ePublisher

WebWorks ePublisher Platform

Developer Guide

Version 2018.1

webworks®

Technical Communication Software

Table of Contents

WebWorks ePublisher Documentation

WebWorks ePublisher Documentation provides many resources online and in the product to help you use the ePublisher system to deliver integrated, single-sourced document sets.

Dockable Help

Provides context-sensitive information and step-by-step guidance for common tasks, as well as descriptions of each field on each window. Dockable Help is the ePublisher documentation set in Reverb 2.0 format and provides a comprehensive, integrated deliverable that includes all the information from WebWorks resources. Dockable Help displays when ePublisher is opened, or you can click **Documentation** on the **Help** tab in the product interface.

Contacting Quadralay

Please contact us with your questions and comments. We look forward to hearing from you. If you need assistance with an issue, please contact Technical Support. The Support Web site allows you to review the support policy and create a case to track an issue.

Telephone:	877.893.2967 (only in the United States and Canada) 512.719.3399
Email:	info@webworks.com
Support	www.webworks.com/Support/
Web Site:	www.webworks.com

Conventions

WebWorks ePublisher documentation uses consistent conventions to help you identify items. The following table summarizes these conventions.

Convention	Use
Bold	• Window and menu items • Technical terms, when introduced
Italics	• Book titles • Variable names • Emphasized words
`Fixed Font`	• File and folder names • Commands and code examples • Text you must type • Text (output) displayed in the command-line interface
Blue	• Links
>	• Submenu selections, such as **Target > Target Settings**
Brackets, such as `[`*`value`*`]`	• Optional parameters of a command
Braces, such as `{`*`value`*`}`	• Required parameters of a command
Logical OR, such as *`value1`* `|` *`value2`*	• Exclusive parameters. Choose one parameter.

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:

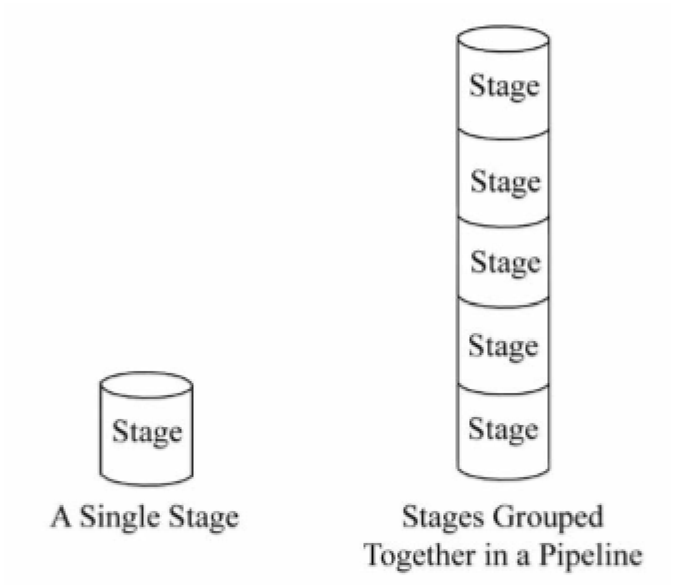
XSLT:
XSLT describes how to transform a source document from one markup language to another.

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

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

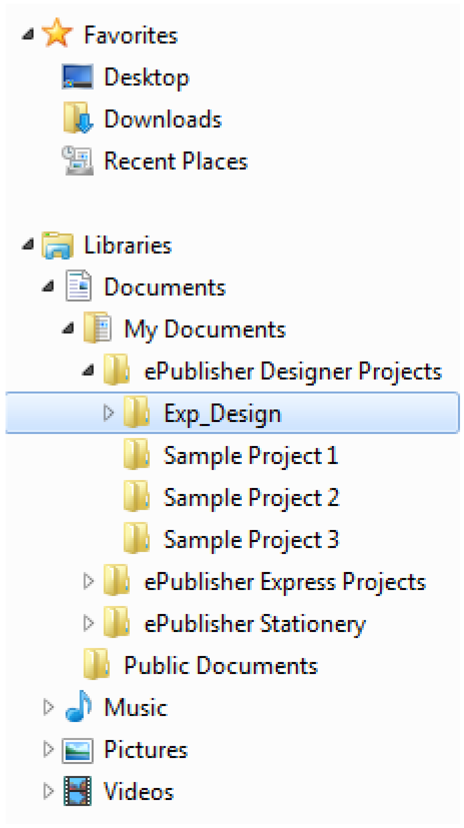
There are two locations to keep in mind when planning to modify a file:

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 Format 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 Format Settings and Style Designer are consulted, stored in the project file (**.wep**) and incorporated in the final generation of output.

Components

```
<Classes>
<Groups>
<Settings>
<RuleTraitsSet>
```

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 Format Settings button. A user may customize the format 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

```
<Pipelines>
<Capabilities>
```

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

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, Format 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 Format Settings and Style Designer.

Project Format 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 Target Override directory. By placing a file of the same name at the correct location in the Project Target Override directory, any file in a given format within the installation directory may be overridden.

Creating Project Format Overrides

Information about Overriding files

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

Procedure

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.

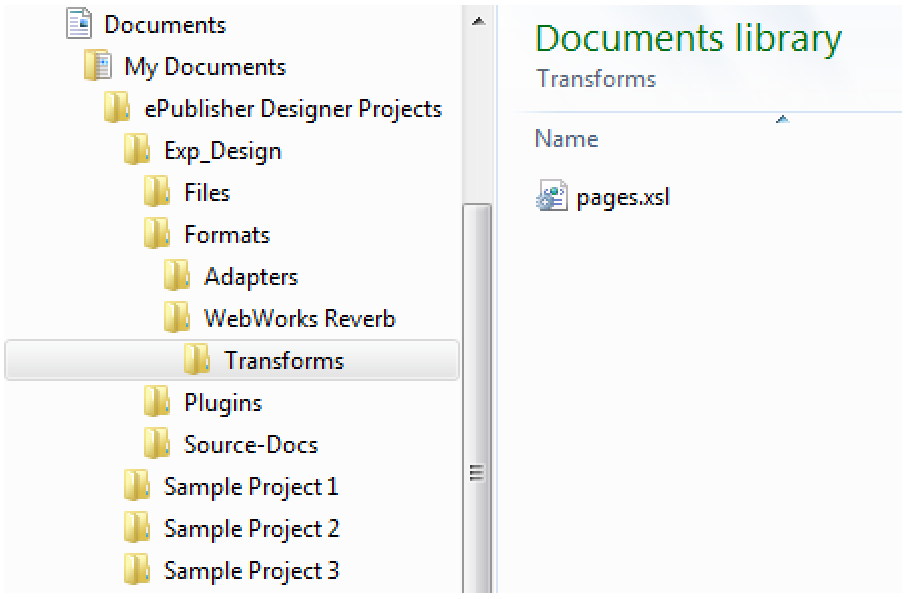


Figure 2: Folder Hierarchy Required To Modify pages.xsl in WebWorks Reverb

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:

- XSLT - Basics and applying to ePublisher
- XSLT, Part II - Beyond the Basics, Applying to ePublisher
- XSLT, Part III - Extending ePublisher Styles and Settings

From the WebWorks Wiki:

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

XSLT Quick Reference Guides

- XML Syntax
- XSLT 1.0 and XPATH 1.0
- XQuery 1.0
- XQuery/XPath Functions and Operators

XSLT Quick External Reference

- XSL Transformations (XSLT)
- XSLT Elements Reference
- XSLT, XPath, and XQuery Functions
- XSLT Examples
- 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

Difference between HTML and XHTML

Useful tool for generating API documentation: Doxygen

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>
```