



XMetal[®] 19 Developer

Customization Guide

2024 JustSystems Canada Inc.



About JustSystems

JustSystems is a leading global software provider with three decades of successful innovation in office productivity, information management, and consumer and enterprise software. With over 2,500 customers worldwide and annual revenues over \$110M, the company is continuing a global expansion strategy that includes its enterprise software offering called xfy, its XMetaL content lifecycle solutions, and its pioneering work in the definition of the XBRL standard and commercialization of enabling technologies. A Gartner "Cool Vendor" selection in 2008, JustSystems is also a member of KMWorld's 100 Companies that Matter in Knowledge Management for 2008 and the 2007 EContent 100. XMetaL is a 2008 KMWorld Trend-Setting Product. Major strategic partnerships include IBM, Oracle and EMC. For more information, please visit <http://www.justsystems.com>.

Copyright JustSystems Canada, Inc. All rights reserved. XMetaL is a registered trademark of JustSystems Canada, Inc. Other product names may be trademarks or registered trademarks of their respective owners.

Contact Information:

Support:

North America: +1 866 647 2003

Sales:

North America: +1 866 793 1542

Office Locations:

XMetaL Sales & Support

Suite 3220

666 Burrard Street

Box 207

Vancouver, BC, Canada

V6C 2X8

T: 604-602-9928

Toll-Free Sales: 1-866-793-1542

Tokushima Head Office

Brains Park Kawauchi-cho

Tokushima-city Tokushima 771-0189

Japan

T: 088 666 1000

(+81 88 666 1000 from outside Japan)

Contents

Introduction.....	6
Creating customizations.....	7
Before you begin.....	7
Components.....	8
File properties.....	9
Visual Studio .NET Solution Explorer.....	10
Element properties.....	11
General.....	11
Change list.....	12
Display As.....	12
Followed by.....	13
On Insert.....	14
Text layout.....	15
Treat As.....	15
Global properties.....	16
Virtual element.....	17
In-Parent element.....	17
Building a customization.....	18
Configuring the build environment.....	18
Debugging a customization.....	19
Debugging and testing an XMAX customization.....	20
Customizing using C++.....	22
Explicit application-level customization support (without using XAC).....	22
External event handling in XMetaL.....	26
Scripts.....	31
Script editor.....	32
Creating scripts.....	32
Create a script.....	33
Importing scripts.....	33
Import a script.....	33
Import scripts from an MCR file.....	34
Testing scripts.....	34
Importing data.....	34
Converting Microsoft Word documents.....	35

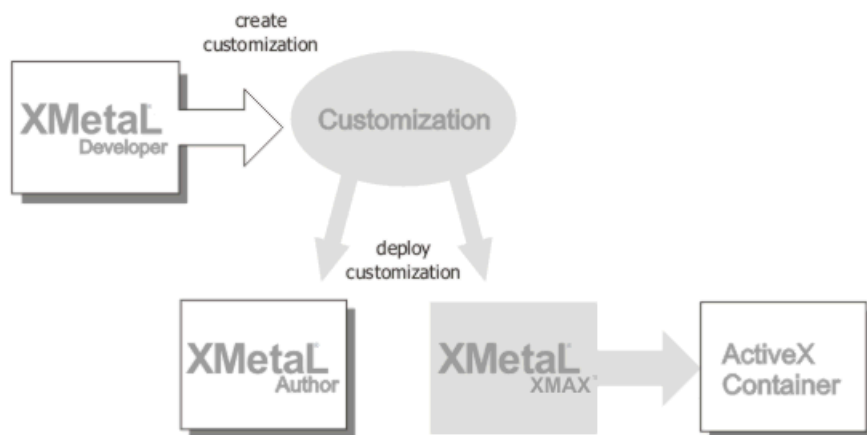
PDF and HTML previewing and printing.....	35
DTDs and schemas.....	37
Creating a DTD.....	37
Modifying your DTD.....	38
Rules files.....	38
Document type declarations.....	39
Internal subset.....	40
Mapping identifiers.....	41
Catalog files.....	41
Resolving catalog file entries.....	42
Catalog support for schemas.....	43
Finding catalog files.....	44
Giving priority to system or public identifiers.....	44
External identifier map file.....	45
Creating an external identifier map file.....	46
Language support.....	47
SGML declaration.....	47
Attribute description files.....	49
Forms.....	50
XMetaL Forms Toolkit.....	51
Create a form.....	53
Binding a form to XML content.....	54
External data.....	56
Connect to an external data source.....	57
Associate a form with a customization object.....	58
Executing a form as a modal dialog in XMetaL.....	58
Sample forms.....	59
Editor display styles.....	60
CSS editor.....	60
Creating selectors.....	61
XMetaL-specific selectors.....	63
Create a selector.....	63
Custom selectors.....	64
Setting style properties.....	64
Extensions properties.....	65
View support for properties and selectors.....	65
Using counters and autonumbering.....	69

Formatting elements as tables.....	71
Example style rules.....	72
Resource Manager.....	73
Configuring the Asset Manager.....	73
Creating asset types.....	75
Asset display file.....	75
Asset catalog file.....	76
Text file and text block assets.....	77
Master asset catalog file.....	77
Remote assets.....	78
Set up a remote assets folder.....	78
Configuring XMetaL.....	80
Adding new toolbar icons.....	80
Frequently used configuration variables.....	81
Configuration variables.....	81
Glossary.....	90
Index.....	96

Introduction

With XMetaL Developer, you can create custom user environments for XMetaL Author and XMetaL XMAX. Customizations are created in the Microsoft® Visual Studio .NET development environment.

Creating a customization is a multi-step process. However, the first step requires no development tools at all. Before beginning any customization you must have a clear definition of the project, including its purpose, scope, audience, and workflow. Only then is it possible to begin a customization that satisfies your project plan.



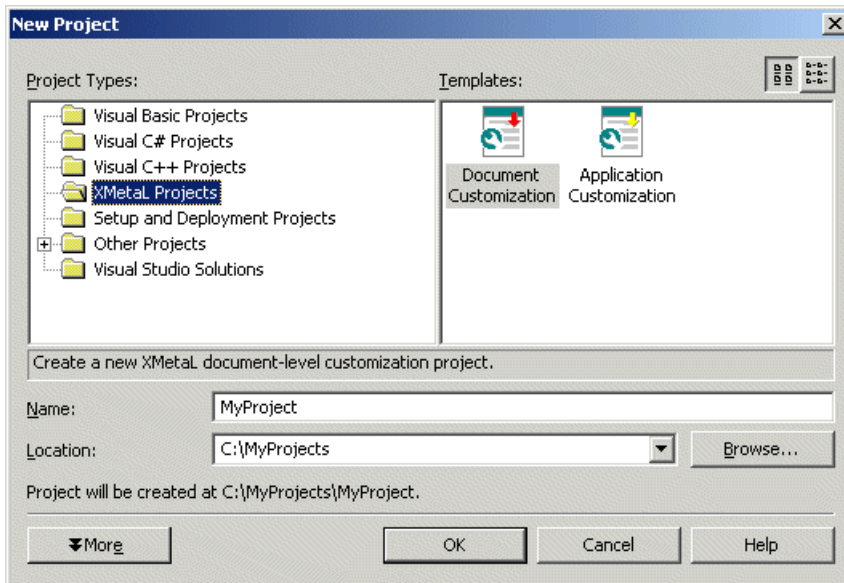
Feedback

Send comments or questions about XMetaL documentation to docs-feedback@xmetal.com.

Creating customizations

You create and manage the components of your customization in a Visual Studio .NET solution.

You can create a customization solution through the **File > New > Project** menu. You then choose a solution template in the **XMetaL Projects** folder. Follow the wizard to finish creating your customization.



Before you begin

You need to have a project plan and knowledge of the development environment, XMetaL Author, and related technologies.

Before you start creating a customization you should have a project plan. The objective of the customization is to achieve the goals set out in your plan. It should reflect the following:

- The tasks you want to perform with your customization
- The order of these tasks
- Dependencies on these tasks

By understanding your project as a series of tasks, you can better design your DTD and determine the scope of the customization, that is, whether it is an application-level or document-level customization.

In order to work effectively, you also need to have knowledge of the following:

- XML
- DTD or schema
- CSS
- XMetaL Author or XMetaL XMAX
- Visual Studio .NET

Components

A customization solution contains file components that determine the appearance and behavior of elements and tools in the XMetaL Author or XMetaL XMAX environment. These components are compiled into an XMetaL Application Customization (XAC) file.

Document Type Definitions or schemas

The Document Type Definition (DTD) or schema (XSD) forms the basis of your customization. After you have planned the workflow, you should create a DTD or schema if you do not already have one. Your DTD or schema should support your workflow and the types of information you need to create and manage.

Scripts

Scripts provide access to XMetaL Author or XMetaL XMAX via an object model that is based on the Document Object Model (DOM) and the Microsoft Word VBA model. Scripts are stored as components (for example, JScript files) in your customization. These components are visible through the Solution Explorer. You can create a new script or import an existing script. Scripts are deployed with your customization.

Settings files

Each installation of XMetaL Author contains a global configuration file and a user configuration file. You can configure XMetaL Author behavior for your customization through the global configuration file. Configuration files contain one or more variables that are read when you start XMetaL. Each variable is a name-value pair. The values can be booleans, pathnames, numbers, and strings.

Customization files

Customization (CTM) files contain the customizations carried out by the XMetaL Customization editor. The customizations for a DTD called *dtdname.dtd* are saved in *dtdname.ctm*. The CTM file must reside in the same folder as the DTD (or Schema or rules file).

Forms

Forms help end-users to create structured content. You can use the XMetaL Forms Toolkit (XFT) to create forms that can be run from within XMetaL either as modal dialog boxes or embedded within a document. In addition, you can create forms that are bound to XML content and apply your organization's business logic via scripting.

Document templates

Document templates provide outlines and let you create a new document that uses a specific DTD or rules file. If you want default content to be entered when an element is selected, you can insert replacement text in the template. This is similar to a user prompt. A template is used when a new file is created with the **File** > **New** command in XMetaL Author. You should create at least one template for each DTD or rules file.

For more information about creating document templates, see the *XMetaL Author User's Guide*.

Toolbars and menus

Toolbars and menus are created when a new customization project is started. This information is stored in a toolbar (TBR) file. You can create new toolbars and menus through XMetaL Author while debugging your customization.

The following components of XMetaL Author are configured based on settings in the toolbar (TBR) file:

- Toolbars (containers for toolbar buttons)
- Toolbar buttons
- Menu (containers for menu items)
- Menu items

XMetaL Author creates toolbars and menus if it opens a customization for which no TBR file is provided. Modification of toolbars should be handled in XMetaL Author while debugging your customization. To modify a toolbar or menu, you first create a macro script and save it in the Macros folder for your customization. You then launch a debugging session and use XMetaL Author to create a new toolbar item (for example, a button) or menu item.



Note: If you stop the debugging session from XMetaL Developer, your changes to the toolbars are *not* saved.

For more information about creating toolbars and menus, see the *XMetaL Author User's Guide*.

Style sheets

When you begin a customization solution that contains no existing style sheets or customization files, XMetaL Developer parses the DTD or schema and generates the following files:

- CSS for the editor view
- CSS for the structure view
- CTM (customization) file

Styling is based on element names. If the names of your elements differ significantly from commonly used names, or if they are in a language other than English, XMetaL Developer cannot recognize the elements and cannot assign styles to them. XMetaL Developer recognizes many elements defined in XHTML, DocBook and the Journalist DTD.

You may need to modify these generated files to obtain the desired appearance or behavior.

File properties

You can specify properties for each component in your customization. Each component has a UseAs property.

UseAs

This property is necessary to communicate to XMetaL Author and XMetaL XMAX the intended purpose of the item. For example, if you have a file, `myitem.xft` that is to be used as an XFT form, you must set the UseAs property to be An XFT Form.

Table 1: UseAs values

UseAs value	Internal name used by XMetaL API	Comment
CSS styles file for Structure view	XMETAL_CSS_STRUCT	CSS used to add additional formatting to the styles used for the Structure View.
CSS styles file for Normal view	XMETAL_CSS_NORMAL	Not supported
Default CSS styles file	XMETAL_CSS_DEFAULT	Overrides the CSS file assigned the UseAs type of CSS styles file for Normal view, even if one is defined.
XMetaL Macro Script file	XMETAL_MACROS	MCR file
Customization file	XMETAL_CUSTOMIZATION	CTM file
An XFT Form	XMETAL_FORM	XFT file
Toolbar File	XMETAL_TOOLBAR	TBR file
XML DTD	XML_DTD	DTD file
Compiled Rules File from an XML DTD	XMETAL_XML_RULES	RLX file
SGML DTD	SGML_DTD	DTD file
Compiled XMetaL Rules File from SGML DTD	XMETAL_SGML_RULES	RLS file
W3C XML Schema document	XML_SCHEMA	XSD file
Compiled rules file from schema	XMETAL_SCHEMA_RULES	RLD

Visual Studio .NET Solution Explorer

You can view and manage all items in your customization within the Visual Studio .NET Solution Explorer. You are advised to use the XMetaL integrated development environment when working with customizations in order to take full advantage of XMetaL Developer.

You can **open** a customization through the **File** menu. After you select a solution, open the Solution Explorer by clicking **View > Solution Explorer**.

You can **add** items to your customization through the **Project** menu. You can add items as you need to work on them or all at once. The number and type of items you can add depends on the scope of your customization.

- For document-level customizations, you can add scripting objects, text files, forms, style sheets, and customization files.
- For application-level customizations, you can add scripting objects, text files, and forms.

You can add almost any other type of file to your project; however, some items may not be deployed properly or at all. In the case of DTDs and schemas, it is recommended that you deploy the compiled rules file (RLX or RLD) instead. This is because external entities are not supported in deployment.

You can **modify** an item in your customization by selecting it in the Solution Explorer and editing it in the main view.

You can **remove** or **delete** any items from a customization, except for:

- Macros folder
- References folder or any of its contents
- Rules files

You can remove items from one project, add them to another project, or add them back into the existing project at a later time. You can also completely remove items from the hard disk by deleting them. You can remove or delete items through the **Edit** menu.

You can **save** changes to your customization through the **File** menu. Customization files must be saved with one of the following encodings:

- ASCII
- ANSI
- UTF-8
- UTF-16

In most cases, ASCII will be acceptable; however, the coding you select should be the same as that of your DTD or Schema.

Element properties

Elements in your customization file (CTM) control the behavior and some aspects of the appearance of the XMetaL Author or XMetaL XMAX user environment.

You can set properties for elements in your customization file in the Properties pane. The properties of each element are displayed in rows in the main view. When you click on a row, the element properties appear in the Properties pane. You can modify the behavior of XMetaL authoring environments by changing properties.

You can set properties on a per-element basis or on a global basis through the #GLOBAL element. You also have the option of specifying an In-Parent context for an element or a Virtual element.

General

General properties determine basic element characteristics including an alias, if you want to call the element by a different name, and a description.

Alias

You can enter a real-language name for the selected element in the Alias field. This is useful when the element name is not very descriptive, or in cases where authors speak a language other than the language of the DTD. For example, if the element that you want authors to use for main titles is '<DT>', you can use the Name field to give the element the real-language name 'Document Title'—or its equivalent in the language of the authors.

If you do not enter a real-language name for the element, the element name from the DTD is used. This name is displayed in tags and in the Element List.

Description

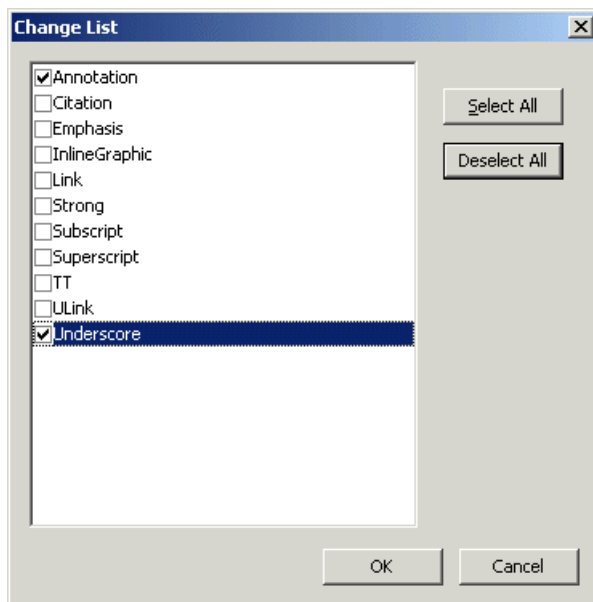
You can type a description of the element in the Description text box. If you do not enter a description, the element name from the DTD is used instead. This text is displayed at the bottom of the Element List.

Change list

You can specify which elements are available in the change list in XMetaL Author through the Change list property.

In XMetaL Author, the change list is displayed at the left end of the Formatting toolbar. The list contains the names of elements that the current element can be changed to. The Change List dialog displays the list of valid elements.

Since all configurations in this dialog box are relative to a parent element, you may have to re-configure several parents in order to cover all contexts in which the element can occur. For example, if you have two list elements, for example, *Bulleted* and *Numbered*, that can occur inside *Book*, *Chapter*, *Section*, and *Para*, you must include *Bulleted* and *Numbered* in the change list for each parent.



Display As

The Display As properties let you associate forms or controls with an element.

You can specify a form or control by clicking the browse button in the **Advanced Display Type** property.

XFT Form

You can customize your documents and templates with forms (or dialogs). These are associated with specific elements to facilitate data entry and the XML authoring process.

You can use the XFT Form Wizard to set up forms that are bound to XML content by an XPath expression. By using the wizard you can specify the following:

- How to run the form (Embedded or Modal)
- How to display the form (Replace, Before, After)
- When to display the form (Always, Dynamic Script)

If you select Dynamic Script, you must create a script for XMetaL to call before it displays the form. Use this script to create business rules that determine whether or not to display the form at run-time.

For example, the following script shows how a dynamic display script can be used to conditionally display a form, depending on the parent of the form node.

```
// Put this script in the Script Text box of the XFT Form Wizard
// Get the form control
// Its node member is the DOMNode of the element to be tested
var aipc = Application.ActiveInPlaceControl;

// If the node's parent is "Publisher" ...
if (aipc.node.parentNode.nodeName = "Publisher") {
// ... then display the form
aipc.ShouldCreate = true;
}
else {
// ... otherwise do not display the form
aipc.ShouldCreate = false;
}
```

In-Place Control

You can use in-place ActiveX controls to represent elements in Normal and Tags On views and determine their behavior. When you specify that an element is to be displayed as an in-place control, you must specify a script prefix. This prefix is used in your macros to enable the control to communicate with XMetaL. You also have the option of specifying Bitmap Printing. This option has been provided because some controls (for example, the Internet Explorer WebBrowser Control) do not print using the standard ActiveX mechanism. If one of your controls does not print using the default mechanism, you should enable this option.

Refer to the documentation for the ActiveX control to learn about possible limitations and restrictions associated with the control.

Associate an element with an in-place control

You can use in-place ActiveX controls to represent elements in Normal and Tags On views.

1. In the Advanced Display Type property, select the **In-place Control** option.
2. In the **ProgID/ClsID** text box, type the identifier for the control you want to use.



Note: If the control you want to use does not appear in the list, you can find its ClsID or ProgID by consulting the control documentation for the control or by searching the registry (advanced users only).

3. Type a script prefix.
4. Set the **Use Bitmap Printing** option.
5. Click **OK**.
6. Create the macros that enable the control to communicate with XMetaL.

Followed by

You can configure XMetaL to create an element of your choice when Enter is pressed at the end of a particular element.

By default, if you press Enter inside a specific block element, XMetaL Author creates a new element of the same type, if the DTD or schema allows it. You can change the default behavior if it is not permitted by the DTD or schema, or if you want a different element to be created. The original element must be defined as a block element in the CSS style sheet for the document.

On Insert

The On Insert properties let you specify default content or a script to be run whenever you insert a particular element.

You can choose the following from the **On Insert Type** dropdown:

- **None.** No content or script is to be inserted.
- **XML content.** Lets you specify a mini template.
- **Script.** Lets you specify script and script language.

MiniTemplate

The MiniTemplate property contains the default content for the selected element. You can use this property to indicate the markup that is to be inserted for this element. The default content can contain *replacement text*, which appears in an element by default when the element is inserted. Replacement text is contained in 'xm-replace_text' processing instructions.

If no content is specified, XMetaL inserts replacement text based on the element name. For example, for an element named 'Para', the replacement text is '{Para}'. XMetaL does not insert replacement text if none is specified in the Content section and Paragraph is selected in the Treat As section.

The default content must start with the specified element, not a sub-element. This allows you to specify default attribute values for the element.

Script

You can specify a script to be loaded and run whenever a particular element is inserted. Most scripts are written in JScript or VBScript, but you can use any language for which you have a scripting engine that conforms to the Microsoft Scripting Engine Interface. The name of the language must correspond to the **ProgId** (for example, PerlScript or Python) of the ActiveX control that implements the script engine.

