator, XPathNodeIterator groupFileNodesIterator, string postScriptFilePath)

Generates a postscript for PDF.

void SetPDFPageNumberOffset (int pageNumberOffset)

Sets PDF page number offset.

bool TemporaryLicense (string toolAdapterName)

Checks if user is running a temporary license for specified 'toolAdapterName'.

Detailed Description

urn:WebWorks-XSLT-Extension-Adapter

XSL extension functions for use in ePublisher Adapter stylesheets.

void AddToPDFPageNumberOffset (int *addToPageNumberOffset*)

Adds to the PDF page number offset.

Parameters:

addToPageNumberOffset	The offset to page number offset.

bool GeneratePDF (string *originalDocumentPath*, string *conversionPDFDocumentPath*, bool *singleFile*, XPathNodeIterator *tocStyleNodesIterator*, XPathNodeIterator *groupFileNodesIterator*, string *pdfJobSettings*, string *pdfFilePath*)

Generates a PDF.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

originalDocumentPath	Full pathname of the original document file.
conversionPDFDocumentPath	Full pathname of the conversion PDF document file.
singleFile	True for single file.
tocStyleNodesIterator	The TOC style node set.
groupFileNodesIterator	The group file node set.
pdfJobSettings	The PDF job settings.
pdfFilePath	Full pathname of the PDF file.

Returns:

True if it succeeds, false if it fails.

bool GeneratePDFWithSaveAs (string *originalDocumentPath*, string *conversionPDFDocumentPath*, bool *singleFile*, XPathNodeIterator *tocStyleNodesIterator*, XPathNodeIterator *groupFileNodesIterator*, string *pdfJobSettings*, string *pdfFilePath*)

Generates a PDF using FrameMaker save as.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

originalDocumentPath	Full pathname of the original document file.
conversionPDFDocumentPath	Full pathname of the conversion PDF document file.
singleFile	True for single file.
tocStyleNodesIterator	The TOC style node set.
groupFileNodesIterator	The group file node set.
pdfJobSettings	The PDF job settings.
pdfFilePath	Full pathname of the PDF file.

Returns:

True if it succeeds, false if it fails.

bool GeneratePostScriptForImage (object *input*, string *postScriptPath*)

Generates postscript for image.

Parameters:

input	Node set containing content to be converted to postscript.
postScriptPath	Full pathname of the postscript file to create.

Returns:

True if it succeeds, false if it fails.

long GeneratePostScriptForPDF (string *originalDocumentPath*, string *conversionPDFDocumentPath*, bool *singleFile*, XPathNodeIterator *tocStyleNodesIterator*, XPathNodeIterator *groupFileNodesIterator*, string *postScriptFilePath*)

Generates a postscript for PDF.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

originalDocumentPath	Full pathname of the original document file.
conversionPDFDocumentPath	Full pathname of the conversion PDF document file.
singleFile	True for single file.
tocStyleNodesIterator	The TOC style node set.
groupFileNodesIterator	The group file node set.
postScriptFilePath	Full pathname of the postscript file.

Returns:

Number of pages in the PDF.

void SetPDFPageNumberOffset (int *pageNumberOffset*)

Sets PDF page number offset.

Parameters:

pageNumberOffset	The page number offset.

bool TemporaryLicense (string *toolAdapterName*)

Checks if user is running a temporary license for specified 'toolAdapterName'.

Parameters:

toolAdapterName	Name of the tool adapter.

Returns:

True if is temporary, false otherwise.

AdapterConfiguration

urn:WebWorks-Adapter-Configuration-Extension

Functions

string GetValue (string name)

Gets a value.

string GetValue (string name, string defaultValue)

Gets a value and uses 'defaultValue' when no value set.

Detailed Description

urn:WebWorks-Adapter-Configuration-Extension

XSL transform functions for getting information about the input configuration for adapters.

string GetValue (string *name*)

Gets a value.

Parameters:

name	The input configuration item name.

Returns:

The value.

string GetValue (string *name*, string *defaultValue*)

Gets a value and uses 'defaultValue' when no value set.

Parameters:

name	The input configuration item name.
defaultValue	The default value.

Returns:

The value.

DateTimeUtilities

urn:WebWorks-XSLT-Extension-DateTimeUtilities

Functions

string GetNow (string format)

Takes a format string and gets DateTime in the given format.

string GetGenerateStart (string format)

Gets mStartDate and converts to given format, returns as string.

string GetFileCreated (string filePath, string format)

Gets created date from filepath and converts to given format.

string GetFileLastModified (string filePath, string Format)

Gets last modified date from filepath and converts to given format.

string GetFromDateTimeString (string dateTime, string inputFormat, string outputFormat)

Gets a date and a specific format, converts the date to another format.

Detailed Description

urn:WebWorks-XSLT-Extension-DateTimeUtilities

Allows XSL pages access to various datetime methods.

string GetNow (string *format*)

Takes a format string and gets DateTime in the given format.

Parameters:

format	The datetime format.

Returns:

A datetime string.

string GetGenerateStart (string *format*)

Gets mStartDate at generation start and converts to given format.

Parameters:

format	The datetime format.

Returns:

A datetime string.

string GetFileCreated (string *filepath,* string *format*)

Gets created date from filepath and converts to given format.

Parameters:

filepath	Full pathname of the original document file.
format	The datetime format.

Returns:

A datetime string.

string GetFileLastModified (string *filepath,* string *format*)

Gets last modified date from filepath and coverts to given format.

Parameters:

filepath	Full pathname of the original document file.
format	The datetime format input.

Returns:

A datetime string.

string GetFromDateTimeString (string *dateTime,* string *inputFormat,* string *outputFormat*)

Gets a date and specific format, converts the date to another format.

Parameters:

dateTime	The datetime input.
inputFormat	The datetime format input.
outputFormat.	The datetime format output.

Returns:

A datetime string.

Environment

urn:WebWorks-XSLT-Extension-Environment

Functions

string ApplicationBaseHelpURI ()

Application base help URI.

string CurrentUILocale ()

Current user interface locale.

long GetTotalMemory ()

Reports the total amount of memory used by the running process. Useful for optimizing XSL to reduce memory usage.

long GetTotalMemory (bool forceFullCollection)

Reports the total amount of memory used by the running process. Useful for optimizing XSL to reduce memory usage.

string HTMLHelpWorkshopPath ()

HTML help workshop path.

string JavaBits ()

Gets latest version of either JDK pr JRE bits stored in registry.

string JavaHome ()

Gets latest version of either JDK or JRE home directory path.

string JavaVersion ()

Gets latest version of either the JDK or JRE version stored in registry.

string JDKBits ()

Gets Java Development Kit (JDK) bits stored in registry.

string JDKHome ()

Gets Java Development Kit (JDK) home directory path.

string JDKVersion ()

Gets Java Development Kit (JDK) version stored in registry.

string JREBits ()

Gets Java Runtime Environment (JRE) bits stored in registry.

string JREHome ()

Gets Java Runtime Environment (JRE) home directory path.

string JREVersion ()

Gets Java Runtime Environment (JRE) version stored in registry.

bool RequestedPipeline (string pipelineName)

Determine if 'pipelineName' has been scheduled for processing.

Detailed Description

urn:WebWorks-XSLT-Extension-Environment

Enable XSL transforms to query the current system environment for the location and state of programs and variables.

string ApplicationBaseHelpURI ()

Application base help URI.

Returns:

A string.

string CurrentUILocale ()

Current user interface locale.

Returns:

A string.

long GetTotalMemory ()

Reports the total amount of memory used by the running process. Useful for optimizing XSL to reduce memory usage.

Returns:

The total memory in bytes.

long GetTotalMemory (bool *forceFullCollection*)

Reports the total amount of memory used by the running process. Useful for optimizing XSL to reduce memory usage.

Parameters:

forceFullCollection	True to force full collection.

Returns:

The total memory in bytes.

string HTMLHelpWorkshopPath ()

HTML help workshop path.

Returns:

A string.

string JavaBits ()

Gets latest version of either JDK pr JRE bits stored in registry.

Returns:

The JDK/JRE bits string stored in registry.

string JavaHome ()

Gets latest version of either JDK or JRE home directory path.

Returns:

JDK/JRE home directory path string.

string JavaVersion ()

Gets latest version of either the JDK or JRE version stored in registry.

Returns:

JDK/JRE version string stored in registry.

string JDKBits ()

Gets Java Development Kit (JDK) bits stored in registry.

Returns:

JDK bits string stored in registry.

string JDKHome ()

Gets Java Development Kit (JDK) home directory path.

Returns:

JDK home directory path string.

Determine the path to the current JDK and jar some files.

```
<xsl:variable name="VarJDKHome" select="wwenv:JDKHome()" />

<xsl:variable name = "VarJarPath" select="wwfilesystem:Combine($VarJDKHome, 'jar')" />

<xsl:variable name = "VarJarCommand" select="wwexec:ExecuteCommand($VarJarPath, 'cvf',
 'output.jar', '.')" />
```

string JDKVersion ()

Gets Java Development Kit (JDK) version stored in registry.

Returns:

JDK version string stored in registry.

string JREBits ()

Gets Java Runtime Environment (JRE) bits stored in registry.

Returns:

The JRE bits string stored in registry

string JREHome ()

Gets Java Runtime Environment (JRE) home directory path.

Returns:

JRE home directory path string.

string JREVersion ()

Gets Java Runtime Environment (JRE) version stored in registry.

Returns:

JRE version string stored in registry.

bool RequestedPipeline (string *pipelineName*)

Determine if 'pipelineName' has been scheduled for processing.

Parameters:

pipelineName	Name of the pipeline.

Returns:

True if scheduled for processing, false otherwise.

Generate a given report only if specifically enabled in a project or a user has specifically requested that report.

```
<xsl:variable name="VarGenerateReportSetting" select="wwprojext:GetFormatSetting('report-
filenames-generate', 'true') = 'true'" /> <xsl:variable name = "VarRequestedPipeline"
 select="wwenv:RequestedPipeline($GlobalPipelineName)" />

<xsl:variable name = "VarGenerateReport" select="($VarGenerateReportSetting) or
 ($VarRequestedPipeline)" />
```

Exec

urn:WebWorks-XSLT-Extension-Execute

Functions

XPathNodeIterator Execute (string commandLine)

Runs the command line.

XPathNodeIterator ExecuteCommand (string command [, string argument1, string argument2, string argument3, string argument4, string argument5, string argument6, string argument7, string argument8, string argument9, string argument10, string argument11, string argument12, string argument13, string argument14, string argument15, string argument16, string argument17, string argument18, string argument19, string argument20])

Runs the specified command formatted as a string with optional argument(s) using the current target's output directory as the working directory.

XPathNodeIterator ExecuteCommandNoReturn (string command)

Runs identical to ExecuteCommand (string command) with the exception that the stdout and stderr streams are not returned in the node set. Instead these are written to the Log directory in a file with the same base filename as the command.

XPathNodeIterator ExecuteCommandInDirectory (string directoryPath, string command [, string argument1, string argument2, string argument3, string argument4, string argument5, string argument6, string argument7, string argument8, string argument9, string argument10, string argument11, string argument12, string argument13, string argument14, string argument15, string argument16, string argument17, string argument18, string argument19, string argument20])

Runs the specified command formatted as a string with optional argument(s) using the specified directory as the working directory.

XPathNodeIterator ExecuteCommandInDirectoryWithTimeout (long timeoutInSeconds, string directoryPath, string command [, string argument1, string argument2, string argument3, string argument4, string argument5, string argument6, string argument7, string argument8, string argument9, string argument10, string argument11, string argument12, string argument13, string argument14, string argument15, string argument16, string argument17, string argument18, string argument19, string argument20])

Runs the specified command formatted as a string with optional argument(s) using the specified directory as the working directory.

XPathNodeIterator ExecuteCommandWithTimeout (long timeoutInSeconds, string command [, string argument1, string argument2, string argument3, string argument4, string argument5, string argument6, string argument7, string argument8, string argument9, string argument10, string argument11, string argument12, string argument13, string argument14, string argument15, string argument16, string argument17, string argument18, string argument19, string argument20])

Runs the specified command formatted as a string with optional argument(s) using the current target's output directory as the working directory.

XPathNodeIterator ExecuteInDirectory (string directoryPath, string commandLine)

Runs the command line in the specified working directory.

XPathNodeIterator ExecuteInDirectoryWithTimeout (long timeoutInSeconds, string directoryPath, string commandLine)

Runs the command line in the specified working directory.

XPathNodeIterator ExecuteProgramWithArguments (string program, string arguments)

Runs the specified program with arguments formatted as a string using the current target's output directory as the working directory.

XPathNodeIterator ExecuteProgramWithArgumentsInDirectory (string directoryPath, string program, string arguments)

Runs the specified program with arguments formatted as a string in the specified working directory.

XPathNodeIterator ExecuteProgramWithArgumentsInDirectoryWithTimeout (long timeoutInSeconds, string directoryPath, string program, string arguments)

Runs the specified program with arguments formatted as a string in the specified working directory.

XPathNodeIterator ExecuteProgramWithArgumentsWithTimeout (long timeoutInSeconds, string program, string arguments)

Runs the specified program with arguments formatted as a string using the current target's output directory as the working directory.

XPathNodeIterator ExecuteWithTimeout (long timeoutInSeconds, string commandLine)

Runs the command line.

Detailed Description

urn:WebWorks-XSLT-Extension-Execute

Allows XSL stylesheets to execute external programs and process results. Provides the return code, stdout, and stderr results from the running process.

```
<wwexec:Result version="1.0" retcode="-1">

  <wwexec:Stream name = "Output" >

    Standard output will show up here, aka stdout.

  </wwexec:Stream>

  <wwexec:Stream name = "Error" >

    Standard error will show up here, aka stderr.

  </wwexec:Stream>

</wwexec:Result>
```

XPathNodeIterator Execute (string *commandLine*)

Runs the command line.

Parameters:

commandLine	The command line to be executed.

Returns:

A node set.

Execute prettycool.exe –stdout "Isn't this grand?" –stderr nothing.

```
<xsl:variable name="VarExecResult" select="wwexec:Execute('prettycool.exe -stdout &quot;Isn\'t
 this grand?&quot; --stderr nothing')" />
```

XPathNodeIterator ExecuteCommand (string *command [*, string *argument1*, string *argument2*, string *argument3*, string *argument4*, string *argument5*, string *argument6*, string *argument7*, string *argument8*, string *argument9*, string *argument10*, string *argument11*, string *argument12*, string *argument13*, string *argument14*, string *argument15*, string *argument16*, string *argument17*, string *argument18*, string *argument19*, string *argument20]*)

Runs the specified command formatted as a string with optional argument(s) using the current target's output directory as the working directory.

Parameters:

command	The command.
argument1 – argument20	Optional arguments.

Returns:

A node set.

XPathNodeIterator ExecuteCommandNoReturn (string *command*)

Runs identical to **ExecuteCommand (string command)** with the exception that the stdout and stderr streams are not returned in the node set. Instead these are written to the Log directory in a file with the same base filename as the *command* parameter.

Parameters:

command	The command.

Returns:

A node set.

XPathNodeIterator ExecuteCommandInDirectory (string *directoryPath*, string *command [*, string *argument1*, string *argument2*, string *argument3*, string *argument4*, string *argument5*, string *argument6*, string *argument7*, string *argument8*, string *argument9*, string *argument10*, string *argument11*, string *argument12*, string *argument13*, string *argument14*, string *argument15*, string *argument16*, string *argument17*, string *argument18*, string *argument19*, string *argument20]*)

Runs the specified command formatted as a string with optional argument(s) using the specified directory as the working directory.

Parameters:

directoryPath	Full pathname of the directory file.
command	The command.
argument1 – argument20	Optional arguments.

Returns:

A node set.

XPathNodeIterator ExecuteCommandInDirectoryWithTimeout (long *timeoutInSeconds*, string *directoryPath*, string *command [*, string *argument1*, string *argument2*, string *argument3*, string *argument4*, string *argument5*, string *argument6*, string *argument7*, string *argument8*, string *argument9*, string *argument10*, string *argument11*, string *argument12*, string *argument13*, string *argument14*, string *argument15*, string *argument16*, string *argument17*, string *argument18*, string *argument19*, string *argument20]*)

Runs the specified command formatted as a string with optional argument(s) using the specified directory as the working directory.

Will stop operation if timeout in seconds elapses.

Parameters:

timeoutInSeconds	The timeout in seconds.
directoryPath	Full pathname of the directory file.
command	The command.
argument1 – argument20	Optional arguments.

Returns:

A node set.

XPathNodeIterator ExecuteCommandWithTimeout (long *timeoutInSeconds*, string *command [*, string *argument1*, string *argument2*, string *argument3*, string *argument4*, string *argument5*, string *argument6*, string *argument7*, string *argument8*, string *argument9*, string *argument10*, string *argument11*, string *argument12*, string *argument13*, string *argument14*, string *argument15*, string *argument16*, string *argument17*, string *argument18*, string *argument19*, string *argument20]*)

Runs the specified command formatted as a string with optional argument(s) using the current target's output directory as the working directory.

Will stop operation if timeout in seconds elapses.

Parameters:

timeoutInSeconds	The timeout in seconds.
command	The command.
argument1 – argument20	Optional arguments.

Returns:

A node set.

XPathNodeIterator ExecuteInDirectory (string *directoryPath*, string *commandLine*)

Runs the command line in the specified working directory.

Parameters:

directoryPath	Full pathname of the directory.
commandLine	The command line to be executed.

Returns:

A node set.

Execute prettycool.exe –stdout "Isn't this grand?" –stderr nothing in the directory C:\ workingdir.

```
<xsl:variable name="VarExecResult" select="wwexec:ExecuteInDirectory('C:\\workingdir',
 'prettycool.exe --stdout &quot;Isn\'t this grand?&quot; --stderr nothing')" />
```

XPathNodeIterator ExecuteInDirectoryWithTimeout (long *timeoutInSeconds*, string *directoryPath*, string *commandLine*)

Runs the command line in the specified working directory.

Will stop operation after specified elapsed time is exceeded.

Parameters:

timeoutInSeconds	The timeout in seconds.
directoryPath	Full pathname of the directory.
commandLine	The command line to be executed.

Returns:

A node set.

XPathNodeIterator ExecuteProgramWithArguments (string *program*, string *arguments*)

Runs the specified program with arguments formatted as a string using the current target's output directory as the working directory.

Parameters:

program	The program.
arguments	The arguments.

Returns:

A node set.

Runs the specified program with arguments formatted as a string using the current target's output directory as the working directory.

```
<xsl:variable name="VarExecResult" select="wwexec:ExecuteProgramWithArguments('prettycool.exe',
 '--stdout &quot;Isn\'t this grand?&quot; --stderr nothing')" />
```

XPathNodeIterator ExecuteProgramWithArgumentsInDirectory (string *directoryPath*, string *program*, string *arguments*)

Runs the specified program with arguments formatted as a string in the specified working directory.

Parameters:

directoryPath	Full pathname of the directory.
program	The program.
arguments	The arguments.

Returns:

A node set.

Execute prettycool.exe –stdout "Isn't this grand?" –stderr nothing in the directory C:\ workingdir.

```
<xsl:variable name="VarExecResult" select="wwexec:ExecuteProgramWithArgumentsInDirectory('C:\
\workingdir', 'prettycool.exe', '--stdout &quot;Isn\'t this grand?&quot; --stderr nothing')" />
```

XPathNodeIterator ExecuteProgramWithArgumentsInDirectoryWithTimeout (long *timeoutInSeconds*, string *directoryPath*, string *program*, string *arguments*)

Runs the specified program with arguments formatted as a string in the specified working directory.

Will stop operation after specified elapsed time is exceeded.

Parameters:

timeoutInSeconds	The timeout in seconds.
directoryPath	Full pathname of the directory.
program	The program.
arguments	The arguments.

Returns:

A node set.

XPathNodeIterator ExecuteProgramWithArgumentsWithTimeout (long *timeoutInSeconds*, string *program*, string *arguments*)

Runs the specified program with arguments formatted as a string using the current target's output directory as the working directory.

Will stop operation after specified elapsed time is exceeded.

Parameters:

timeoutInSeconds	The timeout in seconds.
program	The program.
arguments	The arguments.

Returns:

A node set.

XPathNodeIterator ExecuteWithTimeout (long *timeoutInSeconds*, string *commandLine*)

Runs the command line.

Will stop operation after specified elapsed time is exceeded.

Parameters:

timeoutInSeconds	The timeout in seconds.
commandLine	The command line to be executed.

Returns:

A node set.

ExecPython

urn:WebWorks-XSLT-Extension-ExecPython

Functions

XPathNodeIterator ExecutePyScriptInCommandLine (string commandLine)

Runs python with the specified commandLine.

XPathNodeIterator ExecPyScript (string pyScriptPath [, string argument1, string argument2, string argument3, string argument4, string argument5, string argument6, string argument7, string argument8, string argument9, string argument10, string argument11, string argument12, string argument13, string argument14, string argument15, string argument16, string argument17, string argument18, string argument19])

Runs python with the specified script with zero or more arguments using the current target's output directory as the working directory.

XPathNodeIterator ExecutePyScriptInDirectoryInCommandLine (string directoryPath, string commandLine)

Runs python with the specified script with zero or more arguments using the specified directory as the working directory.

XPathNodeIterator ExecPyScriptInDirectory (string directoryPath, string pyScriptPath [, string argument1, string argument2, string argument3, string argument4, string argument5, string argument6, string argument7, string argument8, string argument9, string argument10, string argument11, string argument12, string argument13, string argument14, string argument15, string argument16, string argument17, string argument18, string argument19])

Runs python with the specified script with zero or more arguments using the specified directory as the working directory.

Detailed Description

urn:WebWorks-XSLT-Extension-ExecPython

Allows XSL stylesheets to execute python programs and process results. Provides the return code, stdout, and stderr results from the running process.

```
<wwexec:Result version="1.0" retcode="-1">

  <wwexec:Stream name = "Output" >

    Standard output will show up here, aka stdout.

  </wwexec:Stream>

  <wwexec:Stream name = "Error" >

    Standard error will show up here, aka stderr.

  </wwexec:Stream>

</wwexec:Result>
```

XPathNodeIterator ExecutePyScriptInCommandLine (string *commandLine*)

Runs python with the specified command line.

Parameters:

commandLine	The command line in python to be executed.

Returns:

A node set.

```
<xsl:variable name="VarExecResult" select="wwpython:ExecutePyScriptInCommandLine('script.py
 some_arg')" />
```

XPathNodeIterator ExecPyScript (string pyScriptPath [, string argument1, string argument2, string argument3, string argument4, string argument5, string argument6, string argument7, string argument8, string argument9, string argument10, string argument11, string argument12, string argument13, string argument14, string argument15, string argument16, string argument17, string argument18, string argument19])

Runs python with the specified script with zero or more arguments using the current target's output directory as the working directory.