Inserting a mini-template

For example:

```
<Title><?xm-replace_text {Book Title} ?><Title>
```

When the <Title> is inserted, it contains the text '{Book Title}'. Authors can select this text and type over it with real content.

Inserting a script

The following example illustrates the specifying of default content using a template and using a script.

Consider the following template:

```
<Warning><Para>  
<?xm-replace_text {Warning}?>  
</Para></Warning>
```

The same results can be obtained with this VBScript code:

```
Selection.InsertElement "Warning"  
Selection.InsertElement "Para"  
Selection.InsertReplaceableText "{Warning}"
```



Note: The Selection.InsertWithTemplate method can be used in a script to insert elements and their default content.

Related Links

[Microsoft Scripting Engine Interface](#)

Text layout

You can set text layout properties to determine how an element appears in Plain Text view. These properties can be set on a per-element basis or globally.

XMetaL makes XML source files (as displayed in Plain Text view) easier to view by inserting indentation and line breaks and distinguishing between content and markup. This formatting is called *text layout* or *pretty printing*. You can specify this setting through the Disable Text Layout property in the Global properties.

Text layout properties are also available on a per-element basis. For example, you can

- Preserve space or indent content
- Add blank lines before or after tags
- Change tag color

Layout options are applied to files saved from Tags On and Normal views. These options are also applied when you switch from Tags On or Normal view to Plain Text view.

Preserve Space

When you turn on Preserve Space for a parent element, any Element Options changes you specify for child elements are ineffective. If you want to use the Preserve Space option for a parent element, be sure that all child elements have the appropriate Element Options selected.

Treat As

You can configure XMetaL to treat selected elements as paragraphs, toggling elements, image elements, or list elements.




You can also rank the elements of each type. Whenever XMetaL can insert a paragraph, image, or list element, it inserts the **first-ranked** valid element of the required type. If the first element in the given ranking is not valid, XMetaL tries to insert the second-ranked element, and so on.

Paragraphs

Designating an element as a paragraph element determines how the element is processed:

- If you attempt to enter text where text is not allowed but where a new paragraph is allowed, XMetaL uses one of the elements designated as paragraph elements to create a new paragraph element and places the text inside the new element.
- If you press Enter at the end of any element and there is no followed-by element or required element and the element cannot be split, XMetaL inserts one of the elements designated as a paragraph element.
- When adding items to the list of elements in the Paragraph Order, only items that have been set as paragraphs can be added.

Toggleing elements

You can assign inline elements to the **Bold** , **Underscore** , and **Italic**  buttons. When an element is assigned to one of these buttons, clicking the button inserts an empty inline element at the cursor location or surrounds selected text with the inline element markup (if the assigned element is valid in that location). You can also click the button to remove markup for the assigned element from the selected text.

You can also assign an element to a macro. In this case, the macro should be assigned to a toolbar button.

Images

You can designate an element as an image element. The following attributes are available on image elements:

- Source (a filename, a URL, or an entity name used to map to the image)
- Height
- Width
- Alt Text
- Valign
- Scale

Lists

You can use special list editing features to insert and edit list elements. The following list types are available:

- Numbered lists
- Bulleted lists
- Definition lists

Numbered and bulleted lists consist of a list element containing one or more list item elements. Some lists may have a list header element.

Definition lists are two-part lists that consist of a list element and two kinds of sub-elements: term elements and definition elements.

Global properties

You can specify some properties for all elements on a global basis. Global properties are available through the #GLOBAL element.

Disable text layout

This property is set to Yes by default, meaning that all text layout properties for every element are disabled. In order to set any of the Text Layout properties for any element, this property must be set to No.

Paragraph order

When authors press **Enter** at the end of a paragraph element, XMetaL inserts a new paragraph element. The default behavior is to insert the same paragraph element type after the current paragraph. You can use the Paragraph order to set up a list of elements (which you have previously set as Paragraph types using the TreatAs property) that XMetaL tries to insert. If XMetaL cannot find a valid element to insert from the list, no element is inserted.

Namespaces

You can set unique customization properties to similarly-named elements from different schemas. Namespaces are displayed in the Customization Editor as prefixes to the element name in the form:

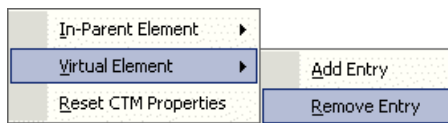
```
{namespace} : {elementname}
```

You can edit namespaces by right-clicking on the #GLOBAL element and selecting **Edit Namespace Prefix-URI Map**.

Virtual element

Elements that you specify as Virtual elements are added via script in the macro editor with the AddElement() and AddElementToInclusion() methods.

To add or remove a virtual element, right-click an element and select **Virtual Element > Add Entry** or **Virtual Element > Remove Entry**.



In-Parent element

You can define a context for an element by specifying an In-Parent element. If no In-Parent element is specified, the properties apply to the element in all contexts.

When you define an In-Parent element you are informing XMetaL Author that the properties assigned to this element apply only in a specific context. To assign an In-Parent element, right-click an element and select **In-Parent Element**.



Assigning an In-Parent element

For example, to assign properties to all Title elements in the following document, you do not need to specify an In-Parent element. However, to assign properties only to Title elements that appear within Chapter elements, you need to specify Chapter as the In-Parent element.

```
<Book>
<Title>My Book</Title>
<Chapter>
<Title>Welcome to Chapter 1</Title>
<Para>This is a paragraph...</Para>
</Chapter>
<Bibliography>
<Title>This is my bibliography</Title>
<Entry>This is an entry...</Entry>
<Entry>This is an entry...</Entry>
</Chapter>
</Book>
```

Building a customization

When you build a customization, you compile all of your customization files into a binary distribution file called an XMetaL Application Customization (XAC) file. The XAC file is deployed to the XMetaL Author or XMetaL XMAX authoring platforms.

Build options are available through the **Build** menu. You can check the build status, including warnings and errors, in the output pane.

After you have built the project, check the task list to view information about the build that may require your attention before the customization can be deployed.

You can see the results of the build by browsing to the Output folder. The files in this folder are either copied project files (for example, CSS and INI files) or compiled files (for example, MCR files). DTDs and schemas are not directly included in XAC files. Instead, they are built into rules files, which are included in the XAC file.

Related Links

[Configuring the build environment](#) on page 18

Before you build your customization for the first time, you may have to configure the build environment. You can do this by setting project properties.

[Deploying customizations](#) You can deploy XMetaL customizations by distributing individual files or a single XMetaL Application Customization (XAC) file.

Configuring the build environment

Before you build your customization for the first time, you may have to configure the build environment. You can do this by setting project properties.

In most cases, you do not have to set all project properties. However, the general and debugging properties may have to be set for every customization. You can view these properties by selecting the customization project folder in the Solution Explorer and clicking **View > Property Pages**. These properties include general, debugging, and post-build properties.

Table 2: General properties

Name	Description
Output directory	The destination folder for the build files. If no folder is specified, the subfolder <code>\BuildProject</code> is used.
Output File Name	The name of the generated XAC file.

Table 3: Debugging properties

Name	Description
Command	The application to launch when testing your customization. If you have indicated XMetaL Author in the configuration list, the default setting for this is the latest version of XMetaL Author that you have installed on your system. If you have indicated XMetaL XMAX in the configuration list, there is no default setting. In addition, you must indicate the container application that instantiates the XMetaL XMAX

Name	Description
	<p>control, or, in the case of an HTML page that contains the ActiveX control, you must specify Internet Explorer (usually in the {drive}\Program Files\Internet Explorer folder, where {drive} is the drive letter of the installation path).</p> <p>XMetaL Developer is used by default XMAX_x64_test.exe/XMAX_x86_test.exe applications for Debugging and testing an XMAX customization on page 20.</p>
Command line	<p>The command-line arguments to be added. If you have indicated XMetaL Author in the configuration list, this property is typically an XML document based on your customization. When opening a document in XMetaL Author, enclose the document path and name in quotation marks (""). If you have indicated XMetaL XMAX in the configuration list and your customization has XMetaL XMAX embedded in an HTML page, this property is the HTML page that contains the embedded XMetaL XMAX object. If you have a container application that instantiates XMetaL XMAX, the arguments have no effect unless your container application accepts command line arguments.</p> <p>XMetaL Developer is used by default XMAX_x64_test.exe/XMAX_x86_test.exe applications command line arguments for XMAX debugging on page 21.</p>
Working folder	<p>The folder that the application specified in the Command property opens in. It is the default folder for the launched application.</p>

Table 4: Post-build properties

Name	Description
Command line	The application or batch file plus the command-line arguments added to the call to launch the program.
Description	The text that Visual Studio .NET displays when the post-build application runs.
Exclude from build	Toggles the post-build application launch on or off.

Debugging a customization

If you are testing your customization using XMetaL Author as your test application, you can quickly move between XMetaL Author and XMetaL Developer to test and change scripts, style sheets, and customization files.

When XMetaL Author is launched from inside XMetaL Developer, the following items are added to the **Edit** menu:

- **Edit CSS file in XMD.** By clicking this menu item, the currently-selected element in XMetaL Author is passed back to the XMetaL Developer environment and the CSS Editor is opened with the element selected.

Changes made to any of the display settings are automatically updated once you save the CSS. By returning to the debugging application, you can immediately see your changes in the document view.

- **Edit CTM file in XMD.** Clicking this menu item returns control to the XMetaL Developer environment and opens the CTM Editor. From there you can modify this file and change the behavior of the macros it contains.



Note: Sometimes, complex selectors cannot be resolved, in which case XMetaL Developer asks you to locate the selector yourself from the list provided in the CSS editor.

If you are testing your customization in XMetaL Author, and you indicate that you want to create a new document from a template, XMetaL Author uses the DTD or schema for the customization project to create the new document.

If you are testing your customization in XMetaL Author, and a script stops at a line that contains a run-time error, you cannot edit the script and continue execution. To continue executing a script after a run-time error, click **Break**, then set the next line to be the next statement to be executed by right-clicking the next line and selecting then select **Set next statement** from the context menu. You can edit the script only after it has finished executing. To edit the script after execution is complete, use the Visual Studio .NET script editor.



Note: XMetaL Developer cannot extract SGML declarations from documents in debug mode. Additionally, documents with internal element declarations (internal subsets) produce errors during the build process.

Related Links

[Scripts](#) on page 31

Scripts provide access to XMetaL Author or XMetaL XMAX via an object model that is based on the Document Object Model (DOM) and the Microsoft Word VBA model.

Debugging and testing an XMAX customization

The XMetaL Developer installation contains container applications for debugging and testing XMetaL XMAX customization.

Application locations:

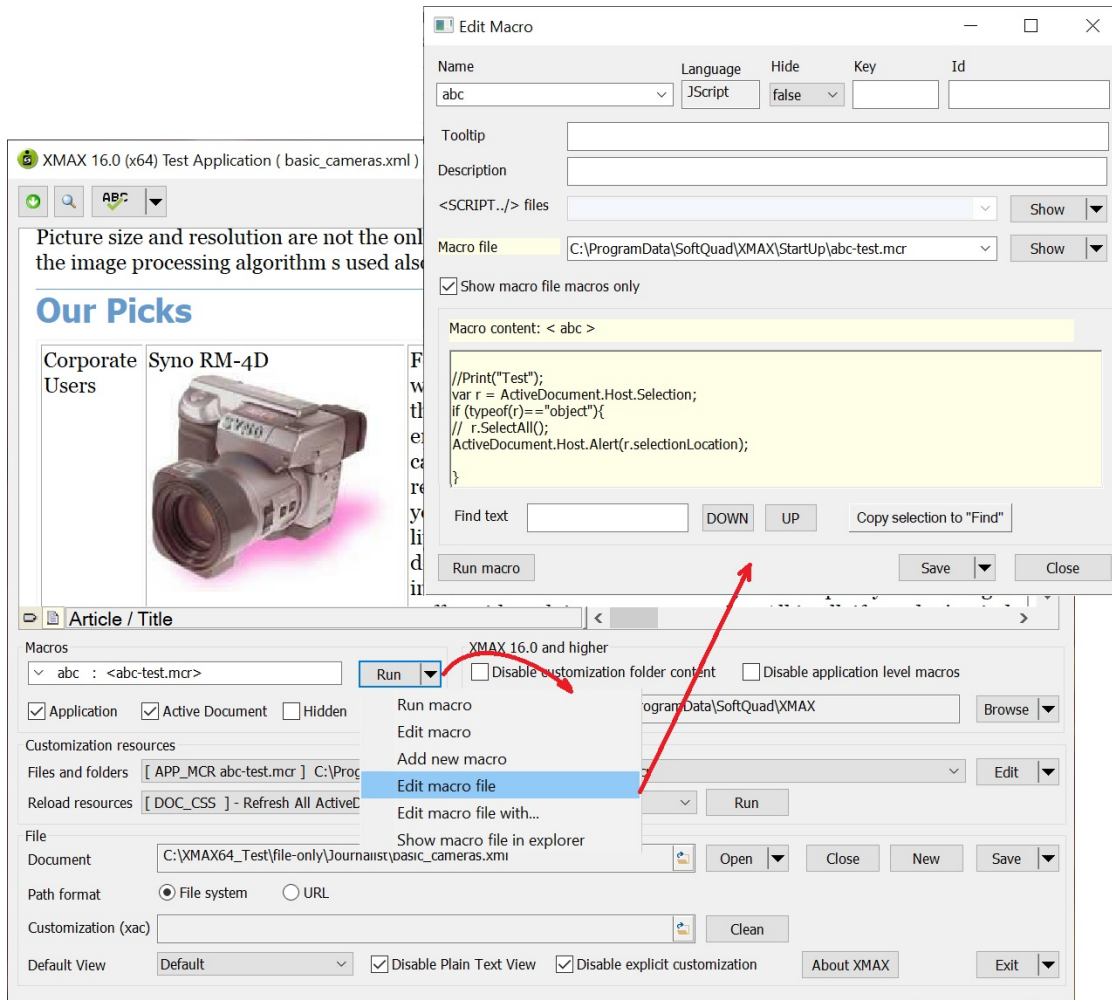
- <installation folder>\Developer\Utilities\XMAX_x64_test.exe
- <installation folder>\Developer\Utilities\XMAX_x86_test.exe

Applications support base operation:

- New, Open, Close, Save and Save As document
- Find in document
- Spell check
- Enable/disable document XAC customization
- Enable/disable plain text view
- Find and run customization macros

For testing a customization outside XMetaL Developer, the following operations are supported in addition to base operations:

- Add, view and modify macros
- View and open customization resources (*.mcr, *.css, *.ctm, etc.) associated with customization
- Reload resources, document or application
- Open new instance of application and documents



Command line parameters:

- Load the document in the application XMAX_x64_test.exe "<document full path>"
- Load the document with the customization file for testing without using a Visual Studio debugging environment: XMAX_x64_test.exe -f="<document full path>" -x="XAC file full path"

Initial UI related command line parameters:

- "Plain Text" is default view: -v=S
- "Tags On" is default view: -v=T
- "Normal" is default view: -v=N
- Enable source view by default (can be turned on/off later in the application UI): -ESV
- Disable XAC customization specified in -x="XAC file full path" (can be turned on/off later in application UI). Use this parameter for debugging in Visual Studio: -DXAC
- Hide settings and resources specific information (can be turned on/off later in the application UI) : -m

Command line examples:

- XMAX_x64_test.exe "c:\Temp\My documents\example.xml"
- XMAX_x64_test.exe -m -f="c:\Temp\My documents\example.xml" -x=" c:\Temp\My cusomizations\dtd-subjects.xac"

- `XMAX_x64_test.exe -m -ESV -DXAC -v=T -f="c:\Temp\My documents\example.xml" -x="c:\Temp\My customizations\author.xac"`

Customizing using C++

You can extend the functionality of XMetaL by creating an ActiveX control (such as a DLL) in C++. For example, you can use C++ to create a custom dialog.

The following example uses the Journalist DTD, which is based on DocBook. Here, you create a DLL in Microsoft Visual C++ and test the output using XMetaL.

Example

You create a new ATL project called Journalist that contains the following:

- An ATL class called `CCitation`
- A method called `NewCitation`
- XMetaL interfaces imported from `..\Program Files\XMetaL\Author\xmetal.tlb`

You can then test to see if you can call the new object from XMetaL. First, add the following line to the body of the `NewCitation` method:

```
::AfxMessageBox("Inside NewCitation() Procedure!");
```

After you have built `Journalist.dll`, open `..\Program Files\XMetaL\Macros\journalist.mcr` using a text editor and add the following macro:

```
<MACRO name="InsCite" key="Ctrl+Alt+A" lang="JScript">
var obj=new ActiveXObject("Journalist.Citation");
obj.NewCitation();
</MACRO>
```

When you start XMetaL, create a new document using the Journalist DTD, and press `Ctrl+Alt+A`, XMetaL displays a dialog containing the text you added to the `NewCitation` method.

Explicit application-level customization support (without using XAC)

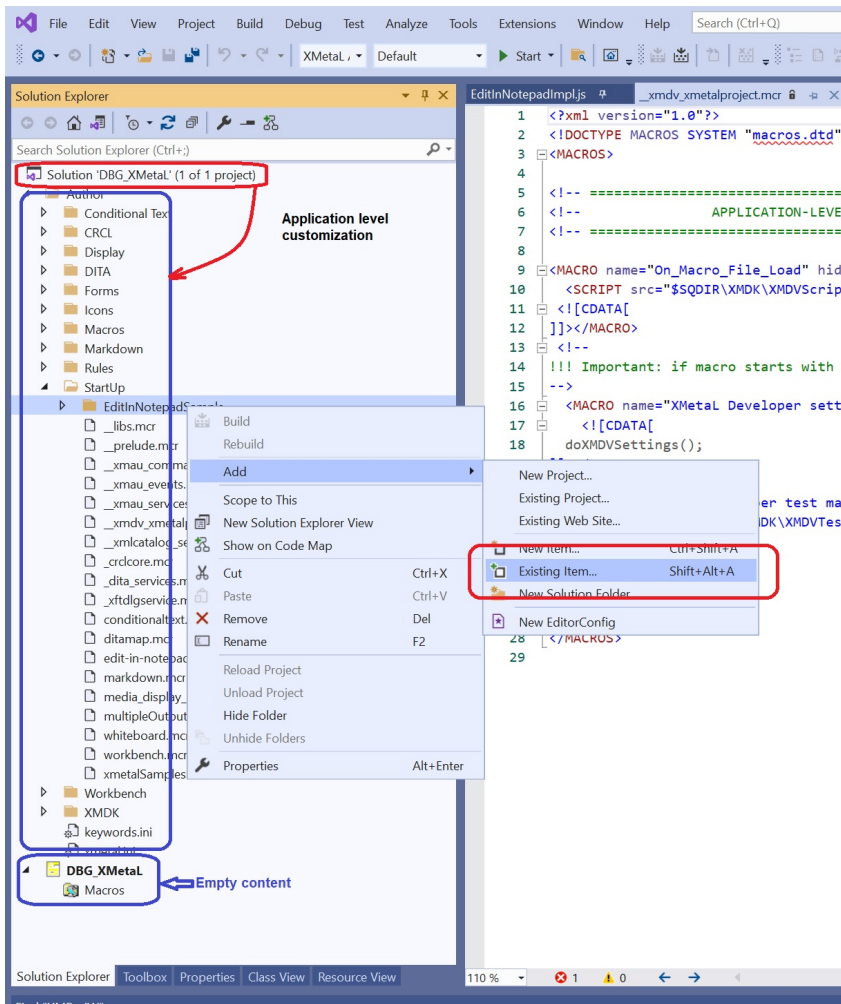
Integrating XMetaL Developer with Microsoft Visual Studio provides efficient tools for XMetaL application-level customization script debugging. Application-level customization macros apply to all types of documents opened in XMetaL. The `<XMetaL installation folder>\Author\Startup` folder contains a set of `*.mcr` files that defines customization scripts and meta information for integration with XMetaL. Complex customizations can contain hundreds utilities and tens of thousands line of code. XMetaL `*.mcr` files are written in xml format. Optional first children of "`<MACRO>`" tag are "`<SCRIPT/>`" elements.

`<SCRIPT/>` elements allow users to specify the list of script files that XMetaL embeds into the macro during XMetaL start up initialization. The XMetaL Developer solution provides an efficient way of maintaining, native editing, navigation and script debugging in Visual Studio. It includes breakpoints, variable watching, XMetaL API intellisense support, etc.