Parameters:

pyScriptPath	Path to the script file.
argument1 – argument19	Optional arguments.

Returns:

A node set.

```
<xsl:variable name="VarExecResult" select=" wwpython:ExecPyScript('script.py', 'some_arg')" />
```

XPathNodeIterator ExecutePyScriptInDirectoryInCommandLine (string directoryPath, string commandLine)

Runs python with the specified script with zero or more arguments using the specified directory as the working directory.

Parameters:

directoryPath	The working directory path where the python command line is going to be executed.
commandLine	The command line in python to be executed.

Returns:

A node set.

XPathNodeIterator ExecPyScriptInDirectory (string directoryPath, string pyScriptPath [, string argument1, string argument2, string argument3, string argument4, string argument5, string argument6, string argument7, string argument8, string argument9, string argument10, string argument11, string argument12, string argument13, string argument14, string argument15, string argument16, string argument17, string argument18, string argument19])

Runs python with the specified script with zero or more arguments using the specified directory as the working directory.

Parameters:

directoryPath	The working directory path where the python command line is going to be executed.
pyScriptPath	Path to the script file.
argument1 – argument19	Optional arguments.

Returns:

A node set.

Sass

urn:WebWorks-XSLT-Extension-Sass

Functions

XPathNodeIterator SassToCss (string inputSassFilePath, string outputCssFilePath)

Creates a CSS file out of a SASS file.

void ReplaceAllVariablesInFile (string inputSassFilePath, object replacements)

Replaces a collection of SASS variables in a file with those specified by the user.

Detailed Description

urn:WebWorks-XSLT-Extension-Sass

Allows XSL stylesheets to manage SASS files.

XPathNodeIterator SassToCss (string inputSassFilePath, string outputCssFilePath)

Creates a CSS file out of a SASS file.

Parameters:

inputSassFilePath	Path to the input SASS file.
outputCssFilePath	Path to the output CSS file.

Returns:

A node set.

```
<xsl:variable name="VarSassToCssResult" select="wwsass:SassToCss('input.sass', 'output.css')" />
```

void ReplaceAllVariablesInFile (string inputSassFilePath, object replacements)

Replaces a collection of SASS variables in a file with those specified by the user.

Parameters:

inputSassFilePath	Path to the input SASS file.
replacements	Object that represents an XML collection of SASS variables.

Replaces matching variables in the input file with variables specified inside the replacements object.

```
<xsl:variable name="VarSassVariableReplacementsAsXML">
 <wwsass:Variable name="footer_height" value="50px" />
 <wwsass:Variable name="menu_width" value="100px" />
</xsl:variable>
<xsl:variable name = "VarSassVariableReplacements" select="msxsl:node-
set($VarSassVariableReplacementsAsXML)/*" />

<xsl:variable name="VarReplaceVariables" select="wwsass:ReplaceAllVariablesInFile('file.scss',
 $VarSassVariableReplacements)"/>
```

ExslDocument

urn:WebWorks-XSLT-Extension-Document

Functions

void Document (object input, string path [, string encoding, string method, string version, string indent, string omit_xml_declaration, string standalone, string doctype_public, string doctype_system, string cdata_section_elements, string media_type])

Writes the specified node set to a file at the location defined by path. Optionally uses specified encoding for writing content. Optionally uses specified method (i.e. text, xhtml, xml...) and version. Optionally specify if file is to be formatted with indentation with 'yes' or 'no'. Optionally specify if file will have XML declaration with 'yes' or 'no'. Optionally specify if the file will be standalone with 'yes' or 'no'. Optionally specify public and system doctype modifiers.

XPathNodeIterator LoadXMLWithoutResolver (string uriAsString [, bool preserveSpace])

Loads an XML file without resolving internal paths and validation DTDs.

XPathNodeIterator LoadXMLWithResolver (string uriAsString [, bool preserveSpace])

Loads an XML file while also resolving internal paths and validating DTDs.

XPathNodeIterator MakeEmptyElement (object input)

Converts a non-empty XML node to an empty node.

Detailed Description

urn:WebWorks-XSLT-Extension-Document

Allow multiple output files from an single XSL transform. Also provides routines to quickly load XML files without invoking XML validators as well as utility methods to enable correct output formatting.

void Document (object *input*, string *path [*, string *encoding*, string *method*, string *version*, string *indent*, string *omit_xml_declaration*, string *standalone*, string *doctype_public*, string *doctype_system*, string *cdata_section_elements*, string *media_type]*)

Writes the specified node set to a file at the location defined by path. Optionally uses specified encoding for writing content. Optionally uses specified method (i.e. text, xhtml, xml...) and version. Optionally specify if file is to be formatted with indentation with 'yes' or 'no'. Optionally specify if file will have XML declaration with 'yes' or 'no'. Optionally specify if the file will be standalone with 'yes' or 'no'. Optionally specify public and system doctype modifiers.

Parameters:

input	The input.
path	Full pathname of the file.
encoding	The encoding (optional).
method	The method (optional).
version	The version (optional).
indent	Indent 'yes' or 'no' (optional).
omit_xml_declaration	Omit XML declaration 'yes' or 'no' (optional).
standalone	Standalone 'yes' or 'no' (optional).
doctype_public	The doctype public (optional).
doctype_system	The doctype system (optional).
cdata_section_elements	The cdata section elements (optional).
media_type	Type of the media (optional).

Write a node set to a file.

```
<xsl:variable name="VarResultAsXML">

...

</xsl:variable>

<xsl:variable name = "VarResult" select="msxsl:node-set($VarResultAsXML)/*" />
```

```
<xsl:variable name = "VarWriteDocument" select="wwexsldoc:Document($VarResult, 'C:
\badplacefor.xml')" />
```

XPathNodeIterator LoadXMLWithoutResolver (string *uriAsString [*, bool *preserveSpace]*)

Loads an XML file without resolving internal paths and validation DTDs.

Parameters:

uriAsString	The URI as string.
preserveSpace	True to preserve space (optional).

Returns:

A node set.

Load the page template without resolving attribute paths.

```
<xsl:variable name="VarPageTemplate"
 select="wwexsldoc:LoadXMLWithoutResolver($VarPathTemplatePath)" />
```

XPathNodeIterator LoadXMLWithResolver (string *uriAsString [*, bool *preserveSpace]*)

Loads an XML file while also resolving internal paths and validating DTDs.

Parameters:

uriAsString	The URI as string.
preserveSpace	True to preserve space (optional).

Returns:

A node set.

XPathNodeIterator MakeEmptyElement (object *input*)

Converts a non-empty XML node to an empty node.

Parameters:

input	The input node set.

Returns:

A node set containing a single empty node.

```
<img src="blue.jpg">

</img>
```

becomes:

```
<img src="blue.jpg" />
```

Insure <img> element is emitted as an empty XML node for proper display in HTML browsers.

```
<xsl:variable name="VarImageElementAsXML">

<xsl:element name = "img" >

 < xsl:attribute name = "src" >

   < xsl:value-of select = "'blue.jpg'" />

 </ xsl:attribute>

</xsl:element>

</xsl:variable>

<xsl:variable name = "VarImageElement" select="msxsl:node-set($VarImageElementAsXML)/*" />

<xsl:value-of select = "wwexsldoc:MakeEmeptyElement($VarImageElement)" />
```

Files

urn:WebWorks-XSLT-Extension-Files

Functions

bool UpToDate (string path, string projectChecksum, string groupID, string documentID, string actionChecksum)

Compares the provided parameters for a given path against a previously recorded values. If the values match, the result is true(). Otherwise, the result is false().

Detailed Description

urn:WebWorks-XSLT-Extension-Files

Enables incremental build support.

bool UpToDate (string *path*, string *projectChecksum*, string *groupID*, string *documentID*, string *actionChecksum*)

Compares the provided parameters for a given path against a previously recorded values. If the values match, the result is true(). Otherwise, the result is false().

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.
projectChecksum	The project checksum.
groupID	Identifier for the group.
documentID	Identifier for the document.
actionChecksum	The action checksum.

Returns:

True if it succeeds, false if it fails.

FileSystem

urn:WebWorks-XSLT-Extension-FileSystem

Functions

bool AppendFileWithFile (string targetPath, string sourcePath)

Appends one file to another file.

bool ChecksumUpToDate (string path, string checksum)

Compares the provided checksum with the current checksum.

string Combine (string path, string component1 [, string component2, string component3, string component4, string component5, string component6, string component7, string component8, string component9, string component10])

Combines the given component(s) to the file path.

XPathNodeIterator CopyDirectoryFiles (string sourceDirectoryPath, string destinationDirectoryPath)

Copies all files from the source directory to the destination directory. Destination files are reported as <FILE> elements. Source files are reported as <Depends> elements for each destination file.

XPathNodeIterator CopyFile (string sourcePath, string destinationPath)

Copies the source file to the destination path. Destination files are reported as <FILE> elements. Source files are reported as <Depends> elements for each destination file.

bool CreateDirectory (string path)

Creates a directory with the given path. If the directory already exists, the method will return false().

void DeleteDirectory (string path)

Recursively deletes the directory with the given path.

void DeleteFile (string path)

Deletes the file described by path.

bool DirectoryExists (string path)

Determines if a directory exists at the given path. If a file exists with the given path, this method will return false().

bool Exists (string path)

Determine if file or directory 'path' exists.

bool FileExists (string path)

Determines if a file exists at the given path. If directory exists with the given path, this method will return false().

bool FilesEqual (string alphaPath, string betaPath)

Compares the contents of two files to determine if they are equal.

string GetAbsoluteFrom (string relativePath, string referencePath)

Determines an absolute path given a relative path and a reference path to create the absolute path from. Returns the relative path argument if it is absolute to begin with.

string GetBaseName (string path)

Gets the base name of any path. It handles as separators: '\' and '/'. If the path ends with a separator or it's an empty string then it returns an empty string.

string GetChecksum (string path)

Determines the checksum for the specified file.

string GetDirectoryName (string path)

Determines the directory prefix of the given path.

string GetExtension (string path)

Determines the file extension for the given path.

string GetFileName (string path)

Determines the name of the file with the directory prefix removed.

string GetFileNameWithoutExtension (string path)

Determines the name of the file with the directory prefix and extension removed.

XPathNodeIterator GetFiles (string path)

Returns an XML node set containing absolute file paths to all files in the specified path. If the path specifies a file, a single file entry will be returned. If the path specifies a directory, all file paths in the directory and their children are returned.

string GetLongPathName (string path)

Gets long path name of a specified short path filename.

XPathNodeIterator GetRelativeFiles (string path)

Returns an XML node set containing RELATIVE file paths to all files in the specified path. If the path specifies a file, a single file entry will be returned. If the path specifies a directory, all file paths in the directory and their children are returned.

string GetRelativeTo (string path, string anchorPath)

Determines the relative path from the absolute anchor path to the absolute destination path.May return an absolute path if no relative path exists.

string GetShortPathName (string path)

Gets short path filename from a specified filename.

string GetTempFileName ()

Gets temporary unique filename path.

string GetTempPath ()

Gets path to the user's temporary files directory.

string GetWithExtensionReplaced (string path, string extension)

Replaces the current file extension with the provided one.

string MakeValidFileName (string fileNameSeed)

Makes a valid filename by eliminating the invalid characters from the specified seed filename.

void TranslateFileToEncoding (string sourceFilePath, string sourceFileEncodingName, string destinationFilePath, string destinationFileEncodingName)

Translate file to encoding.

Detailed Description

urn:WebWorks-XSLT-Extension-FileSystem

Allow XSL transforms to query and manipulate files and directories. Also handles system path parsing and processing.

bool AppendFileWithFile (string *targetPath*, string *sourcePath*)

Appends one file to another file.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

targetPath	Full pathname of the target file.
sourcePath	Full pathname of the source file.

Returns:

True if it succeeds, false if it fails.

Append the contents of "C:\FileSampleDoc.txt" to "C:\OutputAllContent.txt".

```
<xsl:variable name="VarADoc">C:\\File\Sample\Doc.txt</xsl:variable>

<xsl:variable name = "VarAllDocs" > C:\\Output\All\Content.txt</xsl:variable>

<xsl:variable name = "ActionAppendingDocToAllContent"
 select="wwfilesystem:AppendFileWithFile($VarAllDocs, $VarADoc)" />
```

bool ChecksumUpToDate (string *path*, string *checksum*)

Compares the provided checksum with the current checksum.

This is a convenience method for XSL developers.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.
checksum	The checksum.

Returns:

True if the same, false if not the same.

string Combine (string *path*, string *component1 [*, string *component2*, string *component3*, string *component4*, string *component5*, string *component6*, string *component7*, string *component8*, string *component9*, string *component10]*)

Combines the given component(s) to the file path.

Parameters:

path	Full pathname of the file.
component1	The first component.
component2 – component10	Additional components (optional).

Returns:

A string.

XPathNodeIterator CopyDirectoryFiles (string *sourceDirectoryPath*, string *destinationDirectoryPath*)

Copies all files from the source directory to the destination directory. Destination files are reported as <FILE> elements. Source files are reported as <Depends> elements for each destination file.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

sourceDirectoryPath	Pathname of the source directory.
destinationDirectoryPath	Pathname of the destination directory.

Returns:

A <Files></Files> node set.

XPathNodeIterator CopyFile (string *sourcePath*, string *destinationPath*)

Copies the source file to the destination path. Destination files are reported as <FILE> elements. Source files are reported as <Depends> elements for each destination file.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

sourcePath	Full pathname of the source file.
destinationPath	Full pathname of the destination file.

Returns:

A <Files> node set.</Files>

bool CreateDirectory (string *path*)

Creates a directory with the given path. If the directory already exists, the method will return false().

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

True if it succeeds, false if it fails.

Create directory C: if it does not exist.

```
<xsl:if test="wwfilesystem:Exists('C:\\exists')">

 <xsl:variable name = "VarCreateDirectory" select="wwfilesystem:CreateDirectory('C:\\exists')" />

</xsl:if>
```

void DeleteDirectory (string *path*)

Recursively deletes the directory with the given path.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the directory.

Delete the directory C:.

```
<xsl:variable name="VarDeleteDirectory" select="wwfilesystem:DeleteDirectory('C:\\deleteme')" />
```

void DeleteFile (string *path*)

Deletes the file described by path.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

bool DirectoryExists (string *path*)

Determines if a directory exists at the given path. If a file exists with the given path, this method will return false().

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

True if it succeeds, false if it fails.

Log existence of directory C:.

```
<xsl:if test="wwfilesystem:DirectoryExists('C:\\direxists')">

 <xsl:variable name = "VarLog" select="wwlog:Message('Directory \'', 'C:\\direxists', '\'
 exists!')" />

</xsl:if>
```

bool Exists (string *path*)

Determine if file or directory 'path' exists.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

True if it succeeds, false if it fails.

Determine if a file or directory exists at the given path.

```
<xsl:if test="wwfilesystem:Exists('C:\\exists')">

 <xsl:variable name = "VarCreateDirectory" select="wwfilesystem:CreateDirectory('C:\\exists')" />

</xsl:if>
```

bool FileExists (string *path*)

Determines if a file exists at the given path. If directory exists with the given path, this method will return false().

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

True if it succeeds, false if it fails.

Log existence of file C:.

```
<xsl:if test="wwfilesystem:DirectoryExists('C:\\fileexists')">

 <xsl:variable name = "VarLog" select="wwlog:Message('Directory \'', 'C:\\fileexists', '\'
 exists!')" />

</xsl:if>
```

bool FilesEqual (string *alphaPath*, string *betaPath*)

Compares the contents of two files to determine if they are equal.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

alphaPath	Full pathname of the first file.
betaPath	Full pathname of the second file.

Returns:

True if equal, false if not.

string GetAbsoluteFrom (string *relativePath*, string *referencePath*)

Determines an absolute path given a relative path and a reference path to create the absolute path from. Returns the relative path argument if it is absolute to begin with.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

relativePath	Pathname of the relative file.
referencePath	Full pathname of the reference file.

Returns:

The absolute from.

string GetBaseName (string *path*)

Gets the base name of any path. It handles as separators: '\' and '/'. If the path ends with a separator or it's an empty string then it returns an empty string.

Parameters:

path	The relative or absolute path.

Returns:

A base name string.

string GetChecksum (string *path*)

Determines the checksum for the specified file.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

The checksum as string.

string GetDirectoryName (string *path*)

Determines the directory prefix of the given path.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

The directory name.

string GetExtension (string *path*)

Determines the file extension for the given path.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

The filename extension.

string GetFileName (string *path*)

Determines the name of the file with the directory prefix removed.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

The file basename.

string GetFileNameWithoutExtension (string *path*)

Determines the name of the file with the directory prefix and extension removed.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

The file name without extension or directory prefix.

XPathNodeIterator GetFiles (string *path*)

Returns an XML node set containing absolute file paths to all files in the specified path. If the path specifies a file, a single file entry will be returned. If the path specifies a directory, all file paths in the directory and their children are returned.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file or directory.

Returns:

The files in a node set.

string GetLongPathName (string *path*)

Gets long path name of a specified short path filename.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

The long path filename.

XPathNodeIterator GetRelativeFiles (string *path*)

Returns an XML node set containing RELATIVE file paths to all files in the specified path. If the path specifies a file, a single file entry will be returned. If the path specifies a directory, all file paths in the directory and their children are returned.

Filename path(s) created will be relative to 'relativeToPath' (if non-zero in length).

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

The relative files in a node set.

string GetRelativeTo (string *path*, string *anchorPath*)

Determines the relative path from the absolute anchor path to the absolute destination path.May return an absolute path if no relative path exists.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.
anchorPath	Full pathname of the anchor file.

Returns:

The relative to path.

string GetShortPathName (string *path*)

Gets short path filename from a specified filename.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.

Returns:

The short path name.

string GetTempFileName ()

Gets temporary unique filename path.

Returns:

The temporary filename path.

string GetTempPath ()

Gets path to the user's temporary files directory.

Returns:

The directory path ending with backslash.

string GetWithExtensionReplaced (string *path*, string *extension*)

Replaces the current file extension with the provided one.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

path	Full pathname of the file.
extension	The extension.

Returns:

The with extension replaced.

string MakeValidFileName (string *fileNameSeed*)

Makes a valid filename by eliminating the invalid characters from the specified seed filename.

Parameters:

fileNameSeed	The file name seed.

Returns:

A filename string.

void TranslateFileToEncoding (string *sourceFilePath*, string *sourceFileEncodingName*, string *destinationFilePath*, string *destinationFileEncodingName*)

Translate file to encoding.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

sourceFilePath	Full pathname of the source file.
sourceFileEncodingName	Name of the source file encoding.
destinationFilePath	Full pathname of the destination file.
destinationFileEncodingName	Name of the destination file encoding.

Fonts

urn:WebWorks-XSLT-Extension-Fonts

Functions

bool UnicodeFont (string fontFamily)

Determines if the specified font family is a Unicode font family. Used to detect Symbol font families.

Detailed Description

urn:WebWorks-XSLT-Extension-Fonts

Answer questions about fonts that might affect format output.

bool UnicodeFont (string *fontFamily*)

Determines if the specified font family is a Unicode font family. Used to detect Symbol font families.

Parameters:

fontFamily	The font family.

Returns:

True if a unicode font, false if not.

Determine if Symbol is a unicode font family.

```
<xsl:variable name="VarIsUnicodeFont" select="wwfonts:UnicodeFont('Symbol')" />
```

Imaging

urn:WebWorks-Imaging-Info

Functions

XPathNodeIterator GetInfo (string imageFilePath)

Gets image information of file path: 'imageFilePath'.

void MapPDFLinks (object fileTable, string fileToFix, string fileToWrite, string originalFilePath, string outputFilePath, bool useAbsPath)

Map PDF links.

bool MergePDFs (object sourceFileList, string targetFilePath)

This merges a set of PDFs and/or PostScript files into one PDF.

bool MergePDFs (object sourceFileList, object fileTable, string targetFilePath)

This merges a set of PDFs and/or PostScript files into one PDF.

bool PostScriptToPDF (string postScriptFilePath, string pdfJobSettings, string pdfFilePath)

Convert the specified PostScript file to a PDF.

XPathNodeIterator RasterizePostScript (string postScriptFilePath, int renderHorizontalDPI, int renderVerticalDPI, int renderWidth, int renderHeight, string targetImageFormatAsString, int targetImageColorDepth, bool targetImageGrayscale, bool targetImageTransparent, bool targetImageInterlaced, int targetImageQuality, string targetFilePath)

Render a PostScript file to a known image format such as BMP, JPEG, PNG, or GIF.

XPathNodeIterator Transform (string inputImageFilePath, string outputImageFormat, int outputImageWidth, int outputImageHeight, string outputImageFilePath)

Create a new version of an image with a different format or with different dimensions.

XPathNodeIterator Transform (string inputImageFilePath, string outputImageFormatAsString, int Choice_outputImageQuality_outputImageWidth, int

Choice_outputImageWidth_outputImageHeight, string Choice_outputImageHeight_outputImageFilePath, string Choice_outputImageFilePath_outputResolution)

Create a new version of an image with a different format or with different dimensions.

XPathNodeIterator Transform (string inputImageFilePath, string outputImageFormatAsString, int outputImageQuality, int outputImageWidth, int outputImageHeight, string outputImageFilePath, int outputResolution)

Create a new version of an image with a different format or with different dimensions.

Detailed Description

urn:WebWorks-Imaging-Info

Enable processing of images within XSL transforms.

Returns information about a particular image file, including width and height, image format, bit-depth, path on system, etc.

```
<wwimageinfo:ImageInfo format="jpeg" width="200" height="300" bitdepth="32" grayscale="false"
 path="C:\\image.jpg" />
```

XPathNodeIterator GetInfo (string *imageFilePath*)

Gets image information of file path: 'imageFilePath'.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

imageFilePath	Full pathname of the image file.

Returns:

Image info node set.

void MapPDFLinks (object *fileTable*, string *fileToFix*, string *fileToWrite*, string *originalFilePath*, string *outputFilePath*, bool *useAbsPath*)

Map PDF links.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

fileTable	Node set of links to update.
fileToFix	The file to fix.
fileToWrite	The file to write.
originalFilePath	Full pathname of the original file.
outputFilePath	Full pathname of the output file.
useAbsPath	True to use abs path.

bool MergePDFs (object *sourceFileList*, string *targetFilePath*)

This merges a set of PDFs *and/or* PostScript files into one PDF.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

sourceFileList	List of source files.
targetFilePath	Full pathname of the target file.