<SCRIPT/>" elements contain information about script files' location, language and title (usually it is the name of the file that appears in Visual Studio during debugging). The "src" attribute contains the relative path to a referenced script file. The '\$SQDIR' variable can be used at the beginning. XMetaL will resolve it as path to the ...\\Author folder.

```
<!DOCTYPE MACROS SYSTEM "macros.dtd">
<MACROS>
  <MACRO name="On Macro_File_Load" lang="JScript" hide="true">
    <SCRIPT src="$SQDIR\Test\ExtensionsData.js" lang="JScript" name="ExtensionsData.js"></SCRIPT>
    <SCRIPT src="$SQDIR\Test\WhiteBoardService.js" lang="JScript" name="WhiteBoardService.js"></SCRIPT>
    <SCRIPT src="$SQDIR\Test\WhiteBoardImpl.js" lang="JScript" name="WhiteBoardImpl.js"></SCRIPT>
    <SCRIPT src="$SQDIR\Test\WhiteBoardDebug.js" lang="JScript" name="WhiteBoardDebug.js"></SCRIPT>
    <![CDATA[
  ]]>
  </MACRO>
  <MACRO name="Add to favorites..." lang="JScript" hide="false" desc="ABC">
    <![CDATA[
// OpenAddToFavoritesDlg(...) defined in WhiteBoardImpl.js
OpenAddToFavoritesDlg(1);
  ]]>
  </MACRO>
  <MACRO name="Test macro" lang="JScript" hide="true">
    <SCRIPT src="$SQDIR\XMDK\WhiteBoard\js\test.js" lang="JScript" name="test.js"></SCRIPT>
    <![CDATA[
// doTest(...) defined in test.js
doTest(1);
  ]]>
  </MACRO>
</MACROS>
```

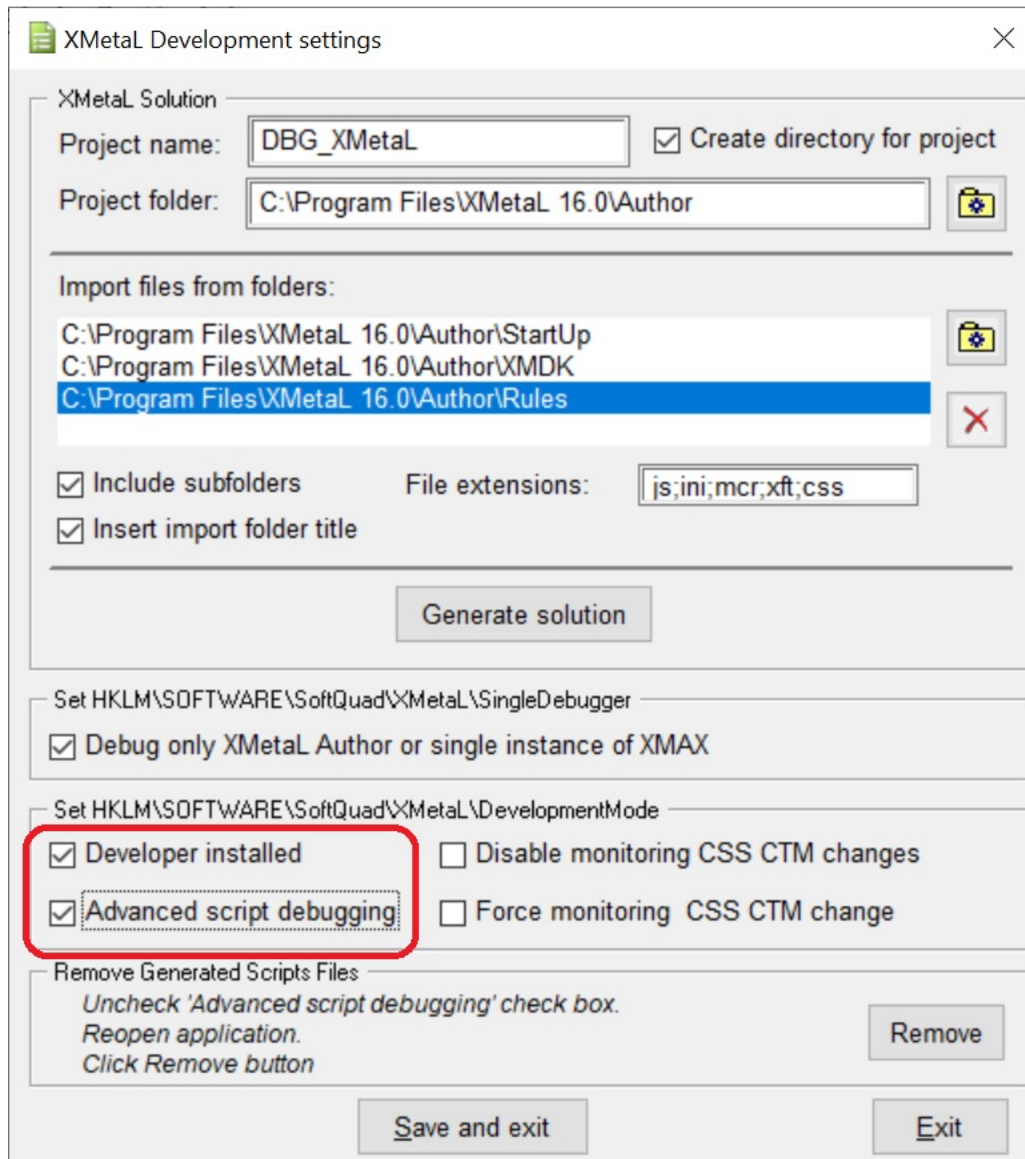
To use this capability in XMetaL Developer, a user should create an XMetaL application-level customization project. An XMetaL application-level customization based on <SCRIPT/> references in .mcr files does not use XAC so the initial customization content can be empty. All document files (scripts, macros (*.mcr), forms (*.xft), *.css styles or any other types) must be added in the Visual Studio "Solution Explorer" tab **outside of the XMetaL Project node** as an "Existing item...".



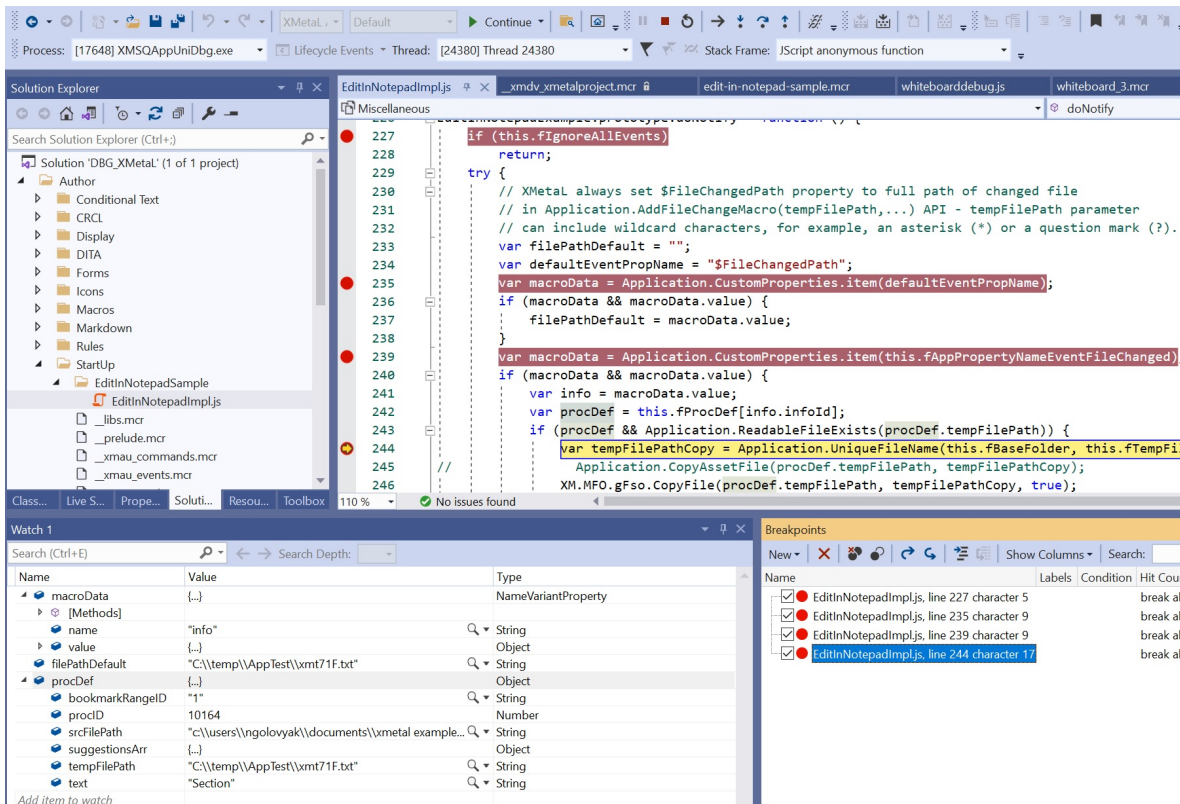
Customization documents are usually located under the <XMetaL installation folder>\Author folder. To modify them, check that the user has system permissions to modify files and folders. Running Visual Studio as an administrator is recommended.

The XMetaL Developer installation deploys to the XMetaL Author installation folder, application-level customization for modifying default XMetaL Developer settings.

1. Run xmetal.exe.
2. Run the **XMetaL Developer Settings** macro. To do this:
 - a. Click "Ctrl+?" to open quick navigation tools.
 - b. Start to type "developer" in the find edit box.
 - c. Select and run the macro.
 - d. Check that "Developer installed" and "Advanced script debugging" check boxes are "checked".



3. Close XMetaL application
4. Open the script documents in Visual Studio, set breakpoints and start debugging as usual for an XMetaL customization.



External event handling in XMetaL

While writing XMetaL script customization, some actions require using third-party external applications or running bat files. These actions can take significant time but it is possible to handle such operations asynchronously in XMetaL.

XMetaL Customization JScript APIs allow users to:

- Create processes
- Specify command lines
- Run processes asynchronously

The return value of such operations is "process ID".

XMetaL API: Application.RunAfterProcessDone(...) and **Application.RunAfterProcessDone2(...)** run XMetaL macros after process (identified by "process ID") termination and XMetaL is in idle state.

A VARIANT type optional parameter type can be passed to **Application.RunAfterProcessDone(...)**. The assigned XMetaL application-level custom property contains the parameter's value when XMetaL runs the macro.

Application.RunAfterProcessDone(ProcessID, MacroName, ResourceName, ResourceData);

Application.RunAfterProcessDone2(ProcessID, MacroName, ResourceName, ResourceData);

Runs the XMetaL macro after process termination.

Returns: No return value.

Parameters:

- ProcessID: long - application process id for monitoring the "process terminated" event
- MacroName: string - macro name to run
- ResourceName: string - XMetaL Application level Custom Property name
- ResourceData: VARIANT - value of "ResourceName" property. "ResourceName" property is valid only during "MacroName" macro execution.

Helper methods:

Application. IsProcessRunning(ProcessID);

Checks if a process is running.

Returns: boolean true if process is running. otherwise false

Parameters:

ProcessID : long - application process id

Application. TerminateProcessEx (ProcessID, Parameter1, Parameter2, Parameter3);

Terminates a running process.

Returns: long - internal use value.

Parameters:

ProcessID : long - application process id to terminate

Combination of Parameter1, Parameter2, Parameter3

0 : 0: 0 - "brute force" to terminate process. Uses windows API TerminateProcess(...)

-1 : 0 : 0 - find window by process id and send WM_CLOSE message

-1: -1: 0 - find window by process id and post WM_CLOSE message

Parameter1: contains window handle (> 0)

Parameter1: 0 : 0 - send WM_CLOSE message to window

Parameter1: -1: 0 - post WM_CLOSE message to window

Parameter2: > 0 use windows API SendMessageTimeout(...)

::SendMessageTimeout(xxx, WM_CLOSE, 0, NULL, Parameter3, Parameter2, ...);

XMetaL APIs:

- Application. AddFileChangeMacro(filePath, macroName, propertyName, propertyValue)
- Application. RemoveFileChangeMacro (filePath, macroName)
- Application. IsFileChangeMacro (filePath, macroName)

Allow file changes on the local file system to be monitored and XMetaL macros run when events occur and XMetaL is in idle state.

A parameter that specifies the local file system folder and file name can include wildcard characters (for example, an asterisk (*) or a question mark (?)) in the file name. The "file changed" event triggers running the XMetaL macro " macroName" . XMetaL sets the "\$FileChangedPath" application-level custom property to the file path of the changed file. A VARIANT type optional parameter can also be passed to **Application**.

AddFileChangeMacro (...). The assigned XMetaL application-level custom property contains the parameter's value when an XMetaL macro is running. Both properties are valid only during the macro execution.

Application. AddFileChangeMacro(filePath, macroName, propertyName, propertyValue)

Monitors file change events and runs a macro when file content changes.

Returns: long - non zero if operation succeeded.

Parameters:

- filePath : string - local file system path to file for monitoring "file changed" event.
File name part can include wildcard characters (for example, an asterisk (*) or a question mark (?)) for monitoring changes in multiple files.
- macroName : string - macro name to run
- propertyName : string - XMetaL application-level custom property name
- propertyValue : VARIANT - value of "propertyName" property. "propertyValue" property is valid only during "macroName" macro execution.

Application. RemoveFileChangeMacro(filePath, macroName)

Removes the monitoring of a file change event.

Returns: long - number of objects that are monitoring "file changed" events. Object is identified by "filePath + macroName" string.

Parameters:

- filePath : string - local file system path to the file for monitoring the "file changed" event.
- macroName : string - macro name to run. If macroName is an empty string, then remove all macros associated with filePath.

Application. IsFileChangeMacro(filePath, macroName)

Checks if the file change event "filePath + macroName" is monitored.

Returns: long - number of objects that are monitoring "file changed" events. The object is identified by the "filePath + macroName" string.

Parameters:

- filePath : string - local file system path to file for monitoring "file changed" event.
- macroName : string - macro name to run. If macroName is an empty string, then return the number of all macros associated with filePath.

XMetaL API Application. RunMacroOnIdle(MacroName, mlsDelay, propertyName, propertyValue)

Runs the macro specified by the "MacroName" on idle.

Returns: long - 1 if macro was added to queue - 0 otherwise.

Parameters:

- MacroName : string - macro name to run
- mlsDelay: long - minimum delay interval in millisecond before macro can run
- propertyName : string - XMetaL Application level custom property name

- `propertyValue` : VARIANT - value of "propertyName" property. "propertyValue" property is valid only during "MacroName" macro execution.

XMetaL API Application. RunMacroOnIdle(MacroName, mlsDelay, propertyName, propertyValue)

pushes a macro into the queue of macros that XMetaL runs in idle state. An optional command line parameter can set a minimal delay interval in milliseconds before the macro can run. The VARIANT type optional parameter "propertyValue" can be passed to **Application. RunMacroOnIdle(...)**. The assigned XMetaL application-level custom property contains the parameter's value when the XMetaL macro runs.

This API prevents the interruption of time-sensitive actions or macros being execution (for example `xmetal ON_UPDATE_UI` event macros).

The XMetaL installation folder contains the helper application **RunXMetaLMacro.exe**, which runs a macro when (or if) XMetaL is running and in idle state.

```
RunXMetaLMacro.exe [-x=1] -mn="<macro name>" [-mp="<parameter name>"
-mv="<parameter value>"]
```

The mandatory parameter `-mn="<macro name>"` runs the macro.

The optional parameter `"-x=1"` forces `xmetal.exe` to launch if it is not running.

Optional parameters `-mp="parameter name"` and `-mv="parameter value"` type string define the XMetaL application-level custom property that is valid only when `<macro name>` macro is running.

The XMetaL installation folder contains the helper application **XMProjWait.exe**, which is a "minimal system resources consuming process" that can be terminated any time without affecting application integrity. Optional command line parameters can set a time interval in milliseconds after which the process terminates itself. It allows using it as an "external global, out of process variable or event" for avoiding concurrent access to XMetaL script resources.

The XMetaL Developer installation contains an example of using the "XMetaL External Event Handling API" in an XMetaL application-level customization. The example is located in the

`C:\ProgramData\SoftQuad\Developer\Samples\AsynchronousEventHandling\Author` folder. Copy the contents of this folder to the `Author` folder in the XMetaL Author 16.0 or higher installation folder.

The example can be demonstrated in the following test scenario:

Scenario:

1. Run the **File changed XMetaL API example** macro. It opens the modeless dialog that allows:
 - a. Select the word under the cursor or select text in the open document.
 - b. Save to a temporary file: selected text, information about the selected text location in an open document, an open document xml content .
 - c. Select and launch a third-party application and pass the temporary file as command line parameters.
 - d. Watch and change the status of running applications and "temporary file changed" events tracking.
2. When the "temporary file changed" event occurs:
 - a. Cancel listening to the "temporary file changed" event.
 - b. If the temporary file contains a list of suggestions for inserting/replacing the original text:
 - Show the XFT form modeless dialog with the notification.
 - Open or activate the original document and move the section to its original location.

- Show XFT form modeless dialog with options: to insert/replace new text, or start new instance of the third-party application with the new text as the parameter.

3. When the "application terminated" event occurs:

- a. Cancel listening to the "temporary file changed" event, and remove the "temporary file".

4. When the "XMetaL close" event occurs:

- a. Close all "temporary file changed" events.
- b. Terminate all applications.

File changed XMetaL API example (silent) macro runs the last third-party application with Active Document selected as parameter.

Scripts

Scripts provide access to XMetaL Author or XMetaL XMAX via an object model that is based on the Document Object Model (DOM) and the Microsoft Word VBA model.

The following definitions apply:

- A *macro* is a set of instructions consisting of condition statements, API commands, and functions.
- A *function* can run other macros but must be contained inside a macro, though it is not a macro itself.

Scripts are stored as components (for example, JScript files) in your customization. These components are visible through the Solution Explorer. You can create a new script or import an existing script. Scripts are deployed with your customization. You are advised to test your scripts before you deploy them.

For more information on scripting for XMetaL Author and XMetaL XMAX, see the *XMetaL Programmer's Guide*.

Table 5: Script properties

Name	Description
Name	The filename of the macro
Description	The description of the macro
FilePath	The filepath to (location of) the script
Hidden	Indicates whether or not the macro is hidden from the user
Language	The scripting language
MacroName	The name given to the script at the time of its creation
ShortcutKey	The Hot Keys assigned in XMetaL Author to launch the script
UseAs	Indicates the UseAs type

Macros folder

Scripts that are included in the Macros folder become XMetaL Author macros and are deployed in an MCR file that you create when you build your project.

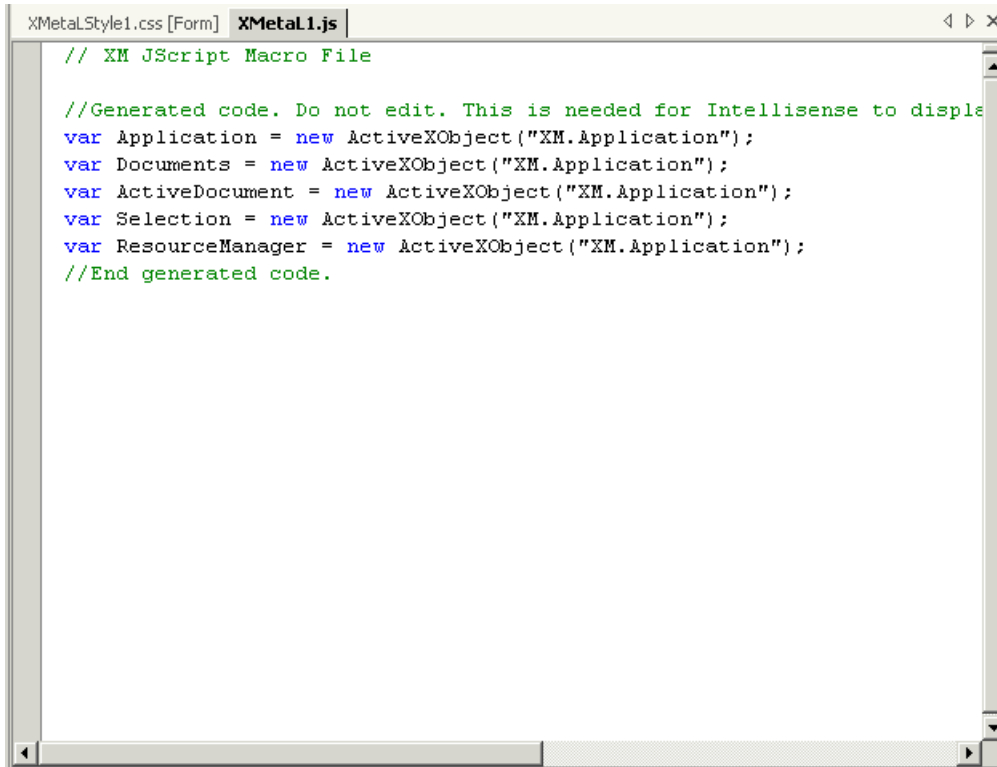
All scripts inside the MCR file are surrounded with <MACROS> and </MACROS> element tags. Individual scripts are denoted by <MACRO> and </MACRO> element tags.

```
<MACROS>
<MACRO>
.
// Macro one
.
</MACRO>

<MACRO>
.
// Macro two
.
</MACRO>
</MACROS>
```

Script editor

The script editor is part of the integrated development environment. It appears when you create or open a script.



```
XMetalStyle1.css [Form] XMetal1.js
// XM JScript Macro File

//Generated code. Do not edit. This is needed for Intellisense to display
var Application = new ActiveXObject("XM.Application");
var Documents = new ActiveXObject("XM.Application");
var ActiveDocument = new ActiveXObject("XM.Application");
var Selection = new ActiveXObject("XM.Application");
var ResourceManager = new ActiveXObject("XM.Application");
//End generated code.
```

Creating scripts

You have the option of inserting new scripts as separate files within your project or including them in the Macros folder. When you build your customization, all scripts included in the Macros folder become XMetaL macros.