Returns:

True if it succeeds, false if it fails.

bool MergePDFs (object *sourceFileList*, object *fileTable*, string *targetFilePath*)

This merges a set of PDFs *and/or* PostScript files into one PDF.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

sourceFileList	List of source files.
fileTable	Node set of links to update.
targetFilePath	Full pathname of the target file.

Returns:

True if it succeeds, false if it fails.

bool PostScriptToPDF (string *postScriptFilePath*, string *pdfJobSettings*, string *pdfFilePath*)

Convert the specified PostScript file to a PDF.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

postScriptFilePath	Full pathname of the post script file.
pdfJobSettings	The PDF job settings.
pdfFilePath	Full pathname of the PDF file.

Returns:

True if it succeeds, false if it fails.

XPathNodeIterator RasterizePostScript (string *postScriptFilePath*, int *renderHorizontalDPI*, int *renderVerticalDPI*, int *renderWidth*, int *renderHeight*, string *targetImageFormatAsString*, int *targetImageColorDepth*, bool *targetImageGrayscale*, bool *targetImageTransparent*, bool *targetImageInterlaced*, int *targetImageQuality*, string *targetFilePath*)

Render a PostScript file to a known image format such as BMP, JPEG, PNG, or GIF.

Parameters:

postScriptFilePath	Full pathname of the post script file.
renderHorizontalDPI	The render horizontal DPI.
renderVerticalDPI	The render vertical DPI.
renderWidth	Width of the render.
renderHeight	Height of the render.
targetImageFormatAsString	Target image format as string.
targetImageColorDepth	Depth of the target image color.
targetImageGrayscale	True to target image grayscale.
targetImageTransparent	True to target image transparent.
targetImageInterlaced	True if target image interlaced.
targetImageQuality	Target image quality.
targetFilePath	Full pathname of the target file.

Returns:

Image info node set.

XPathNodeIterator Transform (string *inputImageFilePath*, string *outputImageFormat*, int *outputImageWidth*, int *outputImageHeight*, string *outputImageFilePath*)

Create a new version of an image with a different format or with different dimensions.

Parameters:

inputImageFilePath	Full pathname of the input image file.
outputImageFormat	The output image format.
outputImageWidth	Width of the output image.
outputImageHeight	Height of the output image.
outputImageFilePath	Full pathname of the output image file.

Returns:

Image info node set.

XPathNodeIterator Transform (string *inputImageFilePath*, string *outputImageFormatAsString*, int *Choice_outputImageQuality_outputImageWidth*, int *Choice_outputImageWidth_outputImageHeight*, string *Choice_outputImageHeight_outputImageFilePath*, string *Choice_outputImageFilePath_outputResolution*)

Create a new version of an image with a different format or with different dimensions.

Parameters:

inputImageFilePath	Full pathname of the input image file.
outputImageFormatAsString	The output image format as string.
Choice_outputImageQuality_outputImageWidth	The choice of the output image quality output image.
Choice_outputImageWidth_outputImageHeight	The choice of the output image width output image.
Choice_outputImageHeight_outputImageFilePath	The choice of the output image height output image file.
Choice_outputImageFilePath_outputResolution	The choice of the output image file path output resolution.

Returns:

Image info node set.

XPathNodeIterator Transform (string *inputImageFilePath*, string *outputImageFormatAsString*, int *outputImageQuality*, int *outputImageWidth*, int *outputImageHeight*, string *outputImageFilePath*, int *outputResolution*)

Create a new version of an image with a different format or with different dimensions.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

inputImageFilePath	Full pathname of the input image file.
outputImageFormatAsString	The output image format as string.
outputImageQuality	The output image quality.
outputImageWidth	Width of the output image.
outputImageHeight	Height of the output image.
outputImageFilePath	Full pathname of the output image file.
outputResolution	The output resolution.

Returns:

Image info node set.

Log

urn:WebWorks-XSLT-Extension-Log

Functions

void Error (string message1 [, string message2, string message3, string message4, string message5, string message6, string message7, string message8, string message9, string message10])

Records error(s) to the generation log.

void Message (string message1 [, string message2, string message3, string message4, string message5, string message6, string message7, string message8, string message9, string message10])

Records message(s) to the generation log.

void Warning (string message1 [, string message2, string message3, string message4, string message5, string message6, string message7, string message8, string message9, string message10])

Records warning(s) to the generation log.

Detailed Description

urn:WebWorks-XSLT-Extension-Log

Enables XSL transforms to report messages, warnings, and errors to the generation log.

void Error (string *message1 [*, string *message2*, string *message3*, string *message4*, string *message5*, string *message6*, string *message7*, string *message8*, string *message9*, string *message10]*)

Records error(s) to the generation log.

Parameters:

message1	The first message.
message2 – message10	Additional messages (optional).

Express and Designer display a dialog informing the user errors were encountered during processing. AutoMap returns a non-zero return code.

```
<xsl:variable name="VarMessage" select="wwlog:Error('Put this in your log.')" />
```

void Message (string *message1 [*, string *message2*, string *message3*, string *message4*, string *message5*, string *message6*, string *message7*, string *message8*, string *message9*, string *message10]*)

Records message(s) to the generation log.

Parameters:

message1	The first message.
message2 – message10	Additional messages (optional).

void Warning (string *message1 [*, string *message2*, string *message3*, string *message4*, string *message5*, string *message6*, string *message7*, string *message8*, string *message9*, string *message10]*)

Records warning(s) to the generation log.

Parameters:

message1	The first message.
message2 – message10	Additional messages (optional).

MultiSearchReplaceExtension

urn:WebWorks-XSLT-Extension-MultiSearchReplace

Functions

void ReplaceAllInFile (string inputEncodingAsString, string inputFilePath, string outputFilePath, object replacements)

Replaces strings in a text file with the specified input encoding and writes the result to the output path with the same encoding.

void ReplaceAllInFile (string inputEncodingAsString, string inputFilePath, string outputEncodingAsString, string outputFilePath, object replacements)

Replaces strings in a text file with the specified input encoding and writes the result to the output path with the specified output encoding.

string ReplaceAllInString (string input, object replacements)

Replaces strings in the given string and returns the result. Using this method is much faster than performing multiple search/replace calls in XSL substring methods.

Detailed Description

urn:WebWorks-XSLT-Extension-MultiSearchReplace

Replaces multiple strings in a single operation.

void ReplaceAllInFile (string *inputEncodingAsString*, string *inputFilePath*, string *outputFilePath*, object *replacements*)

Replaces strings in a text file with the specified input encoding and writes the result to the output path with the same encoding.

Note: Not a replacement for using page templates when possible.

Parameters:

inputEncodingAsString	The input encoding as string.
inputFilePath	Full pathname of the input file.
outputFilePath	Full pathname of the output file.
replacements	The replacements in a node set.

void ReplaceAllInFile (string *inputEncodingAsString*, string *inputFilePath*, string *outputEncodingAsString*, string *outputFilePath*, object *replacements*)

Replaces strings in a text file with the specified input encoding and writes the result to the output path with the specified output encoding.

Note: Not a replacement for using page templates when possible.

Parameters:

inputEncodingAsString	The input encoding as string.
inputFilePath	Full pathname of the input file.
outputEncodingAsString	The output encoding as string.
outputFilePath	Full pathname of the output file.
replacements	The replacements in a node set.

string ReplaceAllInString (string *input*, object *replacements*)

Replaces strings in the given string and returns the result. Using this method is much faster than performing multiple search/replace calls in XSL substring methods.

Parameters:

input	The input.
replacements	The replacements in a node-set.

Returns:

A string.

NodeSet

urn:WebWorks-XSLT-Extension-NodeSet

Functions

XPathNodeIterator FirstUnique (object input, string attributeLocalName)

First unique element with matching local name.

XPathNodeIterator FirstUniqueWithNamespace (object input, string attributeLocalName, string attributeNamespaceURI)

First unique element with matching namespace and matching local name.

XPathNodeIterator LastUnique (object input, string attributeLocalName)

Last unique element with matching local name.

XPathNodeIterator LastUniqueWithNamespace (object input, string attributeLocalName, string attributeNamespaceURI)

Last unique element with matching namespace and matching local name.

Detailed Description

urn:WebWorks-XSLT-Extension-NodeSet

Miscellaneous node set functions.

XPathNodeIterator FirstUnique (object *input*, string *attributeLocalName*)

First unique element with matching local name.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

input	The input node set.
attributeLocalName	Local name of the element.

Returns:

A node set.

XPathNodeIterator FirstUniqueWithNamespace (object *input*, string *attributeLocalName*, string *attributeNamespaceURI*)

First unique element with matching namespace and matching local name.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

input	The input node set.
attributeLocalName	Local name of the element.
attributeNamespaceURI	Namespace URI.

Returns:

A node set.

XPathNodeIterator LastUnique (object *input*, string *attributeLocalName*)

Last unique element with matching local name.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

input	The input node set.
attributeLocalName	Local name of the element.

Returns:

A node set.

XPathNodeIterator LastUniqueWithNamespace (object *input*, string *attributeLocalName*, string *attributeNamespaceURI*)

Last unique element with matching namespace and matching local name.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

input	The input node set.
attributeLocalName	Local name of the element.
attributeNamespaceURI	Namespace URI.

Returns:

A node set.

Progress

urn:WebWorks-XSLT-Extension-Progress

Functions

bool Abort ()

Checks to see if the user has requested to abort the current operation.

void Cancel ()

Forces an abort to ocurr with Cancel as reason.

void End ()

End the current progress step.

void QueueAlert (string message)

Queue an alert to display.

void Retry ()

Forces an abort to ocurr with Retry as reason.

void SetStatus (string message)

Set the status bar for the current progress step.

void Start (int totalSubSteps)

Create a new progress step with the given number of sub-steps.

Detailed Description

urn:WebWorks-XSLT-Extension-Progress

Reports progress during long lived XSL transforms.

bool Abort ()

Checks to see if the user has requested to abort the current operation.

Returns:

True if abort requested, false otherwise.

void Cancel ()

Forces an abort to ocurr with Cancel as reason.

void End ()

End the current progress step.

void QueueAlert (string *message*)

Queue an alert to display.

Parameters:

message	Message to display.

void Retry ()

Forces an abort to ocurr with Retry as reason.

void SetStatus (string *message*)

Set the status bar for the current progress step.

Parameters:

message	Message to display.

```
<xsl:variable name="VarParagraphCount" select="count($VarParagraphs)" />

<xsl:variable name = "VarProgressParagraphsStart" select="wwprogress:Start($VarParagraphCount)" /
>

<xsl:variable name = "VarProgressParagraphsStatus"
 select="wwprogress:SetStatus(concat('Processing ', $VarParagraphCount))" />


<xsl:for-each select = "$VarParagraphs" >

 < xsl:variable name = "VarParagraph" select="." />


 <xsl:variable name = "VarProgressParagraphStart" select="wwprogress:Start(1)" />


 <xsl:variable name = "VarProgressParagraphStatus" select="wwprogress:SetStatus('Processing
 paragraph ', position(), ' of ', $VarParagraphCount, '.'))" />


 <xsl:variable name = "VarProgressParagraphEnd" select="wwprogress:End()" />

</xsl:for-each>

<xsl:variable name = "VarProgressParagraphsEnd" select="wwprogress:End()" />
```

void Start (int *totalSubSteps*)

Create a new progress step with the given number of sub-steps.

Parameters:

totalSubSteps	The total sub steps.

```
<xsl:variable name="VarProgressTotalStart" select="wwprogress:Start(2)" />

<xsl:variable name = "VarProgressStep1Start" select="wwprogress:Start(1)" />

<xsl:variable name = "VarProgressStep1End" select="wwprogress:End()" />

<xsl:variable name = "VarProgressStep2Start" select="wwprogress:Start(1)" />

<xsl:variable name = "VarProgressStep2End" select="wwprogress:End()" />

<xsl:variable name = "VarProgressTotalEnd" select="wwprogress:End()" />
```

Project

urn:WebWorks-XSLT-Extension-Project

Functions

bool DocumentExtension (string extension)

Determine if filename 'extension' has a configured input adapter.

bool GetConditionIsPassThrough (string conditionName)

Determines if condition called: 'conditionName' is pass through.

string GetConfigurationChangeID ()

Gets configuration change identifier.

XPathNodeIterator GetContextRule (string ruleTypeAsString, string ruleName, string documentID, string uniqueID)

Gets context rule.

string GetDocumentDataDirectoryPath (string documentID)

Gets document data directory path using 'documentID'.

string GetDocumentGroupPath (string documentID)

Gets document group path using 'documentID'.

string GetDocumentID (string documentPath [, string groupID])

Gets document identifier using 'documentPath' and optionally 'groupID'.

string GetDocumentPath (string documentID)

Gets document path using 'documentID'.

string GetDocumentsToGenerateChecksum ()

Get string representing all documents in project so checksum can be generated.

string GetFormatID ()

Gets format identifier.

string GetFormatName ()

Gets format name.

string GetFormatSetting (string name)

Gets 'name' format setting.

string GetFormatSetting (string name, string defaultValue)

Gets 'name' format setting but returns 'defaultValue' if no setting configured.

string GetGroupDataDirectoryPath (string groupID)

Gets group data directory path using 'groupID'.

string GetGroupName (string groupID)

Gets group name using 'groupID'.

XPathNodeIterator GetOverrideRule (string ruleTypeAsString, string ruleName, string documentID, string uniqueID)

Gets override rule.

string GetProjectDataDirectoryPath ()

Gets project data directory path.

string GetProjectDirectoryPath ()

Gets project directory path.

long GetProjectDocumentsCount ()

Gets project documents count.

string GetProjectFilesDirectoryPath ()

Gets project files directory path.

string GetProjectFormatDirectoryPath ()

Gets project format directory path.

string GetProjectName ()

Gets project name.

string GetProjectReportsDirectoryPath ()

Gets project reports directory path.

string GetProjectTargetName ()

Gets project target name currently being processed.

string GetProjectTargetOverrideDirectoryPath ()

Gets project target override directory path.

XPathNodeIterator GetRule (string ruleTypeAsString, string ruleName)

Gets a rule.

string GetTargetDataDirectoryPath ()

Gets target data directory path.

string GetTargetFilesInfoPath (string targetIDAsString)

Gets target files information path.

string GetTargetOutputDirectoryPath ()

Gets target output directory path.

string GetTargetReportsDirectoryPath ()

Gets target reports directory path.

Detailed Description

urn:WebWorks-XSLT-Extension-Project

Query for information about the currently running project.

bool DocumentExtension (string *extension*)

Determine if filename 'extension' has a configured input adapter.

Parameters:

extension	The extension.

Returns:

True if extension has a configured adapter, false otherwise.

bool GetConditionIsPassThrough (string *conditionName*)

Determines if condition called: 'conditionName' is pass through.

Parameters:

conditionName	Name of the condition.

Returns:

True if pass through, false otherwise.

string GetConfigurationChangeID ()

Gets configuration change identifier.

Returns:

The configuration change identifier.

XPathNodeIterator GetContextRule (string *ruleTypeAsString*, string *ruleName*, string *documentID*, string *uniqueID*)

Gets context rule.

Parameters:

ruleTypeAsString	The rule type as string.
ruleName	Name of the rule.
documentID	Identifier for the document.
uniqueID	Unique identifier.

Returns:

The context rule as a node set.

string GetDocumentDataDirectoryPath (string *documentID*)

Gets document data directory path using 'documentID'.

Parameters:

documentID	Identifier for the document.

Returns:

The document data directory path.

string GetDocumentGroupPath (string *documentID*)

Gets document group path using 'documentID'.

Parameters:

documentID	Identifier for the document.

Returns:

The document group path.

string GetDocumentID (string *documentPath [*, string *groupID]*)

Gets document identifier using 'documentPath' and optionally 'groupID'.

Parameters:

documentPath	Full pathname of the document file.
groupID	Identifier for the group (optional).

Returns:

The document identifier as string.

string GetDocumentPath (string *documentID*)

Gets document path using 'documentID'.

Parameters:

documentID	Identifier for the document.

Returns:

The document path.

string GetDocumentsToGenerateChecksum ()

Get string representing all documents in project so checksum can be generated.

Returns:

A string representing documents for generating checksum.

string GetFormatID ()

Gets format identifier.

Returns:

The format identifier as string.

string GetFormatName ()

Gets format name.

Returns:

The format name as string.

string GetFormatSetting (string *name*)

Gets 'name' format setting.

Parameters:

name	The name.

Returns:

The format setting as string.

string GetFormatSetting (string *name*, string *defaultValue*)

Gets 'name' format setting but returns 'defaultValue' if no setting configured.

Parameters:

name	The name.
defaultValue	The default value.

Returns:

The format setting as string.

string GetGroupDataDirectoryPath (string *groupID*)

Gets group data directory path using 'groupID'.

Parameters:

groupID	Identifier for the group.

Returns:

The group data directory path.

string GetGroupName (string *groupID*)

Gets group name using 'groupID'.

Parameters:

groupID	Identifier for the group.

Returns:

The group name as string.

XPathNodeIterator GetOverrideRule (string *ruleTypeAsString*, string *ruleName*, string *documentID*, string *uniqueID*)

Gets override rule.

Parameters:

ruleTypeAsString	The rule type as string.
ruleName	Name of the rule.
documentID	Identifier for the document.
uniqueID	Unique identifier.

Returns:

The override rule node set.

string GetProjectDataDirectoryPath ()

Gets project data directory path.

Returns:

The project data directory path.

string GetProjectDirectoryPath ()

Gets project directory path.

Returns:

The project directory path.

long GetProjectDocumentsCount ()

Gets project documents count.

Returns:

The project documents count.

string GetProjectFilesDirectoryPath ()

Gets project files directory path.

Returns:

The project files directory path.

string GetProjectFormatDirectoryPath ()

Gets project format directory path.

Returns:

The project format directory path.

string GetProjectName ()

Gets project name.

Returns:

The project name.

string GetProjectReportsDirectoryPath ()

Gets project reports directory path.

Returns:

The project reports directory path.

string GetProjectTargetName ()

Gets project target name currently being processed.

Returns:

The project target name.

string GetProjectTargetOverrideDirectoryPath ()

Gets project target override directory path.

Returns:

The project target override directory path.

XPathNodeIterator GetRule (string *ruleTypeAsString*, string *ruleName*)

Gets a rule.

Parameters:

ruleTypeAsString	The rule type as string.
ruleName	Name of the rule.

Returns:

The rule as a node set.

string GetTargetDataDirectoryPath ()

Gets target data directory path.

Returns:

The target data directory path.

string GetTargetFilesInfoPath (string *targetIDAsString*)

Gets target files information path.

Parameters:

targetIDAsString	Target identifier as string.

Returns:

The target files information path.

string GetTargetOutputDirectoryPath ()

Gets target output directory path.

Returns:

The target output directory path.

string GetTargetReportsDirectoryPath ()

Gets target reports directory path.

Returns:

The target reports directory path.

StageInfo

urn:WebWorks-XSLT-Extension-StageInfo

Functions

string Get (string param_key)

Gets the value of a given key for this stage.

void Set (string param_key, string param_value)

Set key/value pair for this stage.

Detailed Description

urn:WebWorks-XSLT-Extension-StageInfo

Allows XLST processing to store and retrieve key/value pairs as needed to track state.

string Get (string *param_key*)

Gets the value of a given key for this stage.

Parameters:

param_key	The key to lookup.

Returns:

The value as string.

void Set (string *param_key*, string *param_value*)

Set key/value pair for this stage.

Parameters:

param_key	The key to set.
param_value	The value to set.

StringUtilities

urn:WebWorks-XSLT-Extension-StringUtilities

Functions

string CSSClassName (string styleName)

Convert the given string into a valid CSS class name.

string DecodeURI (string value)

Decode an escaped URI 'value' back to an unescaped URI.

string DecodeURIComponent (string value)

Decode an escaped partial URI component, 'value', back to an unescaped URI component.

string EclipseId (string identifier)

Create a valid Eclipse ID from 'identifier'.

string EncodeURI (string value)

Encode 'value' string as an escaped URI.

string EncodeURIComponent (string value)

Encode 'value' string, a partial URI component, as an escaped URI.

string EscapeForXMLAttribute (string value)

Escape 'value' string so that it can be written as an XML attribute.

string Format (string format, string argument1 [, string argument2, string argument3, string argument4, string argument5, string argument6, string argument7, string argument8, string argument9, string argument10])

Format a message using the C# string formatter.

string FromFile (string sourceFilePath, string sourceFileEncodingName)

Import file contents using 'sourceFileEncodingName' as the file's assumed encoding.

string JavaScriptEncoding (string value)

Convert all non-ASCII characters to Unicode escape sequences. Also convert all ASCII characters less than 32 along with problematic escape characters, i.e. \, to Unicode escape sequences.

bool MatchExpression (string input, string matchExpressionAsString)

Return success of match for 'matchExpressionAsString' in 'input'.

string MatchExpressionValue (string input, string matchExpressionAsString)

Return value of match for 'matchExpressionAsString' in 'input'.

string MD5Checksum (string value)

Compute the MD5 checksum on the given 'value' string.

string NCNAME (string identifier)

Convert 'identifier' to a valid NCNAME as defined by:

string NormalizeQuotes (string value)

Convert the given string to a string where all left/right single/double quotes are normalized. Left single quotation mark = single quotation mark Right single quotation mark = single quotation mark Left double quotation mark = double quotation mark Right double quotation mark = double quotation mark

string OEBClassName (string styleName)

Create a valid Open eBook class name from 'styleName'.

string Replace (string input, string search, string replacement)

Replace all occurrences of 'search' in 'input' with 'replacement'.

string ReplaceWithExpression (string input, string searchExpressionAsString, string replacement)

Replace all occurrences of 'searchExpressionAsString' in 'input' with 'replacement'.

string ReplaceWithExpressionForCount (string input, string searchExpressionAsString, string replacement, int count)

Replace 'count' occurrences of 'searchExpressionAsString' in 'input' with 'replacement'.

string SHA1Checksum (string value)

Compute the SHA-1 (Secure Hash Algorithm 1) checksum on the given 'value' string.

string ToLower (string value)

Convert the given string to lowercase.

string ToUpper (string value)

Convert the given string to uppercase.

string ToCamel (string value)

Convert the given string to camel case.

string ToPascal (string value)

Convert the given string to pascal case.

bool EndsWith (string input, string suffix)

Return success of suffix being the suffix of input.

string WebWorksHelpContextOrTopic (string key)

Converts the given 'key' into a valid WebWorks Help/Reverb context or topic string.

Detailed Description

urn:WebWorks-XSLT-Extension-StringUtilities

Extend the available string processing methods to XSL to include message formatting, specialized text escaping, regular expression operations, etc.

string CSSClassName (string *styleName*)

Convert the given string into a valid CSS class name.

Parameters:

styleName	Name of the style.

Returns:

A string.

Convert Blue_Moon.Detective;Agency into a valid CSS style name.

```
<xsl:value-of select="wwstring:CSSClassName('Blue_Moon.Detective;Agency')" />
```

string DecodeURI (string *value*)

Decode an escaped URI 'value' back to an unescaped URI.

Parameters:

value	The value.

Returns:

A string.

string DecodeURIComponent (string *value*)

Decode an escaped partial URI component, 'value', back to an unescaped URI component.

Parameters:

value	The value.

Returns:

A string.

string EclipseId (string *identifier*)

Create a valid Eclipse ID from 'identifier'.

Parameters:

identifier	The identifier.

Returns:

A string.

string EncodeURI (string *value*)

Encode 'value' string as an escaped URI.

Parameters:

value	The value.

Returns:

A string.

string EncodeURIComponent (string *value*)

Encode 'value' string, a partial URI component, as an escaped URI.

Parameters:

value	The value.

Returns:

A string.

string EscapeForXMLAttribute (string *value*)

Escape 'value' string so that it can be written as an XML attribute.

Parameters:

value	The value.

Returns:

A string.

Write onClick handler for tag.

```
<html:div onClick="{wwstring:EscapeForXMLAttribute('alert(\'Boo!\');')}">

  Click me!

</html:div>
```

string Format (string *format*, string *argument1 [*, string *argument2*, string *argument3*, string *argument4*, string *argument5*, string *argument6*, string *argument7*, string *argument8*, string *argument9*, string *argument10]*)

Format a message using the C# string formatter.

Parameters:

format	Describes the format to use.
argument1	The first argument.
argument2 – argument10	Additional arguments (optional).

Returns:

The formatted value.

Create the message: '17 total'.

```
<xsl:value-of select="wwstring:Format('{0} total', 17)" />
```

Create the message: '17 total, 15 of 17'.

```
<xsl:value-of select="wwstring:Format('{0} total, {1} of {0}.', 17, 15)" />
```

string FromFile (string *sourceFilePath*, string *sourceFileEncodingName*)

Import file contents using 'sourceFileEncodingName' as the file's assumed encoding.

Parameters:

sourceFilePath	Full pathname of the source file.
sourceFileEncodingName	Name of the source file encoding.

Returns:

A string.

Import the contents of file: 'C:.txt' which was encoded using UTF-8.

```
<xsl:value-of select="wwstring:FromFile('C:\myfile.txt', 'UTF-8')" />
```

string JavaScriptEncoding (string *value*)

Convert all non-ASCII characters to Unicode escape sequences. Also convert all ASCII characters less than 32 along with problematic escape characters, i.e. \, to Unicode escape sequences.

Parameters:

value	The value.

Returns:

A string.

Convert Hello!

to JavaScript encoded text.

```
<xsl:value-of select="wwstring:JavaScriptEncoding('Hello\nworld!\n')" />
```

bool MatchExpression (string *input*, string *matchExpressionAsString*)

Return success of match for 'matchExpressionAsString' in 'input'.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

input	The input.
matchExpressionAsString	The match expression as string.

Returns:

True if match found, false otherwise.

Contains 3-4 "a"s?

```
<xsl:value-of select="wwstring:Replace('<letter>scar's <letter>nly <letter>strich <letter>iled an
 <letter>range <letter>wl t<letter>day.', '<letter>', 'o')" />
```

string MatchExpressionValue (string *input*, string *matchExpressionAsString*)

Return value of match for 'matchExpressionAsString' in 'input'.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

input	The input.
matchExpressionAsString	The match expression as string.

Returns:

String.

Value of 3-4 "a"s.

```
<xsl:value-of select="wwstring:MatchExpressionValue('The end of aa sentenceaaaaalways leaves me
 sad.', 'a{3-4}')" />
```

string MD5Checksum (string *value*)

Compute the MD5 checksum on the given 'value' string.

Parameters:

value	The value.

Returns:

A string.

Determine the MD5 signature for A long time ago, way back in history, when all there was to drink, was nothing but cups 'o tea..

```
<xsl:value-of select="wwstring:MD5Checksum('A long time ago, way back in history, when all there
 was to drink, was nothing but cups \'o tea.')" />
```

string NCNAME (string *identifier*)

Convert 'identifier' to a valid NCNAME as defined by:

http://www.w3.org/TR/1999/REC-xml-names-19990114/#NT-NCNameChar.

Parameters:

identifier	The identifier.

Returns:

A string.

Create an NCName from 'id' attribute.

```
<xsl:value-of select="wwstring:NCNAME($VarTable/@id)" />
```

string NormalizeQuotes (string *value*)

Convert the given string to a string where all left/right single/double quotes are normalized. Left single quotation mark = single quotation mark Right single quotation mark = single quotation mark Left double quotation mark = double quotation mark Right double quotation mark = double quotation mark

Parameters:

value	The value.

Returns:

String.

```
<xsl:value-of select="wwstring:NormalizeQuotes('string-with-left-right-quote-marks...')" />
```

string OEBClassName (string *styleName*)

Create a valid Open eBook class name from 'styleName'.

Parameters:

styleName	Name of the style.

Returns:

A string.

string Replace (string *input*, string *search*, string *replacement*)

Replace all occurrences of 'search' in 'input' with 'replacement'.

Parameters:

input	The input.
search	The search.
replacement	The replacement.

Returns:

Result String.

Replace all instances of <letter> with O.

```
<xsl:value-of select="wwstring:Replace('<letter>scar's <letter>nly <letter>strich <letter>iled an
 <letter>range <letter>wl t<letter>day.', '<letter>', 'o')" />
```

string ReplaceWithExpression (string *input*, string *searchExpressionAsString*, string *replacement*)

Replace all occurrences of 'searchExpressionAsString' in 'input' with 'replacement'.

Parameters:

input	The input.
searchExpressionAsString	The search expression as string.
replacement	The replacement.

Returns:

String.

Replace runs of 3-4 "a"s with ". ".

```
<xsl:value-of select="wwstring:ReplaceWithExpression('The end of aa sentenceaaaaalways leaves me
 sad.', 'a{3-4}', '. ')" />
```

string ReplaceWithExpressionForCount (string *input*, string *searchExpressionAsString*, string *replacement*, int *count*)

Replace 'count' occurrences of 'searchExpressionAsString' in 'input' with 'replacement'.

Parameters:

input	The input.
searchExpressionAsString	The search expression as string.
replacement	The replacement.
count	Number of.

Returns:

A string.

Replace first run of 3-4 "a"s with ". ".

```
<xsl:value-of select="wwstring:ReplaceWithExpression('The end of aa sentenceaaaaalways leaves me
 sad.', 'a{3-4}', '. ', 1)" />
```

string SHA1Checksum (string *value*)

Compute the SHA-1 (Secure Hash Algorithm 1) checksum on the given 'value' string.

Parameters:

value	The value.

Returns:

A string.

Determine the SHA1 signature for A long time ago, way back in history, when all there was to drink, was nothing but cups 'o tea..

```
<xsl:value-of select="wwstring:SHA1Checksum('A long time ago, way back in history, when all there
 was to drink, was nothing but cups \'o tea.')" />
```

string ToLower (string *value*)

Convert the given string to lowercase.

Parameters:

value	The value.

Returns:

String.

```
<xsl:value-of select="wwstring:ToLower('UppERcAse')" />
```

string ToUpper (string *value*)

Convert the given string to uppercase.

Parameters:

value	The value.

Returns:

String.

```
<xsl:value-of select="wwstring:ToUpper('lOwErcAsE')" />
```

string ToCamel (string *value*)

Convert the given string to camel case.

Parameters:

value	The value.

Returns:

String.

```
<xsl:value-of select="wwstring:ToCamel('camel case')" />
```

string ToPascal (string *value*)

Convert the given string to pascal case.

Parameters:

value	The value.

Returns:

String.

```
<xsl:value-of select="wwstring:ToPascal('pascal case')" />
```

bool EndsWith (string *input*, string *suffix*)

Return success of suffix being the suffix of input.

Parameters:

input	The input.
suffix	The suffix.

Returns:

Bool.

Check if ".scss" is suffix of "webworks.scss". This returns True.

```
<xsl:value-of select="wwstring:EndsWith('webworks.scss', '.scss')" />
```

string WebWorksHelpContextOrTopic (string *key*)

Converts the given 'key' into a valid WebWorks Help/Reverb context or topic string.

Note: WebWorks Help/Reverb context and topic strings may only contain the characters A-Z, a-z, 0-9, and _.

Parameters:

key	The key.

Returns:

A string.

Convert A long time... into a valid WebWorks Help context or topic name.

```
<xsl:value-of select="wwstring:WebWorksHelpContextOrTopic('A long time...')" />
```

Units

urn:WebWorks-XSLT-Extension-Units

Functions

double Convert (double sourceValue, string sourceUnits, string targetUnits)

Convert a measurement from one set of units to another.

string CSSRGBColor (string htmlColor)

Convert the given HTML/CSS color to a hex encoded CSS color. Useful for converting named colors such as green to their hex equivalents.

string EncodingFromCodePage (int codePage)

Determine the HTML encoding for a page given a Windows code page value.

string NumericPrefix (string value)

Extract the numeric prefix of 'value'.

string RTFColor (string htmlColor)

Convert a standard HTML/CSS color to an RTF color.

string UnitsSuffix (string value)

Extract the units suffix of 'value'. Only returns a non-zero length string of there is also a numeric prefix.

Detailed Description

urn:WebWorks-XSLT-Extension-Units

Utility methods for extracting units and value from raw strings along with unit-to-unit conversion routines.

double Convert (double *sourceValue*, string *sourceUnits*, string *targetUnits*)

Convert a measurement from one set of units to another.

Parameters:

sourceValue	Source value.
sourceUnits	Source units.
targetUnits	Target units.

Returns:

A number.

Convert 2in to centimeters (cm).

```
<xsl:variable name="VarCentimeters" select="wwunits:Convert(wwunits:NumericPrefix('2in'),
 wwunits:UnitsSuffix('2in'), 'cm')" />
```

string CSSRGBColor (string *htmlColor*)

Convert the given HTML/CSS color to a hex encoded CSS color. Useful for converting named colors such as green to their hex equivalents.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

htmlColor	The HTML color.

Returns:

Hex encoded CSS color as string.

Convert green to the hex equivalent.

```
<xsl:variable name="VarGreenAsRGB" select="wwunits:CSSRGBColor('green')" />
```

string EncodingFromCodePage (int *codePage*)

Determine the HTML encoding for a page given a Windows code page value.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

codePage	The code page.

Returns:

A string.

Determine the HTML encoding for code page 23.

```
<xsl:variable name="VarEncoding" select="wwunits:EncodingFromCodePage(23)" />
```

string NumericPrefix (string *value*)

Extract the numeric prefix of 'value'.

Parameters:

value	The value.

Returns:

A string.

Determine numeric prefix of 24px.

```
<xsl:variable name="VarNumber" select="wwunits:NumericPrefix('24px')" />
```

string RTFColor (string *htmlColor*)

Convert a standard HTML/CSS color to an RTF color.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

htmlColor	The HTML color.

Returns:

RTF color as string.

Convert #FF3399 to an RTF color.

```
<xsl:varaible name="VarRTFColor" select="wwunits:RTFColor('#FF3399')" />
```

string UnitsSuffix (string *value*)

Extract the units suffix of 'value'. Only returns a non-zero length string of there is also a numeric prefix.

Parameters:

value	The value.

Returns:

A string.

Determine units of 24px.

```
<xsl:variable name="VarNumber" select="wwunits:UnitsSuffix('24px')" />
```

URI

urn:WebWorks-XSLT-Extension-URI

Functions

string AsFilePath (string uriAsString)

Converts an uriAsString to a file path.

string AsURI (string filePathAsString)

Converts a file path to an uriAsString.

string EscapeData (string unescapedString)

Convert unescaped string into an escaped URI data.

string EscapeUri (string unescapedUri)

Convert unescaped URI into an escaped URI path.

string GetRelativeTo (string uriAsString, string anchorUriAsString)

Convert an absolute URI into a relative URI.

bool IsFile (string uriAsString)

Determine if the supplied URI refers to the file system.

string MakeAbsolute (string absoluteUriAsString, string uriAsString)

Convert a relative URI into an absolute URI. If the second parameter is already an absolute URI, it will be returned unchanged.

XPathNodeIterator PossibleResolvedUris (string uriAsString)

Convert file paths to URIs.

string Unescape (string escapedString)

Convert URI escaped string into unescaped string (%20 to space, etc.).

Detailed Description

urn:WebWorks-XSLT-Extension-URI

Utility methods which convert to and from file paths and create absolute or relative URIs.

string AsFilePath (string *uriAsString*)

Converts an uriAsString to a file path.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

uriAsString	The URI as string.

Returns:

File path as string.

Convert URIs (i.e. **file:///network/filesystem/file**) to file paths.

```
<xsl:variable name="VarFilePath1" select="wwuri:AsFilePath('\\\\network\\filesystem\\file')">


<xsl:variable name = "VarFilePath2" select="wwuri:AsFilePath('file:///network/filesystem/file')">
```

string AsURI (string *filePathAsString*)

Converts a filePathAsString to a URI.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

filePathAsString	The file path as string.

Returns:

URI as string.

Convert file paths (i.e. C:\file) to URI strings (i.e. file:///C:/file).

```
<xsl:variable name="VarURI" select="wwuri:AsURI('C:\\file')">
```

string EscapeData (string *unescapedString*)

Convert unescaped string into an escaped URI data.

Parameters:

unescapedString	The unescaped string.

Returns:

Escaped string.

Convert http://www.webworks.com/name with spaces.html to http%3a%2f%2fwww.webworks.com%2fname%20with%20spaces.html.

```
<xsl:variable name="VarEscapedData" select="wwuri:EscapeData('http://www.webworks.com/name with
 spaces.html')" />
```

string EscapeUri (string *unescapedUri*)

Convert unescaped URI into an escaped URI path.

Parameters:

unescapedUri	URI of the unescaped.

Returns:

Escaped URI as a string.

Convert http://www.webworks.com/name with spaces.html to http://www.webworks.com/name%20with%20spaces.html.

```
<xsl:variable name="VarEscapedURI" select="wwuri:EscapeUri('http://www.webworks.com/name with
 spaces.html')" />
```

string GetRelativeTo (string *uriAsString*, string *anchorUriAsString*)

Convert an absolute URI into a relative URI.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

uriAsString	The URI as string.
anchorUriAsString	The anchor URI as string.

Returns:

The relative to.

Determine the relative path to http://www.webworks.com/css/webworks.css from http://www.webworks.com/information/index.html.

```
<xsl:variable name="VarRelativeURI" select="wwuri:GetRelativeTo('http://www.webworks.com/css/
webworks.css', 'http://www.webworks.com/information/index.html')" />
```

bool IsFile (string *uriAsString*)

Determine if the supplied URI refers to the file system.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

uriAsString	The URI as string.

Returns:

True if file, false if not.

Determine if \network and file:///network/filesystem/file are file URIs.

```
<xsl:if test="wwuri:IsFile('\\\\network\\filesystem\\file')">

  <xsl:variable name = "VarLog" select="wwlog:Message('File!')" />

</xsl:if>


<xsl:if test="wwuri:IsFile('file:///network/filesystem/file')">

  <xsl:variable name = "VarLog" select="wwlog:Message('File!')" />

</xsl:if>
```

string MakeAbsolute (string *absoluteUriAsString*, string *uriAsString*)

Convert a relative URI into an absolute URI. If the second parameter is already an absolute URI, it will be returned unchanged.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

absoluteUriAsString	The absolute URI as string.
uriAsString	The URI as string.

Returns:

URI as string.

Fully qualify the relative URI ../css/webworks.css against the absolute URI http://www.webworks.com/information/index.html.

```
<xsl:variable name="VarCSSURI" select="wwuri:MakeAbsolute('http://www.webworks.com/information/
index.html', '../css/webworks.css')" />
```

XPathNodeIterator PossibleResolvedUris (string *uriAsString*)

Convert file paths to URIs.

Exceptions:

OutOfMemoryException	Thrown when a low memory situation occurs.

Parameters:

uriAsString	The URI as string.

Returns:

A node set of possible resolved URIs for the given virtual URI.

```
<wwuri:Uris>

  <wwuri:Uri value = "file:///C:/Users/user/Projects/Targets/Reverb/Transforms/pages.xsl" />

  < wwuri:Uri value = "file:///C:/Users/user/Projects/Formats/WebWorks Reverb/Transforms/
pages.xsl" />

  < wwuri:Uri value = "file:///C:/Program Files (x86)/WebWorks/ePublisher/2012.4/Formats/WebWorks
 Reverb/Transforms/pages.xsl" />

</ wwuri:Uris>
```

Resolve virtual URI wwformat:Transforms/pages.xsl.

```
<xsl:variable name="VarPossibleURIs" select="wwuri:PossibleResolvedUris('wwformat:Transforms/
pages.xsl')" />

<xsl:for-each select = "$VarPossibleURIs/wwuri:Uris/wwuri:Uri" >

  <xsl:variable name = "VarPossibleURI" select="." />


  <xsl:if test="wwuri:IsFile($VarPossibleURI)">

    <xsl:variable name = "VarPossibleFilePath" select="wwuri:AsFilePath($VarPossibleURI)" />

    <xsl:if test="wwfilesystem:FileExists($VarPossibleFilePath)">

      ...

    </xsl:if>

  </xsl:if>

</xsl:for-each>
```

string Unescape (string *escapedString*)

Convert URI escaped string into unescaped string (%20 to space, etc.).

Parameters:

escapedString	The escaped string.

Returns:

Unescaped string.

Convert name%20with%20spaces.html to name with spaces.html.

```
<xsl:variable name="VarUnescape" select="wwuri:Unescape('name%20with%20spaces.html')" />
```

ZipExtension

urn:WebWorks-XSLT-Extension-Zip

Functions

void Zip (string zipFilePath, object nodes)

Create a zip archive containing a list of files.

void ZipAdd (string zipFilePath, object nodes)

Add a list of files to a zip archive.

void ZipAddWithoutCompression (string zipFilePath, object nodes)

Add a list of files to a zip archive without compressing any files.

void ZipDirectory (string zipFilePath, string directoryPath)

Zip a directory and recursively include sub-directories without compressing any files.

void ZipDirectoryWithoutCompression (string zipFilePath, string directoryPath)

Zip a directory and recursively include sub-directories without compressing any files.

void ZipExtract (string zipFilePath, string targetDirectory)

Extract contents of a zip archive to a specified directory location.

void ZipWithoutCompression (string zipFilePath, object nodes)

Create a zip archive containing a list of files without compressing any files.

Detailed Description

urn:WebWorks-XSLT-Extension-Zip

Allow XSL transforms to handle zip archives.

void Zip (string *zipFilePath*, object *nodes*)

Create a zip archive containing a list of files.

Parameters:

zipFilePath	Full pathname of the zip file.
nodes	The node set.

Creates a zip archive C:.zip with the contents defined in the node set.

```
<xsl:variable name="VarZipListAsXML">

  <wwzip:File source = "C:\some\directory\alpha.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\beta.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\gamma.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\last\gamma.txt" zip-directory="directory\last" />

</xsl:variable>

<xsl:variable name = "VarZipList" select="wwzip:Zip('C:\some\archive.zip', msxsl:node-
set($VarZipListAsXML)/*" />
```

void ZipAdd (string *zipFilePath*, object *nodes*)

Add a list of files to a zip archive.

Parameters:

zipFilePath	Full pathname of the zip file.
nodes	The node set.

Append to zip archive C:.zip with the contents defined in the node set.

```
<xsl:variable name="VarZipListAsXML">

  <wwzip:File source = "C:\some\directory\alpha.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\beta.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\gamma.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\last\gamma.txt" zip-directory="directory\last" />

</xsl:variable>

<xsl:variable name = "VarZipList" select="wwzip:ZipAdd('C:\some\archive.zip', msxsl:node-
set($VarZipListAsXML)/*" />
```

void ZipAddWithoutCompression (string *zipFilePath*, object *nodes*)

Add a list of files to a zip archive without compressing any files.

Parameters:

zipFilePath	Full pathname of the zip file.
nodes	The node set.

Append to zip archive C:.zip with the contents defined in the node set.

```
<xsl:variable name="VarZipListAsXML">

  <wwzip:File source = "C:\some\directory\alpha.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\beta.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\gamma.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\last\gamma.txt" zip-directory="directory\last" />

  </xsl:variable>

<xsl:variable name = "VarZipList" select="wwzip:ZipAddWithoutCompression('C:\some\archive.zip',
 msxsl:node-set($VarZipListAsXML)/*" />
```

void ZipDirectory (string *zipFilePath*, string *directoryPath*)

Zip a directory and recursively include sub-directories without compressing any files.

Parameters:

zipFilePath	Full pathname of the zip file.
directoryPath	Full pathname of the directory.

Creates zip archive C:.zip with the directory and and subdirectories of 'C:\ projects'.

```
<xsl:variable name="VarZipDirectory" select="wwzip:ZipDirectory('C:\some\archive.zip', 'C:
\projects\myproject'" />
```

void ZipDirectoryWithoutCompression (string *zipFilePath*, string *directoryPath*)

Zip a directory and recursively include sub-directories without compressing any files.

Parameters:

zipFilePath	Full pathname of the zip file.
directoryPath	Full pathname of the directory file.

Creates zip archive C:.zip with the directory and and subdirectories of C:\ projects.

```
<xsl:variable name="VarZipDirectory" select="wwzip:ZipDirectoryWithoutCompression('C:\some
\archive.zip', 'C:\projects\myproject'" />
```

void ZipExtract (string *zipFilePath*, string *targetDirectory*)

Extract contents of a zip archive to a specified directory location.

Parameters:

zipFilePath	Full pathname of the zip file.
targetDirectory	Pathname of the target directory.

Extracts the contents of zip archive C:.zip to C:.

```
<xsl:variable name="VarZipExtract" select="wwzip:ZipExtract('C:\some\archive.zip', 'C:\projects
\myproject'" />
```

void ZipWithoutCompression (string *zipFilePath*, object *nodes*)

Create a zip archive containing a list of files without compressing any files.

Parameters:

zipFilePath	Full pathname of the zip file.
nodes	The nodes.

Creates a zip archive C:.zip with the contents defined in the node set.

```
<xsl:variable name="VarZipListAsXML">

  <wwzip:File source = "C:\some\directory\alpha.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\beta.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\gamma.txt" zip-directory="directory" />

  <wwzip:File source = "C:\some\directory\last\gamma.txt" zip-directory="directory\last" />

</xsl:variable>

<xsl:variable name = "VarZipList" select="wwzip:ZipWithoutCompression('C:\some\archive.zip',
 msxsl:node-set($VarZipListAsXML)/*" />
```