When you create a script, you can specify the following options.

- **Macro Name.** You can select from the list of built-in macro events fired by XMetaL Author or XMetaL XMAX, or you can type a name for a new macro file.
- **Select scripting language for new macro.** By default, JScript and VBScript are provided. A single macro file can contain any of these types of scripts, or any combination of them.



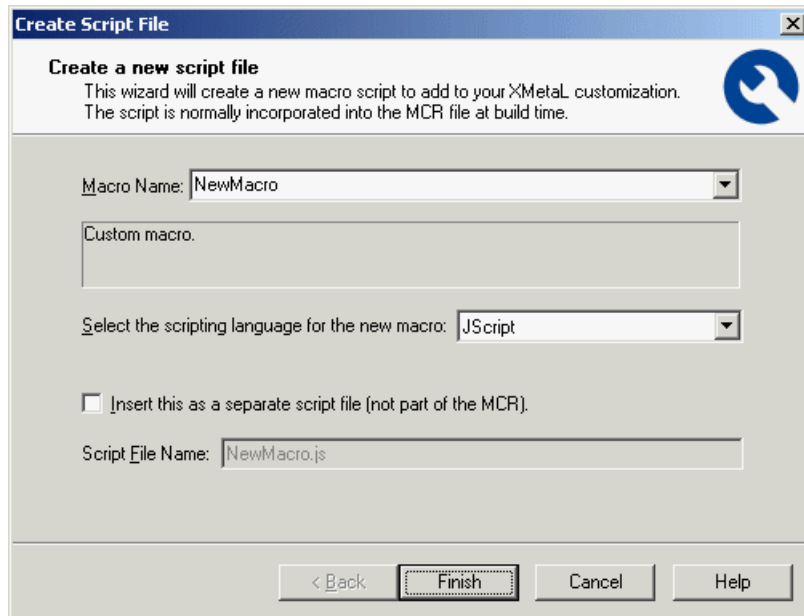
Note: You can add a Perl script or Python script to your XMetaL project (provided the scripting language is installed), however, there may be some Intellisense limitations and you may not be able to use breakpoints in these scripts.

- **Insert as separate script file.** If you want this script to be a separate file in your project, check this option and type a name. The new file appears as a standalone file that is part of your project. However, it is not added to the Macros folder and it is not treated as an XMetaL macro.

Create a script

You create a new script through the Create Script File dialog box. After you create the script, you can edit it in the script editor.

1. Select your project in the Solution Explorer.
2. Click **Project > Add New Item** and select a **Macro Script** template. The **Create Script** dialog box opens.



3. Type a name, select the scripting language, and set the Insert as separate script file option.
4. Click **Finish**.
The script editor opens. Some script is automatically added to the macro to enable IntelliSense functionality. Do not delete or modify this automatically-generated script; simply begin your own script below the existing code.

Importing scripts

You can import individual scripts or an entire file of scripts (an MCR file containing multiple scripts) into your XMetaL customization.

Individual scripts are stored as components and are visible in the Solution Explorer. When you import a script, you have the option of including it in the Macros folder. When you build your customization solution, scripts within the Macros folder become XMetaL macros and are stored in an MCR file.

You can also import all of the scripts within an existing MCR file. When you import scripts this way, they are added to the Macros folder.

Import a script

1. Select your project in the Solution Explorer.
2. Click **Project > Add Existing Item**.
3. Browse to the folder containing your script and click **Open**.

The Import as Macro dialog opens with the fields you can use to control the behavior of the macro.

Import scripts from an MCR file

1. Select your project in the Solution Explorer.
2. Click **Project > Import XMetaL Macros**.
3. Browse to the folder containing the MCR file and select it.
4. Select the macros you wish to add and click **Add**.

Testing scripts

You can test your script within a debugging session without having to deploy your customization.



Note: This information applies to document-level applications only.

In order for your scripts to properly run in the target application (XMetaL Author or XMetaL XMAX), you may first have to set the build properties.

There are some conditions during a debugging session when you cannot edit an XMetaL script. Scripts that create fatal crashes of the scripting engine may result in instability and crashes in XMetaL as well.

For more information on editing scripts during a debugging session, See Debugging Customizations.

Importing data

You can import data from a database using the Import Table dialog. This dialog is implemented using a macro. You can update data you have imported also using a macro.

Using the Import Database dialog, you can create a table in your document based on the contents of a database or spreadsheet file. When you create a query, the parameters are saved to a file that can be used to refresh the data.

To allow users access to this functionality, you must add macros to your customization. For an example implementation, see the **Import Table** and **Update Table** macros in the sample macro file `Macros\journalist.mcr`. These macros may be customized for your DTD and local file system and attached to menus or toolbars.

The Import Database dialog box is implemented by the DBImport interface.

In order to use the Import Database dialog, the following components must be installed *before* you install XMetaL:

- Microsoft Data Access Components (MDAC). If you do not have MDAC installed when you install XMetaL, the XMetaL installer offers to install it.
- Windows Scripting Engine. This is installed when you install Internet Explorer; you can also download the most recent version of the scripting engine from [here](#).

For more information about the DBImport interface, see the *XMetaL Programmer's Guide*.


Converting Microsoft Word documents

One method of converting Microsoft Word documents to XML is through a Visual Basic script. You can see a sample implementation in the Journalist customization. You can create your own solution for converting Word documents by adapting the Journalist example.

The Journalist customization includes a macro that converts the paragraph and character styles used in your MS Word document, as well as various Word objects such as graphics, lists and tables, to specific elements in the Journalist DTD. Any content within the Word document that has been tagged with a style not supported by the macro is converted to a processing instruction.

The macro script is contained in the following VBScript file:

```
..\XMetaL\Author\macros\journalist_openword.vbs.
```

To see how the script works, create a new document using the Journalist DTD and click the Open Word Document  on the toolbar. You can convert the following sample Microsoft word document:

```
..\XMetaL\Author\Samples\Cameras\Word\CamerasInFocus.doc.
```

PDF and HTML previewing and printing

Your XMetaL Author customization can include support for PDF and HTML previewing and printing. This functionality is added via macros. An example implementation is included in the Journalist customization.

You can customize XMetaL to print, view, and save documents in Adobe(TM) PDF and HTML format. The macros you create should be JScript and they should be included in the MCR file. (Ensure that the **Insert this as a separate script file** option is unchecked.) The table below indicates the method that should be added to each macro following the auto-generated script. See the *XMetaL Programmer's Guide* for more information.

Before you begin, check the following:

- You have Adobe Acrobat(TM) Reader (with Web browser integration enabled) installed on your computer
- Your customization does not include the On_Before_Document_Preview macro (if it does, you must remove it)

Table 6: PDF support macros

Create this macro	Include this method	Notes
Setup PDF	XMLToPDFSetup	This macro allows you to specify PDF formatting. After you have deployed your customization, you should run this macro before you save as PDF for the first time or if you want to change the PDF settings.
Save as PDF	saveAsPDF	This macro allows you to save as PDF. You should customize the XMetaL Author user interface to allow easy access to this macro through a toolbar button, menu item, or shortcut key.
View PDF	previewPDF	This macro allows you to view and print PDF. You should customize the

Create this macro	Include this method	Notes
		XMetaL Author user interface to allow easy access to this macro through a toolbar button, menu item, or shortcut key.

Table 7: HTML support macros

Create this macro	Include this method	Notes
Setup HTML	XMLToHTMLSetup	This macro allows you to specify HTML formatting. When you run this macro, it creates an XSLT file used to generate HTML output from an XML document. The file is saved in the same folder as the CSS file.
Save as HTML	saveAsHTML	This macro allows you to save as HTML. You should customize the XMetaL Author user interface to allow easy access to this macro through a toolbar button, menu item, or shortcut key.
View HTML	previewHTML	This macro allows you to view and print HTML. You should customize the XMetaL Author user interface to allow easy access to this macro through a toolbar button, menu item, or shortcut key.

DTDs and schemas

A *Document Type Definition* (DTD) or *schema* forms the basis of your customization of XMetaL. After you have planned the workflow, you need to create a DTD or schema. You typically do this using a third-party tool. You have the following options:

- **Create a new DTD.** You need to determine the elements and attributes required by your documents. You then need to define the document structure.
- **Modify an existing DTD.** You may already have a DTD or schema that you want to use in your customization, but it requires changes to the elements and attributes or structure.
- **Use an existing DTD without modifications.** As with the previous options, you need to have an understanding of the elements attributes and structure.

You can edit elements and attributes within the schema viewer in the XMetaL Developer integrated development environment.

Although XML files can exist as well-formed standalone documents, most XML documents that you edit in XMetaL Author will be associated with either a document type definition or schema. DTDs and schemas facilitate the exchange of information, enabling it to be easily passed between systems and people. SGML files must be associated with a DTD or rules file. Schemas cannot be used with SGML documents.

A DTD is a file that describes document content and structure by means of declarations written in a formal notation defined in the SGML and XML standards. A DTD defines the names of elements that can be used in documents and describes their hierarchy.

Schema support

Like DTDs, schemas describe the document content and structure, but they are written in XML. XMetaL Author supports schemas with the following limitations:

- Identity-constraint definitions are ignored
- The instance attributes `xsi:nil` and `xsi:type` are ignored, and cannot be edited in Normal or Tags On view

Wildcards are not fully supported in XMetaL:

- The `xsd:anyAttribute` is not supported
- The `xsd:any processContents=skip` and the `xsd:any processContents=lax` process contents controls are not supported
- `xsd:any processContents=strict` is supported, but the elements used must be imported using `<xsd:import>`

Creating a DTD

A DTD can consist of a group of files that includes a root file.

A DTD can consist of the `.dtd` file and one or more files including:

- DTD fragments

- Entity declarations (for example, in catalog files)
- An SGML declaration
- An attribute description file

All required DTD fragments and entity files must be at the locations specified by the identifier used to refer to them. For example, a DTD fragment may be referenced in the following entity declaration in the DTD:

```
<!ENTITY % calstdtd PUBLIC "CALs Table DTD" "dtds/cals.dtd">
%calstdtd;
```

In this case, the required fragment should be in the file `cals.dtd`, located in the folder `dtds`, which is in the same folder as the DTD file.

The SGML declaration should be located in the same folder as the DTD file and should carry the same name as the DTD and the extension `.dec` or `.dcl`.

Similarly, the attribute description file should be located in the same folder as the DTD file and should carry the same name as the DTD and the extension `.att`.

Modifying your DTD

You can modify elements and attributes using the XMetaL Developer integrated development environment.

When you open your DTD or schema through the Solution Explorer, you can edit properties of the elements and attributes in your DTD or schema.


XMetaL supports the following content types:

- Mixed content (a mixture of element and character data)
- Element content
- Character data (CDATA)
- Parsed character data (PCDATA)
- Any content (any or none of the different sets above)
- Empty (no content)

Rules files

When you build your customization, the DTD or schema is compiled into a smaller binary file called a *rules* file. This binary file is deployed with your customization and used by XMetaL Author or XMetaL XMAX when determining the rules definition for the customization.

Rules files for XML files have a `.rlx` extension, those for SGML files have a `.rls` extension, and those for schemas have a `.rld` extension. In most cases, no extra work is required to create the compiled rules file, as the compilation and inclusion in the XAC file is automatic. However, you can choose to manually compile a rules file using either Rules Maker or the `mkrls` command-line tool.

 **Note:** When you open or create a document that uses a DTD for which there is no corresponding rules file, XMetaL compiles a rules file and uses it instead of the DTD. If the DTD is changed, the rules file must be deleted so that XMetaL can automatically recompile a new rules file. This rules file has the same format as a rules file generated by Rules Maker.

Using Rules Maker

You can start Rules Maker from `..\XMetaL\Developer\bin\XMmkrules.exe`. You need to select a DTD. You can also choose other files, identifiers, and options.

Using the `mkrls` command-line tool

You can start the Rules Maker command line tool by opening a command window at `..\XMetaL\Developer\bin`. Type `mkrls` followed by options, followed by the DTD filename.

Table 8: Command line options

Option	Description
<code>-o rulesfile_name</code>	Defaults to <code>dtlname.rlx</code> if the <code>-x</code> option is used, and <code>dtlname.rls</code> otherwise
<code>-E catalog_file</code>	OASIS catalog file
<code>-S SGML_declaration_file</code>	SGML declaration file
<code>-h attribute_help_file</code>	Attribute description file
<code>-n root_element_name</code>	Root element (If the first element declared is <i>not</i> the logical top-level element, you should use this option)
<code>-a</code>	Do not report ambiguous content
<code>-c</code>	Do not check elements used
<code>-q</code>	Do not display warnings
<code>-t</code>	Create attribute description file

Document type declarations

An XML or SGML document starts with a *document type declaration* that associates it with a specific DTD or schema.

Here is an example of a document type declaration:

```
<DOCTYPE BOOK PUBLIC "-//Justsystems//Book v1.0//EN" "book.dtd">
```

The DOCTYPE keyword is followed by the *document type name*. By default, this is the top-level element in the DTD or rules file.

External identifiers

The DOCTYPE keyword associates the document with a DTD or rules file using an *external identifier*. An external identifier consists of the keyword SYSTEM or PUBLIC, followed by one of the following:

- A *system identifier*
- A *public identifier* followed by a system identifier
- A public identifier (SGML only)

The system identifier can be a filename or URL.

Each identifier consists of a string of characters enclosed by quotation marks. The system identifier is generally the DTD or rules file, and the public identifier is an arbitrary identifier. Usually, DTDs that are used by a large number of organizations have a standard public identifier.

Examples

The following document type declarations can be used to refer to the same DTD. The name after the word DOCTYPE is usually the name of the top-level element in the rules file; in these examples, that element is BOOK.

```
<!DOCTYPE BOOK PUBLIC "-//Justsystems//Book v1.0//EN" "book.dtd">
```

The keyword PUBLIC indicates that the first identifier that follows it is the public identifier. This document type declaration refers to a DTD that has the public identifier `-//Justsystems//Book v1.0//EN` and the system identifier `book.dtd`.

```
<!DOCTYPE BOOK SYSTEM "book.dtd">
```

The keyword SYSTEM indicates that the identifier that follows it is the system identifier. If the external identifier starts with SYSTEM, there cannot be a public identifier. This document type declaration refers to a DTD that has the system identifier `book.dtd`.

There is a third possibility for SGML documents only:

```
<!DOCTYPE BOOK PUBLIC "-//Justsystems//Book v1.0//EN">
```

The keyword PUBLIC indicates that the identifier that follows it is the public identifier. In this example, there is no system identifier. (DOCTYPE declarations in XML documents **must** include a system identifier.) This DOCTYPE refers to a DTD that has the public identifier `-//Justsystems//Book v1.0//EN`.

Internal subset

Instead of, or in addition to, the external identifier, the document type declaration can have an *internal subset* containing further declarations.

The internal subset is enclosed in square brackets and follows the document type name and the external identifier (if there is one). Here is an example:

```
<DOCTYPE Article SYSTEM "journalist.dtd" [
<ENTITY Title "Weasel populations in a forest in Poland">
...
]>
```

The internal subset can contain attribute list and entity declarations. Declarations in the internal subset are read *before* those in the external DTD or rules file; therefore, they override any external declarations. Duplicate element declarations are not allowed. Attribute list declarations specifying different attributes of the same element are combined.

A document type declaration can omit the external identifier thus making the document's DTD internal, that is, completely contained in the internal subset. For example:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE Article [
<!Element Article (Title, Sect1+)>
<!Element Title (#pcdata)>
<!Element Sect1 (Title,Para+)>
<!Element Para (#pcdata)>
<!Attlist Article Id ID #IMPLIED>
```



```
]>
<Article> ... </Article>
```

The internal subset can also refer to an external DTD using a parameter entity reference. In the following example, `%journalist.dtd;` is the parameter entity reference:

```
<?xml version="1.0"?>
<!DOCTYPE Article [
<!Entity % journalist.dtd SYSTEM "journalist.dtd">
%journalist.dtd;
]>
<Article> ... </Article>
```

When users create an entity with the XMetaL Author **Tools** menu, the declarations are placed in the document's internal subset. If the internal subset contains any declarations other than entity declarations, they are read-only in Tags On and Normal views, and the entity-creation commands are unavailable.



Note: Internal subsets in SGML documents are not supported.

Mapping identifiers

XMetaL uses the OASIS catalog mechanism to associate an external identifier in a document type declaration or external entity declaration with a DTD, rules file, or entity file. This mechanism can also be used to specify an SGML declaration or associate an entity name with a filename.

You typically use the catalog mechanism in the following situations:

- If the document type declaration contains only a public identifier
- If the DTD or rules file is not stored in the `Rules` folder
- If the system identifier in the document type declaration does not match the DTD or rules file that you want to use

If XMetaL cannot resolve the external identifier using the catalog mechanism, it tries to resolve the external identifier using the following methods, in the following order:

1. It attempts to find an entry in the external identifier map file, `extid.map`. This mechanism is provided for backward compatibility with previous versions of XMetaL, and can be disabled.
2. It attempts to retrieve the system identifier as a URL (relative to the document instance).
3. It attempts to retrieve the system identifier as a file path (relative to the document instance).

For the complete specification of the catalog mechanism, see [OASIS Technical Resolution 9401:1997](#).

Catalog files

XMetaL reads one or more *catalog files*, which can contain several types of entries. It reads these files until it finds a matching entry. XMetaL supports a set of keywords that can be used in a catalog file.

BASE absolute-path

Interpret relative paths in the catalog file as relative to `absolute-path` instead of relative to the location of the catalog file (the default). For example, a catalog file may contain these entries:

```
BASE "C:\Windows\Applications\XMetaL"
SYSTEM "mydoc.dtd" "DTDs/mydoc.dtd"
```

In this case, the full path for `mydoc.dtd` is
`C:\Windows\Applications\XMetaL\DTDs\mydoc.dtd`.

CATALOG catalog-file

If a matching entry is not found in the current catalog file, search in `catalog-file`. If no matching entry is found there, return to the normal catalog sequence.

DELEGATE partial-public-id catalog-file

If searching for a public identifier match, and `partial-public-id` matches a substring of the identifier starting at the first character, search in `catalog-file` for a match and do not return to the normal catalog sequence.

DOCTYPE document-type-name filename

If the document type declaration specifies `document-type-name`, then use `filename` as the DTD file.

ENTITY entity-name filename

Use `filename` as the replacement text for the external entity `entity-name`.

LINKTYPE public-id filename

If the SGML document contains a LINKTYPE declaration that contains `public-id`, then use `filename` as the replacement text.

NOTATION notation-name filename

Use `filename` as the content of the notation `notation-name`.

OVERRIDE YES|NO

If YES is specified, public identifiers and entity names are preferred to system identifiers when attempting to find a match for an external identifier. If NO is specified, and an external identifier contains a system identifier, then the public identifier (if there is one) and entity name is be used when attempting to find a match for the external identifier.

PUBLIC public-id filename

If an external identifier (for example, in an entity declaration or document type declaration) contains the public identifier `public-id`, use `filename` to resolve the external identifier.

SGMLDECL filename

Use `filename` as the SGML declaration for the SGML file.

SYSTEM system-id filename

If an external identifier (for example, in an entity declaration or document type declaration) contains the system identifier `system-id`, use `filename` to resolve the external identifier.

-- (comment notation)

Declarations can contain comments, which start and end with a double hyphen (--). For example:

```
-- Catalog entries for the Caracas project --
```

Resolving catalog file entries

Catalog file entries can be resolved using a public identifier, a system identifier, or an entity declaration.

Using a public identifier

```
PUBLIC "ISO 8879-1986//ENTITIES Added Latin 1//EN" "isolat1.ent"
```

The PUBLIC entry in the catalog associates the public identifier ISO 8879-1986//ENTITIES Added Latin 1//EN with the filename isolat1.ent. This catalog entry can be used to resolve the following entity declaration in a DTD:

```
<!ENTITY % isolat1 PUBLIC "ISO 8879-1986//ENTITIES Added  
Latin 1//EN">  
%isolat1;
```

When XMetaL encounters the %isolat1 entity reference, it scans the declaration of isolat1 and finds a public identifier. It then looks in the catalog file for a public entry matching the same identifier. The filename specified in this entry (isolat1.ent) is then used as the replacement for the entity reference.



Note: Filenames can contain absolute or relative paths or URLs. Relative filenames are interpreted as relative to the location of the catalog file, unless the catalog file contains a BASE entry.

Using a system identifier

```
SYSTEM "sqdoc.dtd" "sqdoc-xml.dtd"
```

Here, the system catalog entry associates the system identifier, sqdoc.dtd, with the filename sqdoc-xml.dtd. This catalog entry is used to resolve the following document type declaration:

```
<!DOCTYPE DOC SYSTEM "sqdoc.dtd">
```

When XMetaL reads this declaration in an SGML or XML document, it finds the system identifier, sqdoc.dtd, and looks in the catalog file for a system entry matching that identifier. The filename found (sqdoc-xml.dtd) is used as the DTD file for the document.

Using an entity declaration

```
ENTITY face1 "c:\project1\smallfaces\face1.gif"
```

The entity catalog entry associates the entity name, face1, with the filename face1.gif. When XMetaL encounters a reference to the external entity, it scans the declaration of face1 for a system or public identifier. It then reads the catalog file, looking for system or public entries specifying these identifiers. If it does not find any such entry, it then looks for a matching entity entry. The file face1.gif is used as the replacement for the entity reference.

Catalog support for schemas

If you wish to use catalogs with schemas, you need to be aware of special syntax requirements both in the catalog entry and in your schema.

In order to support catalogs for schemas, you must set up your catalog for the public identifier. For example, suppose your document looks like this:

```
<Article xmlns="AAA//BBB CCC//XML">
</Article>
```

If you want to validate against Article.dtd or Article.xsd, your catalog file must have the following entry:

```
PUBLIC "AAA//BBB CCC//XML" "Article.xsd"
```



Note: You must ensure that Article.xsd uses a TargetNamespace that is a fixed attribute of 'xmlns'.

Finding catalog files

By default, XMetaL uses the document name to find a catalog file. There is a default searching order. You can also specify alternate catalog files.

For example, if the current document is called `docname.xml` and it is located in the folder called `docfldr`, XMetaL searches for the following files, in the following order:

1. `docfldr\docname.soc` (a file called `docname.soc` in the same folder as the XML/SGML document)
2. `docfldr\catalog` (a file called `catalog`)
3. `docfldr\catalog.soc`
4. `Rules\catalog` (a file called `catalog` in the Rules folder)
5. `Rules\catalog.soc`

The *root catalog* may have links to other catalog files. You can specify alternate catalog files from within a catalog file using the CATALOG and DELEGATE keywords.

A catalog file entry such as

```
CATALOG "catalog2"
```

specifies an alternate catalog file. If XMetaL does not find a matching entry in the current catalog file, it reads the alternate file. If no matching entry is found, XMetaL continues with the next catalog file in the normal sequence. A catalog file can contain several CATALOG entries.

A catalog file entry of the form

```
DELEGATE public-id-prefix catalog-file
```

can be used if XMetaL is currently attempting to match a public identifier, though PUBLIC entries take precedence. If XMetaL encounters one or more DELEGATE lines (in a single catalog file) in which the public-id-prefix matches a substring of the public identifier in question (starting at the first character) then XMetaL looks for matching entries in the catalog files specified by the DELEGATE entries. It does not return to the normal sequence of catalog files.

Giving priority to system or public identifiers

The system identifier (if there is one) in an external entity declaration is generally the actual name of the file represented by the entity. Sometimes, however, this may not be the case, and the catalog mechanism provides the option of using other means to obtain the filename.

If the catalog file contains a system entry matching the system identifier in question, then the filename specified in that entry is used to resolve the entity reference.

If the catalog file contains the entry

```
OVERRIDE YES
```

and there is no matching system entry, then

- If the entity declaration contains a public identifier, and a matching public entry is found, the filename specified in that entry is used to resolve the entity reference.
- If a matching entity entry is found, the filename specified in that entry is used to resolve the entity reference.
- Otherwise, the system identifier is used to resolve the entity reference.

If the catalog file contains the entry

```
OVERRIDE NO
```

and there is no matching system entry, then the system identifier is used to resolve the entity reference. In this case XMetaL does not attempt to match the public identifier or entity name.

An OVERRIDE YES or OVERRIDE NO entry is in effect until the end of the current catalog file, or until an OVERRIDE entry with the opposite setting is encountered.

The default mode (YES or NO) is set using the `OASIS_override` setting in the XMetaL configuration file. The default setting is true (YES).

External identifier map file

If the catalog cannot resolve a public identifier, XMetaL uses the *external identifier map file* for mapping the external identifier in a document type declaration to the name and location of a DTD or rules file.

This external identifier map file is used in the following situations:

- If the document type declaration contains only a public identifier
- If the DTD, schema, or rules file is not stored in the `Rules` folder
- If the system identifier in the document type declaration does not match the DTD or rules file that you want to use
- If you want to use regular expressions to match a set of public or system identifiers and map them to a set of filenames

The external identifier map file is, by default, `extid.map`. You can use a different file by specifying a value for `extid_map` in the XMetaL configuration file.

The external identifier map file consists of lines in this form:

```
public-id system-id DTD/rulesfile
```

The first two values are strings or patterns that match the public and system identifiers respectively. The third value is the name of the DTD or rules file that these identifiers refer to. Here is an example:

```
"-//Justsystems//Book v1.0//EN" ! book.dtd
```

If you open a file whose document type declaration contains the public identifier `-//Justsystems//Book v1.0//EN`, XMetaL scans the external identifier map file until it comes to the line in the example. It sees that the two identifiers match, and therefore it looks for `book.dtd`. The exclamation mark (!) is a special character that means 'match any identifier', so in this example it does not matter what the system identifier is, or if one is present.



Note: You can disable the external identifier map mechanism by setting `use_extid_mapping` to `false` in the XMetaL configuration file.

Creating an external identifier map file

Map file entries can specify a public identifier, an alternate DTD or rules location, or they can map one system identifier to another.

XMetaL needs to refer to the external identifier map file, `extid.map`, only when the document type declaration does not have a system identifier that is the same as the filename of a DTD or rules file stored in the `Rules` folder.

Using a public identifier only

For SGML documents, the document type declaration may contain only a public identifier:

```
<DOCTYPE BOOK PUBLIC "-//XMetaL//Book v1.0//EN">
```

If you want the DTD or rules file to be identified by the public identifier only, you should include an entry similar to the following in the external identifier map file:

```
"-//XMetaL//Book v1.0//EN" ! book.dtd
```

This matches `book.dtd` to the public identifier, regardless of whether the system identifier is present, or what it is.



Note: XML external identifiers must contain a system identifier.

Using an alternate DTD or rules location

If you store your DTD, schema, or rules file somewhere other than the `Rules` folder, you need to tell XMetaL the location.

You can put the rules file location in the document type declaration explicitly:

```
<DOCTYPE BOOK SYSTEM "C:\DTDs\book.dtd">
```

Alternatively, you can use the external identifier map file, `extid.map`, to point to the location of the DTD or rules file.

```
"-//Justsystems//Book v1.0//EN" ! "C:/DTDs/book.dtd" ! "book.dtd" "C:/DTDs/book.dtd"
```

The first example maps a public identifier to a DTD; the second maps a system identifier to a DTD. You can use either form.

Mapping one system identifier to another

If the system identifier specifies `dtlname.dtd`, XMetaL looks, by default, for the rules file `dtlname.rlx` or `dtlname.rls`. If the system identifier does not correspond to the desired DTD or rules file in this way, you must create an entry in the external identifier map file.

The system identifier in the document type declaration may specify a DTD name, as in this example:

```
<DOCTYPE BOOK SYSTEM "book.dtd">
```

For example, if you want to use the rules file `realbook.rlx`, instead of `book.rlx`, you can either change the document type declaration to refer to the rules file or create an entry in the external identifier map file that tells XMetaL which rules file corresponds to the DTD name.



Note: If the document type declaration contains a reference to a rules file (instead of a DTD), the document type declaration no longer conforms to the XML specification.

To map a public identifier to a file name, use an entry like this example:

```
! "book.dtd" "realbook.rlx"
```

If you use several rules files, and there is a regular correspondence between DTD names and rules file names (other than the default correspondence between `.dtd` and `.rlx` or `.rls` files), you can map them all using one entry.

For example, if you use names of the form `anything.dtd` for all your DTD file names, and call the corresponding rules files `anything.rules`, the following line in the external identifier map tells XMetaL to use the rules file that corresponds to the DTD (regardless of whether the public identifier is present, or what it is):

```
! (*.*)\.dtd \1.rules
```

Language support

XMetaL supports multi-language documents that you can spell-check in a desired language. You can specify a language in the `xml:lang` attribute for elements in the DTD. By default, the value of the `xml:lang` attribute for an element is inherited from its parent.

For information on specifying languages in XML, see the [W3C website](#).

Example

Suppose `<TopLevel>` is the top-level element. If the default language is American English, the attribute list needs to include the following:

```
<!ATTLIST TopLevel
xml:langNMTOKEN"en-us"
>
```

To allow an element to override the default language, add the following attribute:

```
<!ATTLIST Para
xml:langNMTOKEN#IMPLIED
>
```

SGML declaration

An *SGML declaration* contains information about the character set, markup delimiters, quantity settings, and special markup features.

If your DTD exceeds the defaults for such quantities as the length of an element or attribute name, or you want to turn on an optional feature such as tag minimization, you must specify the desired values and features in your SGML declaration. There is no need to use an SGML declaration with DTDs for XML files.

You can provide an SGML declaration in the following ways:

- If there is a file `dtlname.dec` or `dtlname.dcl` in the same folder as the DTD `dtlname.dtd`, it is used as the SGML declaration for all files that use that DTD
- You can specify an SGML declaration when you compile a rules file

In XML files, XMetaL writes element, attribute, and entity names in the case that matches their declarations (upper, lower, or mixed).

In SGML files, element and attribute names are written in uppercase if the NAMECASE, GENERAL parameter is set to YES (the default). If it is set to NO, elements and attributes are written in the case they were declared in. Entity names are written in the case they were declared in if the NAMECASE, ENTITY parameter is set to NO (the default). If it is set to YES, they are written in uppercase.

For more information on SGML declarations, see the [Cover Pages website](#).

Sample SGML declaration

```
<!SGML "ISO 8879:1986"
-- Copyright Justsystems, 2006 --
CHARSET
BASESET "ISO 646-1983//CHARSET International Reference
Version (IRV) //ESC 2/5 4/0"
DESCSET
09UNUSED
929
11 2UNUSED
13 113
14 18 UNUSED
32 95 32
1271UNUSED
BASESET "ISO Registration Number 109//CHARSET
ECMA-94 Right Part of Latin Alphabet Nr. 3//ESC 2/13 4/3"
DESCSET
12832 UNUSED
160532
16589 32
2541127
2551UNUSED
CAPACITY PUBLIC
"ISO 8879-1986//CAPACITY Reference//EN"
SCOPE DOCUMENT
SYNTAX
SHUNCHAR NONE
BASESET"ISO 646-1983//CHARSET International Reference
Version (IRV) //ESC 2/5 4/0"
DESCSET
01280
FUNCTION
RE10
RS13
SPACE 32
NAMING
LCNMSTRT " "
UCNMSTRT " "
LCNMCHAR "-._"
UCNMCHAR "-._"
NAMECASE
GENERALYES
ENTITYNO
DELIM
GENERALSGMLREF
SHORTREF SGMLREF
NAMESSGMLREF
QUANTITY SGMLREF
```



```
NAMELEN64
LITLEN2048
FEATURES
MINIMIZE
DATATAGNO
OMITTAGNO
RANKNO
SHORTTAG YES
LINK
SIMPLENO
IMPLICIT NO
EXPLICIT NO
OTHER
CONCURNNO
FORMALNO
APPINFONONE
>
```

Attribute description files

An attribute description file contains descriptions of attributes. These are displayed at the bottom of the Attribute Inspector when you click an attribute name in XMetaL Author.

The attribute description file consists of entries of the form:

```
Element Attribute "Help String"
```

This example supplies a help string for the SECURITY attribute of PARA:

```
Para Security "Security level"
```

Attribute description files can be used with DTDs or compiled rules files. The attribute description file should be in the same folder as the DTD (by default, the folder `Rules`) and it must have the same name as the DTD and the extension `.att`.

Forms

Forms provide a way to control content and simplify the content creation process for users of your XMetaL customization. You can create forms and integrate them in your customization using the XMetaL Forms Toolkit (XFT). Through forms, you can exchange data with your own XML documents or with an external database.

Using forms, you can control what information users enter into an XML document, thereby standardizing information and reducing entry errors. You can also hide the details of XML markup from the user, making the job of entering content easier.

The XMetaL Forms Toolkit is designed to work seamlessly with XMetaL. It provides the tools to create forms that can be run from within XMetaL either as modal dialog boxes or embedded within a document. In addition, you can create forms that are bound to XML content, such as elements and attributes, and apply business logic via scripting.

You create forms within the Form Layout Editor, which is an integral part of XFT. Your forms contain controls, which have certain properties and events assigned to them. When you create a form, you drag control objects from the Object Bar onto the workspace and modify their properties through the Property Sheet. The set of tools allows you to precisely position the objects.

Form filename extension

By default, forms are saved with the `.xft` extension. For easy access, save your forms in the `..\XMetaL\FORMS` folder.

Testing forms

You can test and de-bug your forms in the Form Layout Editor by clicking **View > Execute Form**.

Script events

You can define specialized behavior for objects using JScript and VBScript events. All objects have the following events:

- OnInitialize
- OnTerminate
- OnClick

There are also a number of optional events that may be linked in:

- OnDbClick
- OnMouseDown
- OnMouseMove
- OnMouseUp
- OnDragOver (You must set "Effect = 1" to enable OnDragDrop.)
- OnDragDrop
- OnBlur
- OnFocus

Interfaces

All form objects, including the form background and frame, have properties. For more information, see the *XMetaL Programmer's Guide*.

Content mapping model

You can use XFT-specific script events, properties, and methods to transfer data between an XML document and a form. For more information, see the *XMetaL Programmer's Guide*.

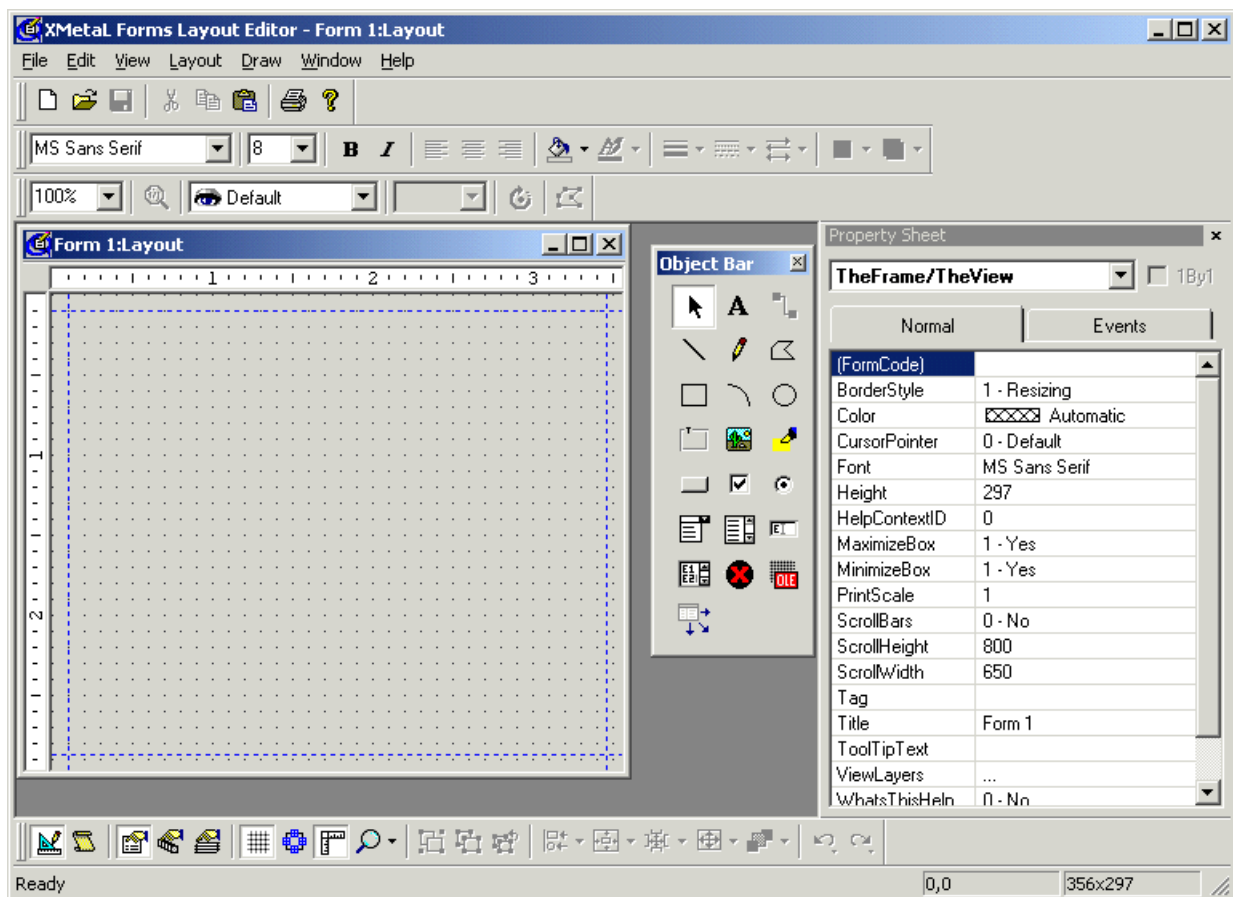
XMetaL Forms Toolkit

The XMetaL Forms Toolkit (XFT) includes an easy-to-use interface for creating and testing forms, a data source control for binding data from a database to your XML document, and a wizard that lets you associate a form with an object in your customization.

Form Layout Editor

The Forms Layout Editor is included in the XMetaL Forms Toolkit. You use this tool to create and edit forms for your customization. You can use the interface provided to position controls and determine their behavior.




















By default, the Form Layout Editor is installed at `.. \XMetaL\Developer\bin\XFLayout.exe`.



Object Bar

The Object Bar appears by default in the Forms Layout Editor interface. It contains all of the objects that you can use on your form. You can add objects to your form by clicking and dragging.

Table 9: Object Bar

Control	Description
	Static (non-editable) text
	Line drawing tool
	Freehand drawing tool
	Connector
	Rectangular border drawing tool
	Elliptical arc drawing tool
	Ellipse drawing tool
	Multi-purpose frame to group objects (with heading text)
	Bitmap
	Highlight
	Button with pre-defined action
	Check box
	Radio button set
	Combination text box and pull-down list
	List box with scroll bar
	Edit control for all variables (text box)
	Multi-line edit control for all variables (multi-line text box)
	ActiveX control
	Data Source control

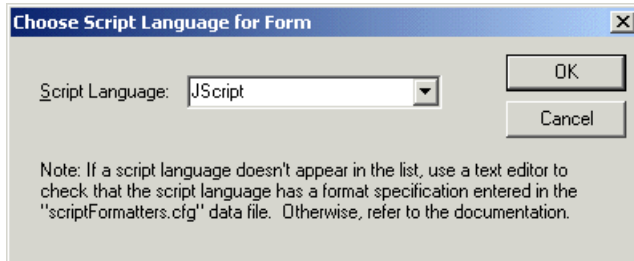
Form design tips

You can create controls that dynamically resize or reposition themselves when a form is resized using the `FlexHorizontal` and `FlexVertical` properties. For buttons, text, check boxes, and radio buttons, and other controls that should not change size, specify a value of 1 (Shift). In order to allow the user to take advantage of extra space created by a newly expanded form, specify a value of 2 (Expand) for controls such as edit boxes and multi-edit boxes. For more information, see the *XMetaL Programmer's Guide*.

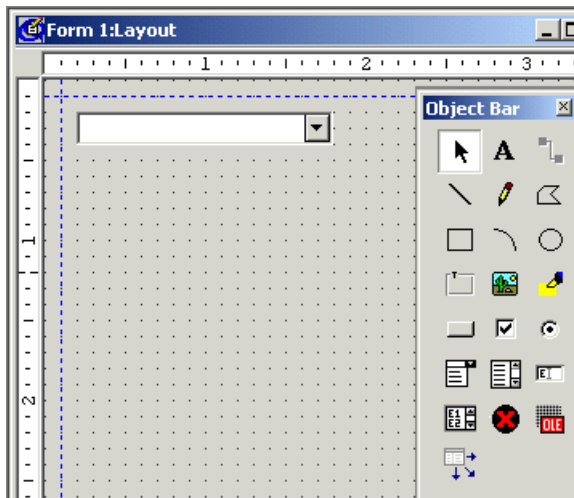
Create a form

You create forms using the Form Layout Editor. The process includes choosing a scripting language, adding objects, and specifying properties. After you have finished, you can associate the form with an object in your customization using the XFT Form Wizard.

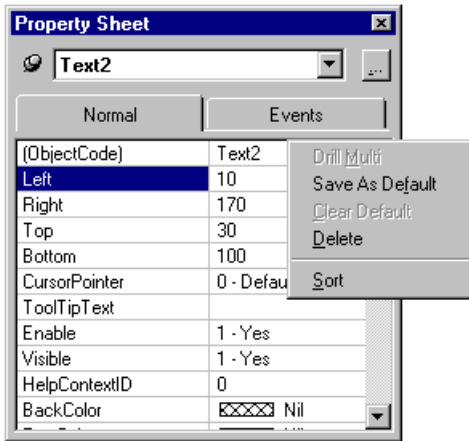
1. In the XMetaL Form Layout Editor, click **File > New** and choose a scripting language.



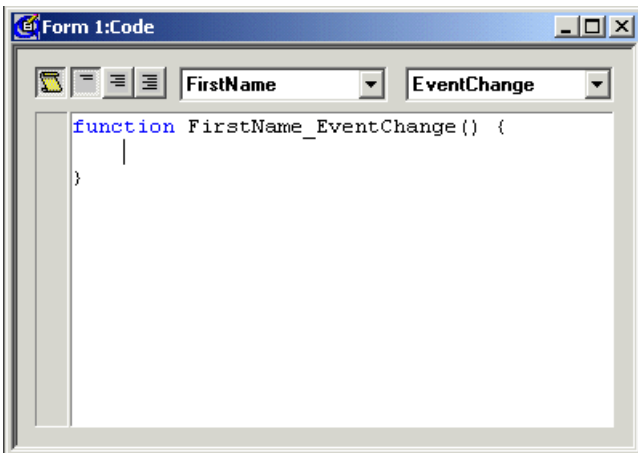
2. Click **OK**.
3. Add controls to your form by clicking and dragging objects from the Object Bar.



4. Edit the object properties as necessary in the Property Sheet.



5. Click **Layout > Objects** and set the tab order.
6. Click **View > Script Editor**.
7. Select the object in the left drop-down list and an event in the right drop-down list and type scripts as necessary.



8. Click **File > Save**.

Binding a form to XML content

You can bind controls in your form to elements and attributes by specifying an XPath property. You can then use the XFT Form Wizard to associate a form with an object in your customization.

XPath syntax is similar to filesystem syntax. In XPath, element nodes are indicated with a slash character (/). XMetaL supports a subset of XPath expressions.

Table 10: Supported XPath expressions

Expression	Refers to
.	Anchor element
./@attributename	Attribute of the anchor element

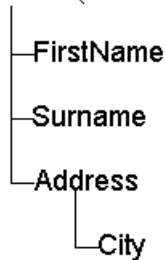
Expression	Refers to
<code>./childelementname</code>	Child element
<code>./childelementname/@attributename</code>	Attribute of child element

Example

Here, you will bind the control for the First Name edit box in a form to the `FirstName` element. The `Author` element is the anchor element for the form.

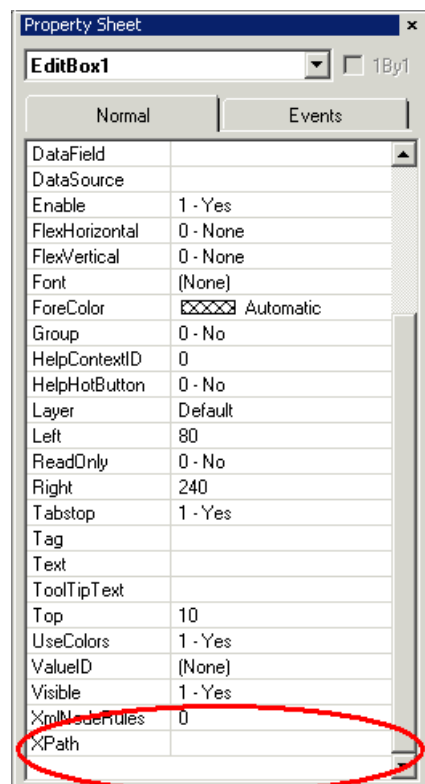
Consider the following document structure:

Author (ID attribute)



The document specifies an `Author` element with child elements called `FirstName`, `Surname`, and `Address`.

To indicate that the First Name control is bound to the element `FirstName`, the XPath expression `./FirstName` is entered in the XPath property.





Special considerations

If the underlying document contains any element nodes that do not exist in the base XML document, XFT displays the form with the element and allows the content to be entered. However, your document is not populated with the data.

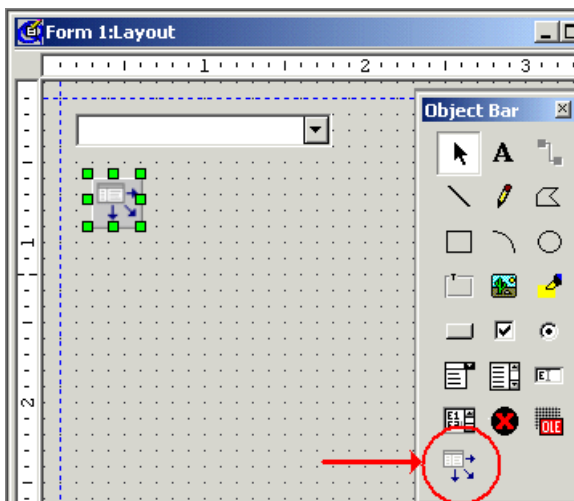
For controls that have an XPath property defined, the underlying XML document requires that the element resolved by the XPath property already be created before XFT sets that element's content as node-type `NODE_TEXT`. The sample form `Author.xft` demonstrates how to create elements before XFT transfers the form control data to the XML document.

XFT supports only DOM node-type `NODE_TEXT`. If other DOM node-types are required, use the `OnXftPutFormToDoc` script hook to create your data. If the DOM node resolved by an XPath property has child nodes or other node-types in its content, XFT replaces that content with `NODE_TEXT` data from the form control or `XmlValue` property. To create this functionality, use the `Tag` property. The `Tag` property can be assigned to all object types, such as DOM nodes, text strings, and numbers. You can assign the `Tag` property to a result and later retrieve the result from the macro that invoked the XFT dialog box. Any type of data can be assigned to the `Tag` property and retrieved by script elsewhere, such as in an `XMetaL` macro.

External data

You can exchange data with an external data source through your form. The Data Source object must be added to any form that uses external data sources.

You can add a Data Source object from the Object Bar.



This form control object is visible during design time, but invisible at run time.

External data can be connected to the following objects:

- Text boxes
- Edit boxes

- List boxes
- Combo boxes

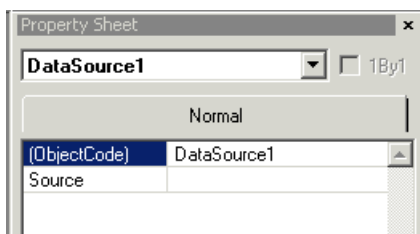


Note: In order to have a way to move from row to row in the data source, your list must include at least one drop down or list box.

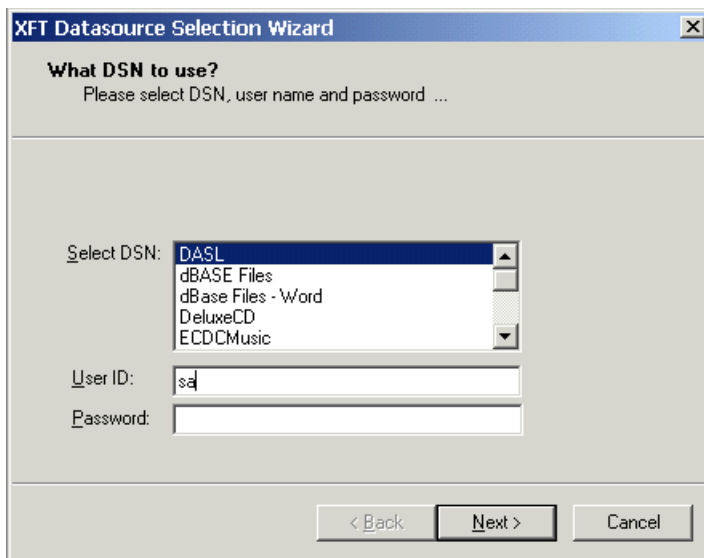
Connect to an external data source

You can connect a Data Source object in your form to an external data source through the data source selection wizard. You then set the Data Field and Data Source properties for the object.

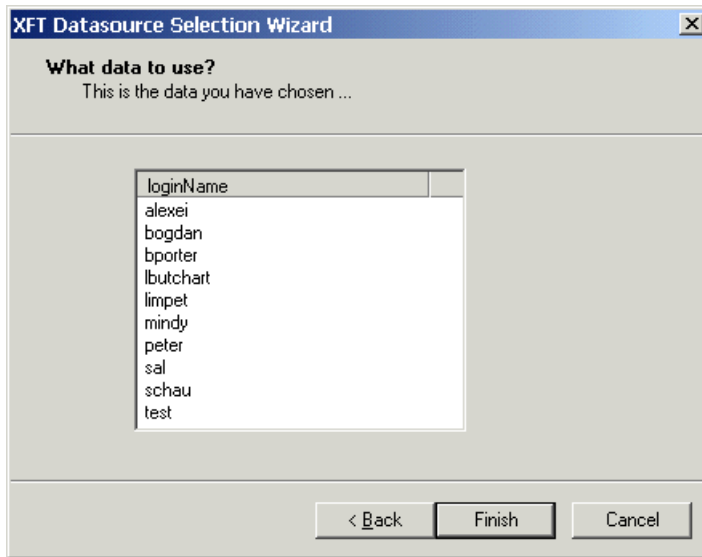
1. Drag the Data Source object onto your form.
2. In the Data Source property sheet, click in the text area to the right of the Source property to start the data source selection wizard.



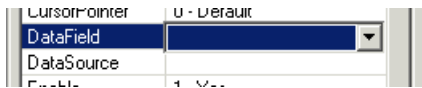
3. In the **Select DSN** list, locate and click the data source.
4. Type the **User ID** and **Password** for the data source if these are required.



5. Click **Next**.
The data for the field you selected is displayed on this page of the wizard.



6. Click **Finish**.
7. Click the control for which you want to select a data source field.
8. Set the **DataField** and **DataSource** properties.



Associate a form with a customization object

You can associate a form with an object in your customization using the XFT Form Wizard.

1. In the Solution Explorer, select an object in your customization that you want to associate with a form.
2. In the Advanced Display Type property, select **XFT Form**.
3. Click **Setup**.
4. Type the path and name of the form.
5. Select an option to determine how XMetaL will run the form.
6. (Optional) If you chose to run the form as an Embedded form, select an option to determine how XMetaL will display the form.

In most cases, you will want to select Replace Content; however, there may be times when you want to preserve content that is related to the data entered on the form. In these cases, select Before Content or After Content.

7. Select an option to indicate when XMetaL is to display the form.

Executing a form as a modal dialog in XMetaL

You can run a form as a modal dialog in XMetaL by using a macro.

The macro shown here uses the `Application.CreateFormDlg` method to call the form `myform.xft`:

```
<MACRO name="RunForm" lang="JScript">
var dlg=Application.CreateFormDlg("C:\\myform.xft");
dlg.DoModal();
dlg=null;
</MACRO>
```

Sample forms

Sample forms are located in `..\\XMetaL\\Author\\Forms`.

Table 11: Sample forms

Name	Description
Annotation.xft	This form is called by the Insert Annotation macro. It is an interface for editors and reviewers to provide initialed annotations to a document that is based on the Journalist DTD. It contains no special scripts and is not bound to any content.
Author.xft	This is an embedded form for the Journalist DTD. It is an interface to enter or change author information in a document. It contains object events script, and is bound to XML content.
Comment.xft	This form is called by the Insert Comment macro. It is an interface for editors and reviewers to provide comments to a document that is based on the Journalist DTD. It contains no special scripts and is not bound to any content.
ListOfComments.xft	This form is called by the ListAllComments macro. It is an interface to view comments added to a document that is based on the Journalist DTD. It contains no special scripts and is not bound to any content.
PubDate.xft	This form was set up as a modal dialog box. It is an interface to enter or change the publication date of a document based on the Journalist DTD. It contains object events script for the <code>OnXftPutXmlValue</code> and <code>OnXftGetXmlValue</code> functions, and is bound to content in the document.
ULink.xft	This form is called by the Insert ULink macro. It is used to create a URL link from text selected in a document based on the Journalist DTD. It contains no special scripts and is not bound to any content.
XMLtoPDFSetup.xft	This form is called by the XMLtoPDF macro. It is used to initialize and configure the PDF print engine. It contains no special scripts and is not bound to any content.

Editor display styles

You can control how XML and SGML documents appear in XMetaL Author and XMetaL XMAX using cascading style sheets (CSS). When you create a customization, two CSS files are created. Style rules from these files is compiled in the XAC file when you build your customization.

Table 12: Default CSS files

Name	UseAs type	Description
{dtdname}.css	CSS styles file for Normal view	The default display styles
{dtdname}_structure.css	CSS styles file for Structure view	Overrides or adds to the default styles

The default files contain selectors and style properties for each element in your DTD or schema. You can modify the style rules in the CSS editor or in a text editor. You can also add existing CSS files to your customization to use instead of the default files. However, if you choose to do this, you must specify the UseAs type accordingly.

For details on using XSLT to transform documents for previewing, see the section on Formatting Object methods in the *XMetaL Programmer's Guide* for details.

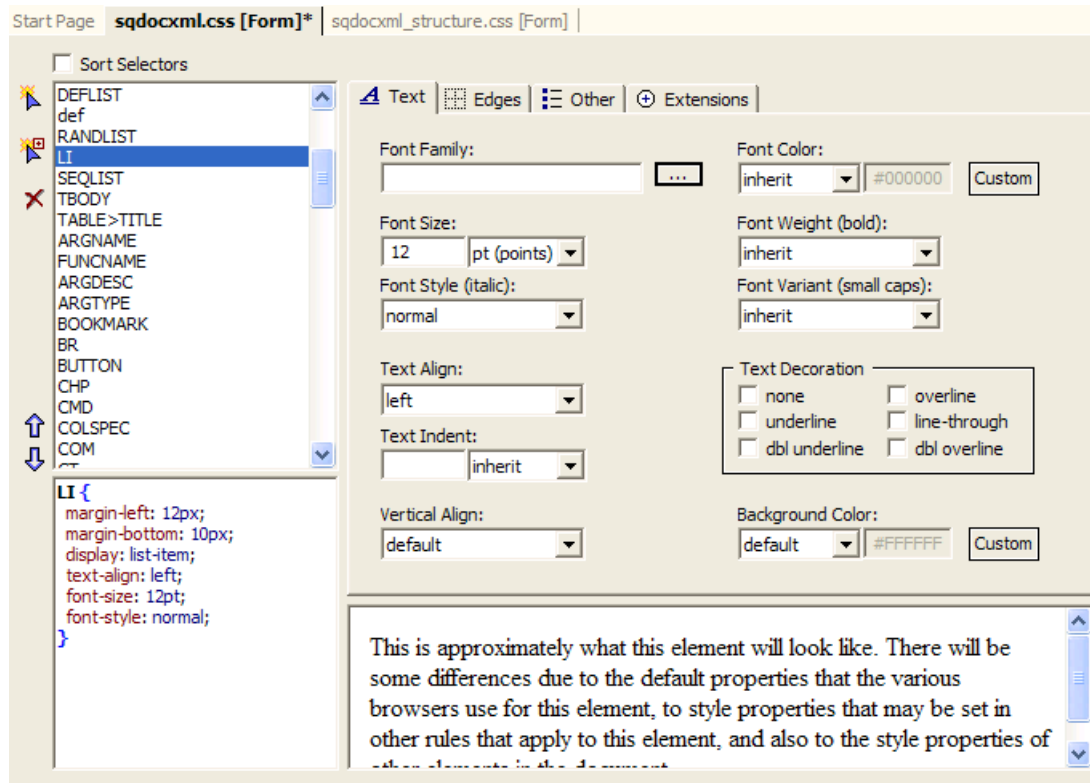
CSS specifications are available on the World Wide Web Consortium website at www.w3c.org.

CSS editor

You can edit CSS files through the CSS editor.

The cascading style sheet editor contains the following areas:

- **Selectors area.** Lets you create, delete, sort, and choose selectors.
- **Style Rule area.** Displays the style rule for the current selector and lets you edit it.
- **Sample Text area.** Formats the text according to the style rule.
- **Properties area.** Lets you create style rules.



You can create a new selector by clicking **Create new CSS selector** or **Create new CSS selector with properties of selected Selector**.

You can remove a selector by clicking **Delete selector from the list**

You can change the order of the selectors by clicking **Move up** or **Move down**.

Creating selectors

A selector is an expression with which you can associate a style property. You can create selectors through the Selectors area of the CSS editor.

A selector can represent the following, or any combination thereof:

- Elements
- Attributes
- Pseudo-elements and pseudo-classes
- XMetaL-specific keywords

By default, selectors appear in the list according to the order in which they were created, with the XMetaL-specific selectors first, followed by a list of the elements defined in the DTD or schema, and finally by any selectors that you create. However, you can change this order when you create or edit selectors by clicking the up and down arrows.

Selector syntax

For a description of selector syntax, see the CSS specifications on the World Wide Web Consortium website at www.w3c.org.

Simple selectors

The following example causes all Title elements to appear in blue:

```
TITLE {color:blue;}
```

Pseudo selectors

The following rule causes all first-child elements of a section to appear in boldface:

```
SEC:first-child {font-weight:bold;}
```

Child selectors

The following rule applies to all title elements that have book as a parent:

```
BOOK>TITLE {color:blue;}
```

Attribute selectors

The following rule causes all emphasis elements with the role bold to appear in blue:

```
EM[role="bold"] {color:blue;}
```

Sibling selectors

The following rule specifies the color blue for all paragraph elements:

```
P {color:blue;}
```

The following rule, causes all paragraphs immediately preceded by a paragraph to appear in red:

```
P+P {color:red;}
```

The result of these rules would be that the first paragraph appears in blue and all subsequent paragraphs appear red.

Class selectors

The following rule causes all titles with the class news to appear in red:

```
TITLE.news {color:red;}
```

Descendant selectors

The following rule causes all titles that are child elements of a book to appear in blue:

```
BOOK TITLE {color: blue;}
```

XMetaL-specific selectors

In a new CSS file, XMetaL Developer automatically populates the list with XMetaL-specific selectors, which let you take advantage of specific XMetaL functionality. In the list of available selectors, these appear first unless you change the order.

Table 13: XMetaL-specific selectors

Name	Description
\$DOCUMENT	Applies styles to an entire document. All other selectors inherit the styles set for this selector unless different styles are set specifically for subordinate selectors.
\$MARKSEC	Applies styles to any marked sections in SGML documents or CDATA sections in XML documents.
\$COMMENT	Applies styles to comments within an XML or SMGL document.
\$PROCINS	Applies styles to processing instructions within an XML or SGML document.



Note: Visual Studio .NET returns an unexpected character sequence error for each of the proprietary XMetaL-specific keywords. Also, if you build your project with the View Code window open, errors these keywords are listed in the task list but the file is copied into the build.

Styling processing instructions

You can style processing instructions using the \$PROCINS keyword and the following qualifier names:

```
xm-pi-target
```

```
xm-pi-data
```

```
xm-pi-target
```

Normal CSS cascading rules apply.

Examples

The following rule hides all occurrences of any PI with a target called 'print':

```
$procins[xm-pi-target="print"] {display:none}
```

The following rule hides all occurrences of ?print page-break?:

```
$procins[xm-pi-target="print"][xm-pi-data="page-break"] {display:none}
```

The following rule colors all PIs with a data value of 'index-start' using blue text:



```
$procins[xm-pi-data="index-start"] {color:#0000ff}
```


The following rule styles all replacement text PIs in green:

```
$PROCINS[xm-pi-target="xm-replace_text"] {color: green;}
```

Create a selector

You can create a blank selector or copy style rules from an existing selector.

1. Open a CSS file through the Solution Explorer.
2. Do one of the following:
 - Click  **Create new CSS selector**.
 - In the Selectors list, click the name of the selector whose style rules you want to copy. Click  **Create new CSS selector with properties of selected Selector**.
3. Choose a type of selector.
4. From the Select Element list, select the name of the element to use as the selector and click **>**.
5. If you want to create a single style rule to apply to all selectors in the Selectors to be created list, click the **Add as a grouped selector** option.

 **Note:** New selectors must have style rules associated with them before the cascading style sheet editor is closed or they will not be saved in the Selectors list.

Custom selectors

The Custom option in the CSS editor lets you type a custom selector. Style rules that you create with custom selectors may not be visible in XMetaL Author or XMetaL XMAX.

Some possible uses for custom selectors include the following:

- To specify a namespace so that similarly-named elements from different DTDs or schemas can have different formatting information
- To add an '@import' statement to import rules from another style sheet into the current style sheet

 **Note:** Styles in an imported style sheet have lower precedence than styles in the current style sheet.

Namespace-aware selector

The following rule causes all emphasis elements in namespace my_ns to be styled in boldface:

```
my_ns|em {font-weight: bold;}
```

@import statement

The following statement imports the rules from the CSS file mystyles.css:

```
@import url("mystyles.css")
```

Setting style properties

You can edit the properties of your style rules through the CSS editor. The Properties area lets you select options to set properties, or you can type properties directly in the Style Rule area. By default, many properties inherit their value from parent and ancestor elements.

The Properties Area of the CSS editor contains four tabs that show the properties that can be displayed in XMetaL. You can set these properties by clicking the tab and moving to the appropriate section.

Table 14: Properties

Click this tab:	To edit:
Text	Fonts, text alignment, and indent
Edges	Margin, padding, and borders
Other	Miscellaneous properties, including classification, whitespace, and counters
Extensions	XMetaL-specific properties, including indent, prefix, and view settings

You can also add other properties by editing the text in the Style Rule area. However, these properties may not be visible in XMetaL.

For descriptions of CSS properties see the World Wide Web Consortium website at www.w3c.org.

Extensions properties

The Extensions tab in the CSS editor contains XMetaL-specific properties. The properties on this tab are not part of the CSS specifications, and can be viewed only in XMetaL.

Left indent

Use this property to display the element(s) indicated at a set distance from the left margin, regardless of the position of any ancestor element.

Prefix options

Use this property to displays prefix text, the attributes of an element, or the parameters and parameter entities of a marked section (in Tags On view only). You can set the following prefix options:

- To set a rule to display prefix text, type the desired text
- To set a rule to display all attributes with non-null values for the element(s) to which the style applies, type `[%attribute-list;]`
- To set a rule to display the value of a specific attribute for the element(s) to which the style applies, type `[%attribute ATTRIBNAME;]`, where ATTRIBNAME is the name of the attribute whose value you want to display
- If you are creating a display rule for marked sections, you can choose to display marked-section parameters and parameter entities by typing `[%parameters;]`

Structure view options

Use these properties to control the behavior of the Structure View. You can set the following options:

- **Show +/-** to display the expand and collapse icons
- **Show icons** to display page and book icons
- **Show start tags** to display a start tag at the beginning of each element

View support for properties and selectors

The tables below indicate XMetaL Developer and XMetaL Author display support for CSS properties and selectors.

Table 15: Assigning property values, cascading, inheritance

Name	XMetaL Developer CSS Editor Preview	XMetaL Author Normal View	XMetaL Author Structure View
'@import' rule	X	X	X
'inherit' value	-	X	X

Table 16: Font properties

Name	XMetaL Developer CSS Editor Preview	XMetaL Author Normal View	XMetaL Author Structure View
font	X	X	X
font-family	X	X	X
font-style	X	X	X
font-variant	X	X	X
font-weight	X	X	X
font-size	X	X	X

Table 17: Color and background properties

Name	XMetaL Developer CSS Editor Preview	XMetaL Author Normal View	XMetaL Author Structure View
background	X	X	X
background-attachment	-	-	-
background-color	X	X	X
background-image	-	-	-
background-position	-	-	-
background-repeat	-	-	-

Table 18: Text properties

Name	XMetaL Developer CSS Editor Preview	XMetaL Author Normal View	XMetaL Author Structure View
color	X	X	X
text-indent	X	X	X
text-align	X	X	-
text-decoration	X (except double-overline and double-underline)	X (except double-overline and double-underline)	X (except double-overline and double-underline)
letter-spacing	-	-	-
word-spacing	-	-	-
text-transform	-	-	-
vertical-align	-	sub and super only	sub and super only
line-height	-	-	-

Table 19: Box properties

Name	XMetaL Developer CSS Editor Preview	XMetaL Author Normal View	XMetaL Author Structure View
border, border-top, border-right, border-bottom, border-left	X	X	X
border-color	X	X (only when set to the same value for all sides)	X (only when set to the same value for all sides)
border-top-color, border-right-color, border-bottom-color, border-left-color	X	X (only when set to the same value for all sides)	X (only when set to the same value for all sides)
border-style	X	X	X
border-top-style, border-right-style, border-bottom-style, border-left-style	X	X	X
border-width	X	X (only when border-xxx-width are set to the same value)	X (only when border-xxx-width are set to the same value)
border-top-width, border-right-width, border-bottom-width, border-left-width	X	X (only when set to the same value)	X (only when set to the same value)
clear	-	-	-
float	X	X	-
height	-	-	-
width	-	-	-
margin	X	X	-
margin-top, margin-right, margin-bottom, margin-left	X	X	-
padding	X	X	-
padding-top, padding-right, padding-bottom, padding-left	X	X	-

Table 20: Classification properties

Name	XMetaL Developer CSS Editor Preview	XMetaL Author Normal View	XMetaL Author Structure View
white-space	X (for 'nowrap' only)	X	-
display	-	X	-
list-style	-	X	-
list-style-type	-	X	-
list-style-position	-	-	-
list-style-image	-	-	-

Table 21: Generated content, auto-numbering, and list properties

Name	XMetaL Developer CSS Editor Preview	XMetaL Author Normal View	XMetaL Author Structure View
content: <string>	-	X	X
content: <uri>	-	-	-
content: <counter>	X	X	X
content: attr(X)	X	X	X
counter-increment	X	X	X
counter-reset	X	X	X

Table 22: Positioning properties

Name	XMetaL Developer CSS Editor Preview	XMetaL Author Normal View	XMetaL Author Structure View
position	-	-	-
top	-	-	-
right	-	-	-
bottom	-	-	-
left	-	-	-
width	-	-	-
height	-	-	-
clip	-	-	-
overflow	-	-	-
z-index	-	-	-
visibility	-	-	-

Table 23: Other properties

Name	XMetaL Developer CSS Editor Preview	XMetaL Author Normal View	XMetaL Author Structure View
@media	-	-	-
direction	X	X	-
unicode-bidi	X	X	-
min-width	-	-	-
max-width	-	-	-
@page	-	-	-
page-break-before, page-break-after	-	-	-
page-break-inside	-	-	-
orphans	-	-	-
widows	-	-	-
border-collapse	-	-	-

Name	XMetaL Developer CSS Editor Preview	XMetaL Author Normal View	XMetaL Author Structure View
border-spacing	-	-	-
caption-side	-	-	-
table-layout	-	-	-
empty-cells	-	-	-
cursor	-	-	-
outline	-	-	-
outline-width	-	-	-
outline-style	-	-	-
outline-color	-	-	-

Table 24: Selectors

Type	XMetaL Developer CSS Editor Preview	XMetaL Author Normal View	XMetaL Author Structure View
Grouping	X	X	X
Universal Selectors	X	X	X
Type Selectors	X	X	X
Descendant Selectors	X	X	X
Child Selectors	X	X	X
Adjacent Sibling Selectors	X	X	X
Attribute Selectors	X	X	X
Class Selectors	X	X	X
ID Selectors	-	-	-
Pseudo-Classes	:first-child only	:first-child only	:first-child only
Pseudo-Elements	:before and :after only	:before and :after only	:before and :after only

Using counters and autonumbering

A counter is an element prefix that is incremented for each successive occurrence of that element. You can create counters in your document using pseudo-selectors and counter properties.

You can add simple or multi-level counters to elements in your document. For example, chapters in a document may be numbered 1, 2, 3, etc. and subsections of a chapter may be numbered 1.1, 1.2, ... , 2.1, 2.2, ..., etc. A counter can be initialized to start at a specified value and various numbering styles are available.

Counters are displayed in `:before` and `:after` generated text by using the `counter` and `counters` functions. They are configured using the `counter-increment` and `counter-reset` properties.

Simple counter

The following rule associates a counter with the `Sect1` element and gives this counter the name 'section1', so that it can be referred to elsewhere in the style sheet.

```
Sect1 {
  counter-increment: section1;
}
```

Displaying a counter

The following rule displays the counter at the start of the `Sect1` element, but it is more common to display the counter before an element's title (if there is one):

This rule displays the counter at the start of the `Sect1` element, but it is more common to display the counter before an element's title (if there is one):

```
Sect1>Title:before {
  content: counter(section1);
}
```

You can include text before or after the counter. For example, the following rule displays 1., 2., ... before the titles:

```
Sect1>Title:before {
  content: counter(section1) ". ";
}
```

Resetting the numbering

The following rules reset the number for each new list:

```
ItemizedList {
  counter-reset: li;
}
ListItem {
  counter-increment: li;
}
ListItem:before {
  content: counter(li) ". ";
}
```

Initializing counters

The following rule resets the counter to 5:

```
Sect1 {
  counter-increment: section1;
  counter-reset: section2 5;
}
```

Multi-level counters

The following rules define two counters, section1 and section2. The `counter-reset` property in the `Sect1` rule means reset the counter called section2 to zero whenever a `Sect1` element is encountered.

```
Sect1 {
  counter-increment: section1;
  counter-reset: section2;
}
Sect1>Title:before {
  content: counter(section1) ". ";
}
Sect2 {
  counter-increment: section2;
}
Sect2>Title:before {
  content: counter(section1) "." counter(section2) ". ";
}
```

Numbering styles

The following rules specify upper-alpha and lower-roman style counters:

```
Sect1>Title:before {
  content: counter(section1, upper-alpha) ". ";
}
Sect2>Title:before {
  content: counter(section1, upper-alpha) "."
  counter(section2, lower-roman) ". ";
}
```

Formatting elements as tables

Using CSS, you can format other groups of elements as tables, provided they have a tabular structure. These elements are sometimes referred to as *semantic tables*.

A tabular structure must have these components:

- An enclosing element, which can be formatted as a table
- A child of the table element, which can be formatted as a table row
- A child of the table row element, which can be formatted as a table cell

Example

The `VariableList` element in the sample journalist DTD can be formatted as a table. This element has the following structure:

```
<VariableList>
<VarListEntry>
<Term>...</Term>
<ListItem>...</ListItem>
</VarListEntry>
<VarListEntry>
...
</VarListEntry>
...
</VariableList>
```

Through the CSS display property (in the Other tab), you can format `VariableList` as a table, `VarListEntry` as a table row, and `Term` and `ListItem` as table cells. The following rules are used to format `VariableList` as a table:

```
VariableList {
display: table;
}
VarListEntry {
display: table-row;
}
VarListEntry>Term {
display: table-cell;
}
VarListEntry>ListItem {
display: table-cell;
}
```

(In this case we specify styles for `Term` and `ListItem` when they occur as sub-elements of `VarListEntry`, since the DTD allows these elements to occur in other contexts.)

Elements styled using this method will appear as tables but they will not support standard table editing functions such as those provided through the Table Properties dialog. This type of functionality requires scripting via macros.

Example style rules

The following sample style sheet contains style rules that were created with the CSS editor:

```
DOCTITLE { font-size: 20pt; line-height: 22pt; color: green }
.student { display: none }
#para1 { font-style: italic; font-weight: bold }
TITLE3, TITLE4 { font-size: 14pt; line-height: 16pt }
QUOTE P { text-indent: 0.5in }
LI.student { display: none }
SEC{prefix-format: "[ %attribute NAME; ]";}
```

The first rule states that all `DOCTITLE` elements have a font size of 20 points, line height of 22 points, and are displayed in green.

The second rule states that all elements in the class **student** (that is, all elements, of any type, whose `CLASS` attribute has the value 'student') are hidden.

The third rule states that the element with ID value **para1** appears in a bold, italic font.

The fourth rule states that **both** `TITLE3` and `TITLE4` elements have a font size of 14 points and a line height of 16 points.

The fifth rule states that a `P` element that is contained in a `QUOTE` element is indented by 0.5 inches.

The sixth rule states that an `LI` element in the class "student" (that is, its `CLASS` attribute has the value 'student') are hidden.

The seventh rule states that, for all `SEC` elements, the value of the `NAME` attribute is displayed before the element content.

Resource Manager

In XMetaL Author, the Resource Manager lets you view and organize resources so that you can easily use them in your documents. By default, the Resource Manager contains the Assets tab (Asset Manager) and the Desktop tab. You can configure the Resource Manager using the `ResourceManager` interface and through configuration files and scripts.

Assets tab (unsupported)

The Asset Manager gives you access to objects such as images and text files. In addition to adding assets to existing asset types, you can define new types and write scripts that define how files are added to them and the behavior that occurs when an asset is dropped into a document. For more information about adding assets, see the *XMetaL User's Guide*.



Note: The Assets tab is unsupported and may be removed in a future release.

Desktop tab

The upper pane of the Desktop tab displays a Windows Explorer-type view of the files and folders on your system. The lower pane shows folder contents. You can drag and drop files from the lower pane into your document.

The Desktop tab supports Windows shell namespace extensions.

When customizing the Desktop tab using scripts, you need to be aware of the following limitations:

- You cannot grab the control with a script
- `IDispatch` is not supported by any of the Microsoft interfaces used by the Desktop tab
- Replacing it with a new tab (using the `AddTab` method) and creating a new instance of Windows Explorer does not create a single instance (opening a folder creates a new instance of Explorer)

Custom tabs

You can use script to add custom tabs (for example, that contain ActiveX controls) to the Resource Manager using the `ResourceManager` APIs.

Configuring the Asset Manager

You can change the way assets are displayed in the Asset Manager and the behavior that occurs when they are dropped into a document.

The tools consist of:

- Configuration files.
- Properties and methods as described in the *XMetaL Programmer's Guide*. For example, you can invoke a dialog when users drop an asset into the Asset Manager or into a document.
- Script variables. These strings correspond to files, element contents, and attribute values used by the Asset Manager. They are resolved by the Asset Manager before scripts are executed. Most of them apply to the `Item` element for the asset currently being dragged.

In order to configure the Asset Manager, you need a working knowledge of a scripting language (JScript, JavaScript, or VBScript), XML, and HTML. Familiarity with the HTML Document Object Model (DOM) as implemented by Internet Explorer is also helpful.

Table 25: Configuration files

Name	Description
fxitems.xml	Asset catalog file. This file contains an entry (<code>Item</code> element) for each asset. It can also include templates that define the structure of each entry, and scripts that define the action that is to take place when an asset is dropped into a document.
fxindex.htm	Asset display file. This file determines the contents of the lower pane of the Assets tab when the user opens a folder, including text and images. Each folder in the Asset Manager must contain an asset display file.
fxmaster.xml	Master asset catalog file, by default <code>.. \Program Files\XMetaL\Author\Assets\fxmaster.xml</code> . If an asset catalog file does not contain the template and script that determine how an asset is represented and how it is dropped into a document, the Resource Manager looks here for this information.
noscrollbars.txt	An empty text file. When this file exists in an asset folder, scroll bars are not displayed in the lower pane of the Asset Manager.

Table 26: Methods and properties

Name	Description
<code>Application.Alert</code> , <code>Application.Confirm</code>	User interaction methods.
<code>Application.AssetsPath</code>	The path to the assets.
<code>Application.CopyAssetFile</code>	Copies a file from any location (including one specified by a URL) into an asset folder or other folder.
<code>Application.DropPoint</code>	Returns a <code>Range</code> object corresponding to the point at which the user drops the script into the document.
<code>Selection.Collapse</code>	Collapses the selection to an insertion point at its start- or end-point.
<code>Selection.ContainerAttribute</code>	Sets or returns the value of a specified attribute of the selection's container.
<code>Selection.ContainerName</code>	Sets or returns the name of the current container (element, processing instruction, section, or comment).
<code>Application.FileToString</code>	Enables you to read an external text file (plain text or tagged) and assign its contents to a variable. It can then be manipulated or pasted.
<code>Selection.InsertElement</code> , <code>Selection.CanInsert</code>	<code>InsertElement</code> inserts the specified element. <code>CanInsert</code> can be used to check if the insertion would be valid.
<code>Selection.InsertImage</code>	Inserts an image using an element defined as an image element.
<code>Selection.MoveToElement</code>	Move to the next instance of a specified element.

Name	Description
<code>Selection.PasteString, Selection.CanPaste</code>	<code>PasteString</code> does a basic paste of a string into the current document. <code>CanPaste</code> can be used to check if the intended paste is valid.
<code>Selection.SelectContainerContents</code>	Selects the contents of the current container.

Table 27: Script variables

Name	Description
<code>%Pathname%</code>	The path part of the current page's URL; that is, everything up to the rightmost forward slash.
<code>%Filename%</code>	The value of the <code>SRC</code> attribute of the <code>Filename</code> child element of the <code>Item</code> element of the asset being dragged. In other words, the asset filename.
<code>%TextData%</code>	The content of the <code>TextData</code> child element of the <code>Item</code> element of the asset being dragged.
<code>%thisItem.SQ_getChildAttribute(element, index, attribute)%</code>	The value of the attribute <code>attribute</code> of the child element <code>element</code> of the <code>Item</code> element corresponding to the asset being dragged.
<code>%thisItem.SQ_getChildInnerHTML(element, index)%</code>	The contents of the subelement <code>element</code> of the <code>Item</code> element corresponding to the asset being dragged.
<code>%htmlItem.SQ_getAttribute(id, index, attribute)%</code>	The value of the attribute <code>attribute</code> of the element in <code>fxitems.xml</code> that has the ID attribute equal to <code>id</code> .

Creating asset types

New asset types are defined in asset templates, which consist of an asset display file and an asset catalog file. When you define a new asset type, you enable users to create assets of this type.

To add an asset type, you create a folder in `.. \Program Files\XMetaL\Author\Asset Templates`. When a user creates a new asset folder and specifies the new asset type for that folder, the files in the template folder are copied to the new asset folder. The scripts in the asset catalog file define the following:

- The action to take when users drag files into the asset folder. This can include prompts and dialogs. The script adds an `Item` element for the new asset into the catalog.
- The action to take when a user drags a file from the asset folder into a document.

As you are developing and testing asset catalog and asset display files, you can refresh the view in the Asset Manager through the right-click menu.


Sample asset catalog and asset display files are located in `.. \Program Files\XMetaL\Author\Samples\Asset Templates`.

Asset display file

Every asset folder contains a display file that reads the entries contained in the catalog file and displays them in the lower pane of the Asset Manager. Users can easily drag and drop assets from the lower pane into a document.

Sample asset display files are located at `..\Program Files\XMetaL\Author\Samples\Asset Templates`. The sample display files have the following characteristics:

- An `OBJECT` element that refers to an ActiveX control that reads an asset catalog file and creates a data structure.
- A JScript that uses this control to read the asset catalog file and create a data structure.
- A script that traverses the data structure and obtains information about an asset from the corresponding `Item` element. This information can be customized, but it usually consists of an asset filename, an icon filename, a description, and a unique identifier. This information is displayed in the lower pane.

 **Note:** Because the asset display file is processed by a browser component, you must use a standard scripting language such as JScript.

To give users information about the asset type, you may want to include text and graphics. If a folder does not contain any assets, you still need to create an asset display file in it. These can include text or graphics that explain what kind of assets are in the sub-folders of the current folder. For assets that may be difficult for some users to understand, you can include a short documentation section or links to other documents.

Asset catalog file

The asset catalog file provides a template for the structure of the catalog entries (`Item` elements) and scripts that define the behavior that occurs when a user adds an asset to a folder or drags it into a document.

Sample asset catalog files are located in `..\Program Files\XMetaL\Author\Samples\Asset Templates`. In addition to the required elements, your template can contain any other elements. The templates in the sample files also contain `Description` and `SCRIPT`.

Here is a sample template:

```
<Item Type="File" Class="Fig" ID="%Filename%" FILEFILTER="*.jpg;*.gif;*.png">
  <Icon SRC="%Filename%"/>
  <Description>%ASK:Description:%{Description}<Description>
  <Filename SRC="%Filename%"/>
  <SCRIPT Language="JScript">
    ...
  <SCRIPT>
</Item>
```

Table 28: Required template elements

Name	Description
Item	The root element. The <code>Type</code> attribute can have the value 'Text' or 'File'. The <code>Class</code> attribute identifies the asset type. <code>ID</code> must be a unique identifier within the set of assets in a particular folder. The <code>FILEFILTER</code> attribute is used if an assets folder is created from Windows Explorer instead of through the Asset manager. When this folder is selected in the Asset tab, you see a Convert to Assets page. The file extensions specified determine which types of files are added to the assets when you do the conversion.
Icon	The icon used to represent the asset in the lower pane of the Asset tab. The <code>SRC</code> attribute specifies the icon file. This file must be in a format that can be displayed by a Web browser. If you do not want to use the actual asset filename, you can create a dialog for the user to enter a filename.
Filename	The asset filename. The <code>SRC</code> attribute specifies the file.

Description

Description can be used to generate a text box prompt in the Asset Details dialog box that appears when the user adds an asset to the folder. A string of the form `%ASK:String1:%String2` is interpreted as follows:

- `%ASK` causes a text box to be displayed.
- `String1` is for the text box label. This should be a short string without spaces.
- `String2` specifies the default content of the text box.

When the `Item` element is written to the asset catalog file, the user's response is substituted for the `%ASK` string. `%ASK` can also be used to prompt the user for an attribute value.

SCRIPT

The `SCRIPT` element contains code that inserts the asset into the document when a user drags it from the Asset tab. If the asset has the type 'File', the script also copies the asset file from the asset folder to a location relative to the document in which it is being inserted.

This script carries out the following steps:

1. Get the full path to the asset file.
2. Create the attribute value that specifies the filename.
3. Define a function, `buildpath`, that calculates the location where the asset file is to be copied.
4. Define a function, `dropfxnow`, that copies the file and performs the insertion.
5. Check whether the document has ever been saved. If it has, call `buildpath` and `dropfxnow`. If it has not, prompt the user to save the document.
6. Check a second time whether the document has been saved. If it has, call `buildpath` and `dropfxnow`.

The sample template can be used to write the following `Item` element:

```
<Item Type="File" Class="Fig" ID="scully06.jpg" FILEFILTER="*.jpg;*.gif;*.png">
  <Icon SRC="scully06.jpg"/>
  <Description>Scully saves Mulder again!</Description>
  <Filename SRC="scully06.jpg"/>
</Item>
```

Text file and text block assets

Assets can also consist of text files or blocks of text. The contents of the text file or the `TextData` element are inserted into the document when a user drags and drops the asset from the Asset Manager.

For these types of assets, you must either specify an icon in the `Item` template, or prompt the user to enter the name of an image file when the asset is created.

The contents of text block assets are stored in the asset catalog file itself, rather than in another file. Text block assets have the following characteristics:

- The `Type` attribute must have the value 'Text'.
- Instead of a `Filename` element, the `Item` template has a `TextData` element that contains character data. The script that drops the asset into a document must obtain the contents of the `TextData` element.

Master asset catalog file

If the catalog file in an asset folder does not contain the template and script that determine how an asset is represented and how it is dropped into a document, the Resource Manager looks in the master asset catalog file for this information.

Storing templates and scripts in a central location is a convenient way of managing your assets. If you choose this approach, you need to create a shell asset catalog file containing the following elements in the asset folder:

```
<?xml version="1.0" standalone="yes"?>
<Items><Items>
```

If the template and script for an asset type are contained in the master catalog file, then the user is prompted to choose the asset type from among all the available asset types when a new asset is created. If the catalog file in the asset folder contains the template and script, the asset type is assumed to be the one specified by the template.

Remote assets

You can store asset folders on a Web server or Intranet, but you cannot add new assets to remote folders.

Remote asset catalog files specify URLs to the asset catalog files in each asset folder. Here is an example:

```
<?xml version="1.0" standalone="yes"?>
<FOLDERS MODIFIED="19990511.1">
  <FOLDER NAME="Buttons" URL="http://www.mysite.com/xmassets/buttons/fxindex.htm" />
  <FOLDER NAME="BigButtons"
URL="http://www.mysite.com/xmassets/buttons/bbuttons/fxindex.htm" />
  <FOLDER NAME="LittleButtons"
URL="http://www.mysite.com/xmassets/buttons/lbuttons/fxindex.htm" />
  . . .
</FOLDERS>
```

Remote asset catalog files have the following characteristics:

- **FOLDER** elements indicate folders. You can create a nested folder structure.
- The **MODIFIED** attribute indicates the last revision of the asset folder structure, in YYYYMMDD.R format, where R is a revision number. You must update this value whenever the structure of the file changes. If the **MODIFIED** value changes in the top-level remote asset catalog file, the local structure is updated the next time the user accesses remote assets.
- The **NAME** is the folder name, the **URL** is the full path to the folder's asset display file.

When you access remote assets, a folder structure is built underneath the folder that you designate as the remote folder on your local system. Its structure parallels that of the assets on the server, but the only content in each local folder is an Internet shortcut file that points to the remote server. Internet shortcut files have the extension '.url'. Here is an example:

```
[InternetShortcut]
URL=http://www.mysite.com/xmassets/fxindex.htm
```

The value of **URL** must be the location of the asset display file in the remote assets folder on the server.

Set up a remote assets folder

You first create assets on your local system and then move them to a remote location on a server.

1. Create a folder for local assets in `.. \Program Files\XMetaL\Author\Assets`.
Give the folder an easily identifiable name, such as 'Local'.
2. In the Local folder, create an asset catalog file that contains entries (`Item` elements) for each asset.
3. Create a folder for remote assets in `.. \Program Files\XMetaL\Author\Assets`.

Give the folder an easily identifiable name, such as 'Remote'.

4. In the Remote folder, use a text editor to create an Internet shortcut file, and save the file as `assets.url`.
5. Move the asset folders from their local location to the server location.
6. In the Remote folder, create an asset catalog file.

Configuring XMetaL

Each installation of XMetaL Author contains a global configuration file and a user configuration file. You can configure XMetaL Author behavior for your customization through the global configuration file.

Configuration files contain one or more variables that are read when you start XMetaL. Each variable is a name-value pair. The values can be booleans, pathnames, numbers, and strings. Changes to configuration files are effective when you re-start XMetaL.

Table 29: XMetaL configuration files

Name	Description
..\Program Files (x86)\XMetaL12.0\Author\XMetaL.ini	Global configuration file. Contains default factory settings and settings made for deployment within a customization. This file should not be modified after XMetaL Author has been installed, except by the installer in Modify or Repair mode. This file is reset to its factory settings if XMetaL Author is updated or re-installed.
..\Users\{username}\AppData\Roaming\SoftQuad\XMetaL\12.0\XMetaL.ini	User configuration file. Contains user-specified settings and settings made through the XMetaL interface, such as Options. These settings take precedence over global settings.

If your customization of XMetaL uses the XMetaL Application Customization (XAC) set, these files can be stored on a web server, a network server, or any user-accessible folder on the local system.

Adding new toolbar icons

You can add icons to the list of those already provided for creating custom toolbar buttons.

The images must conform to the following specifications:

- Standard Windows 16 color palette
- Width 18 pixels, height 16 pixels
- BMP format

The files containing the images must be BMP files containing a row of up to 10 images in the format described above. Each icon set can have up to eight rows. The files must be named `row1.bmp`, `row2.bmp`, ... `row8.bmp`. The files for a single icon set must be in their own sub-folder of the `Icons` folder.

In order to tell XMetaL Author to use a custom icon set, you must edit the `Icons\icons.ini` file. For example, if you have two custom icon sets in the folders `Icons\Flags` and `Icons\Tools`, you need to add two entries to the `icons.ini` file, as follows:

```
[icon groups]
1=Flags
2=Tools
```

If you change the number associated with an icon group after some of its members have been assigned to toolbar buttons, those buttons may then have a different icon, or no icon. In this situation, you must re-assign images to the buttons.

Frequently used configuration variables

The following is a list of frequently used configuration variables.

Table 30: Frequently used configuration variables

To do this:	Change this variable:
Disable well-formed editing	enable_edit_as_wellformed
Create a log file	make_log_file
Disable default validation on save	validate_before_export
Display entity replacement text in Tags On view	expand_entities_in_tags_on_view
Turn off rules checking option	rules_checking_always_off_option_shown
Turn off rules checking	rules_checking_always_off
Set macro scripting language	default_macro_language
Set macro folder location	macro_path
Allow modification of toolbars and menus	enable_toolbar_customization
Set the default width of the Structure View	default_structure_view_width
Change top-level folder in Assets tab	assets_path
Set toolbar filepath	tbr_path
Change backup extension	backup_ext
Change default plain-text font	default_font_name
Change recognized entity extension	entity_ext
Make a log file during debug	make_log_file
Set the name of the log file	log_file
Turn off validation	rules_checking_always_off
Set user name	user_name

Configuration variables

The following variables are recognized by XMetaL Author at startup.

Table 31: INI Variables

Variable name	Default value	Description
about_rtf_file	\${SQDIR}/XMetaL.rtf	File contents to be displayed in the About box.
action_on_space_typed_in_element_content	insert_directly	-
always_undo_clear_after_save	YES	Clears the undo stack whenever a document is saved.
applet_ext	.class	-
assets_path	\${SQDIR}\Assets	-

Variable name	Default value	Description
auto_backup_instead_of_auto_save	False	Instead of automatically saving the document to the same file, automatically create a backup file (with the extension specified in the backup_ext INI variable). When this value is True, XMetaL creates a backup file without performing validation.
backup_ext	.bak	-
CALS_table_auto_layout	NO	-
color_entity_refs_in_text	NO	-
colormap_segment_size	16	-
context_area_width	120	Initial size of context area.
ctm_path	\${SQDIR}\Rules	Path of .ctm (customization) files. Defaults to rules_path, which defaults to "\${SQDIR}\Rules" where \${SQDIR} is the software's install folder path.
cursor_file0	\${SQDIR}/Cursors/Cursor0.cur	-
cursor_file1	\${SQDIR}/Cursors/Cursor1.cur	-
cursor_file2	\${SQDIR}/Cursors/Cursor2.cur	-
cursor_file3	\${SQDIR}/Cursors/Cursor3.cur	-
cursor_file4	\${SQDIR}/Cursors/Cursor4.cur	-
cursor_file5	\${SQDIR}/Cursors/Cursor5.cur	-
cursor_file6	\${SQDIR}/Cursors/Cursor6.cur	-
cursor_file7	\${SQDIR}/Cursors/Cursor7.cur	-
cursor_file8	\${SQDIR}/Cursors/Cursor8.cur	-
cursor_file9	\${SQDIR}/Cursors/Cursor9.cur	-
custom_macro_ext	.mcr	Extension for macro files
custom_macro_path	\${SQDIR}\Startup	Macros to be loaded when XMetaL starts.
default_font_name	Courier New	Font used in Plain Text view.
default_font_size	12	Size of the font used in Plain Text view.
default_macro_language	VBScript	Determine default language when a new macro is created.
default_save_as_ext	.xml	Default Save As extension.
default_structure_view_width	-	Default width of Structure View.
default_template	-	Default template to use when user creates a new document.
display_place_marker_for_float	YES	-
document_path	C:\Users\{username}\Documents	-
draw_grid_on_borderless_tables	true	Draw a dotted line around table cells when the border value is set to 0.

Variable name	Default value	Description
empty_element_image	empty.gif	Image displayed for an empty element in Tags On and Normal views.
enable_advanced_debugging	NO	-
enable_edit_as_wellformed	YES	Allow a user to edit a document as well formed.
enable_toolbar_customization	YES	When set to NO, disables the Customize option in the toolbars context menu (right-click menu).
entity_ext	.ent	Entity file extension.
entity_font_name	Arial	Font used for entity icons in Tags On view.
entity_font_size	8	Size of the font used for entity icons in Tags On view.
entity_icon_background_color	white	Background colour used for entity icons in Tags On view.
entity_icon_color	slate grey	Colour used for entity icons in Tags On view.
entity_path	\${SQDIR}\Rules\entities	Path of entity files.
evaluation_version	NO	-
expand_entities_in_tags_on_view_too	NO	Entity replacement text in Tags On view.
export_doc_type_dec	YES	When set to NO the doctype declaration is omitted from the saved file. Files that contain a doctype declaration when opened have the doctype declaration stripped from them.
export_eol	NO	The control character hex 0A (line feed) is appended to the end of lines so that each line contains 0A 0D at the end. Without this variable set to NO the end of a line is marked only with 0D (carriage return).
export_sgml_dec	NO	-
extid_map	\${SQDIR}/extid.map	Path to the extid.map file.
find_backward	NO	Set the Backwards Search checkbox in the Find and Replace dialog.
find_case_sensitive	NO	Set the Match Case checkbox in the Find and Replace dialog.
find_patterns	NO	Set the Use Pattern Matching checkbox in the Find and Replace dialog.
find_whole_words	NO	Set the Whole Words checkbox in the Find and Replace dialog.
find_wrap	YES	Set the Wrap checkbox in the Find and Replace dialog.
fix_image_urls_on_export	YES	-

Variable name	Default value	Description
fix_sysid_url_on_export	YES	true (default): XMetaL Author changes the SYSTEM URL (path to the DTD) in the Doctype Declaration to a path relative to the location of the XML file. false: The path is left unaltered (i.e., the value it was when the XML file was first loaded)
fix_transparent_color	YES	-
fix_URLs_on_remotesave	YES	-
format_tags	NO	-
fx_chooser_root_page	\${SQDIR}/hmf/hmf.htm	-
fx_chooser_viewer	-	-
help_context_file	\${SQDIR}\xmetal.chm	-
help_file	\${SQDIR}\xmetal.chm	-
help_home_page	http://www.xmetal.com	-
help_on_help_file	\${SQDIR}\viewhlp.chm	-
highlight_mode	Default	-
html_browser	-	-
html_browser_path	-	-
html_browser0	-	External browser (1st on Preview toolbar)
html_browser1	-	External browser (2nd on Preview toolbar)
html_browser2	-	External browser (3rd on Preview toolbar)
html_browser3	-	External browser (4th on Preview toolbar)
html_browser4	-	External browser
html_browser5	-	External browser
html_browser6	-	External browser
html_browser7	-	External browser
html_browser8	-	External browser
html_browser9	-	External browser
html_file_extensions	htm;shtm;html;shtml;mv	-
icons_in_menus	YES	Display icons in the menus on the Menu bar.
image_ext	.gif	-
image_exts	gif;jpg;png;bmp;tif	Image file types listed in the Insert Image dialog box.
image_path	C:\Users\{username}\Documents	Defaults to document_path but at appit time, is set to last place an image was chosen from.
image_viewer	-	-

Variable name	Default value	Description
image_viewer_path	-	-
img_ext	BMP;GIF;EPS;PCX;TIF;WMF;WPG;SDW;CGM;TGA;JPG;PNG;PIC	-
import_ext	.htm	-
import_ext	.sgm	-
import_path	-	Defaults to document_path but at appexit time, it is set to the last place from where an image was chosen.
include_required_elements	YES	-
internet_session_agent	-	-
internet_session_cache	NO	-
internet_session_dialog_delay	1000	-
internet_session_http_request_header	Accept: */*	-
internet_session_proxy	-	-
internet_session_proxy_access	0	-
internet_session_proxy_bypass	-	-
ISMAP_ext	.map	-
keep_sv_mv_in_sync_by_default	NO	Make element collapsing and expanding which you do in either the Structure View or the Main View, get automatically done in the other view.
log_file	\${SQDIR}\xm_lookup.log	Path and filename of the log file XMetaL will create if the 'make_log_file' variable is set to YES.
macro_ext	.mcr	-
macro_file	-	-
macro_path	\${SQDIR}\Macros	Path to macros folder.
make_backup_file	NO	Create a second copy of every file saved using the extension set in the 'backup_ext' variable.
make_catalog_dump_file	NO	-
make_log_file	NO	If set to YES, XMetaL creates a log file of all files it opens and searches when opening or creating a new document.
max_changes_between_saves	65000	When this number of changes to a document is made, XMetaL automatically saves the file.
max_time_between_saves	1000	When this number of minutes passes, XMetaL automatically saves the file.
minimize_tag_icons	false	Can show tags without element names inside them.
net_img_ext	GIF;JPG;PNG	-
OASIS_override	YES	See topic Giving priority to system or public identifiers.

Variable name	Default value	Description
ole_server_busy_timeout	8000	-
open_as_wellformed	NO	-
organization	-	XMetaL - Central. Do not change.
print_left_footer	-	-
print_left_header	-	-
print_middle_footer	-	-
print_middle_header	-	-
print_right_footer	-	-
print_right_header	-	-
product_identifier	1234-5678-9246	-
prompt_for_attrs	NO	-
publish_change_from	-	-
publish_change_to	-	-
remote_edit_temp_dir	full path	Uses 8.3 paths.
restore_last_open_docs	YES	-
rules_checking_always_off	NO	When set to YES, XMetaL never validates a document.
rules_checking_always_off_option_shown	-	When set to true, turns rules checking option off.
rules_ext	.rls	-
rules_file	XMetaL.rlx	-
rules_path	\${SQDIR}\Rules	The directory XMetaL should look in to find rules files (.rld, .rlx, .rls), .dtd files and schema files (.xsd).
save_path	C:\Users\{username}\Documents	The directory that is shown when saving a document for the first time. Defaults to document_path but at app-exit time, it is set to the last place where a document was saved.
show_comments	YES	Show comments in Tags On view.
show_fixed_attrs_in_tag_tips	YES	-
show_head_element	YES	Hide the HEAD element in HTML.
show_ignored_marked_sections	YES	-
show_inline_images	NO	-
show_rules_check_off_dialog	NO	-
show_structure_view_by_default	YES	Show the Structure View when a document is opened.
show_tag_tips	YES	-
show_urls	NO	-

Variable name	Default value	Description
source_color_anchor_tag	008000	Not used.
source_color_background	FFFFFF	-
source_color_cdata	000080	-
source_color_comment	800080	-
source_color_decl	FF0080	-
source_color_end_tag	000080	-
source_color_entity_ref	808080	-
source_color_foreground	000000	Default text color in Plain Text view.
source_color_quote	FF0000	Color of attribute values.
source_color_script	808080	-
source_color_script_comment	008000	-
source_color_script_keyword	0000FF	-
source_color_script_quote	408080	-
source_color_start_tag	0000FF	-
source_color_sub_decl	000080	-
source_color_table_tag	8000FF	-
source_view_auto_indent	YES	-
source_view_color_html	YES	Enable syntax coloring on markup (Plain Text view).
source_view_color_script	YES	Enable syntax coloring on scripts (Plain Text view).
source_view_dont_wrap_in_tags	NO	-
source_view_expand_tabs_on_save	NO	-
source_view_keywords_path	\${SQDIR}\Keywords.ini	-
source_view_line_numbering	YES	Display line numbering in source view.
source_view_tab_size	3	Width of a tab character (Plain Text view).
source_view_use_tabs	YES	Display tabs in source view.
source_view_wrap	0	Wrap lines in source view. 0 = Off; 1 = Break within words; 2 = Break between words.
SQCONFIG	C:\WINDOWS\xmetal.ini;C:\Program Files (x86)\XMetaL 12.0\Author\xmetal.ini	Path to the XMetaL INI files.
SQDIR	C:\Program Files (x86)\XMetaL 12.0\Author OR C:\Program Files (x86)\XMetaL 12.0\XMAX	Path to the XMetaL executable xmetal.exe. This value is used by other INI variables in the form \${SQDIR}.
styles_ext	.css	-
styles_path	\${SQDIR}\Display	-
tag_font_name	Arial	Typeface used for tag icons.
tag_font_size	8	Font size used for tag icons.

Variable name	Default value	Description
tag_icon_background_color	white	Background color for tag icons.
tag_icon_color	slate grey	Text and outline color for tag icons.
tags_on_graphical_tables	NO	-
tbr_path	\${SQDIR}\Rules	Defaults to rules_path
templates_path	\${SQDIR}\Template	Default location of templates.
toolbar_icon_path	\${SQDIR}\Icons	Default location of icons when toolbars are created.
unavailable_image	GIF/noimg.gif	-
undo_limit	1000	Maximum number of undo actions.
unique_attribute_value_max_tries	1000	Number of times XMetaL will try to create a unique attribute value. This affects some API methods.
unlock_version	NO	-
unsupported_image	GIF/badform.gif	-
url_file_ext	Web Documents (* .htm*, *.mv, *.shtm) *.htm;*.mv;*.shtm HTML Documents (* .htm, *.html) *.htm;*.html Miva Documents (*.mv) *.mv Server Includes (*.shtm, *.shtml) *.shtm;*.shtml All Files (*.*) *.*	-
urls_default_to_relative	YES	-
use_extid_mapping	YES	-
use_inline_IME	NO	-
use_open_market	NO	-
user_initials	first initial of username	User's first initial extracted from the Windows logon name.
user_name	same value as the user's Windows logon name	Name used for change tracking. If no name is specified, the default Windows user name is used.
validate_before_export	YES	Validate a file before any action that causes the file to be saved.
view_for_open	2	Startup View. Possible values are: 0 (Plain Text), 1 (Tags On), 2 (Normal). Note that these values are different from those used in the 'ViewType' API property.
warn_before_saving	NO	-
wysiwyg_printer_font_name	Arial	-
wysiwyg_printer_font_size	10	-
wysiwyg_nodes_to_hide	-	-
xml_export_blank_line_after_end_tags_0001	-	-
xml_export_blank_line_after_start_tags_0001	-	-

Variable name	Default value	Description
xml_export_blank_line_before_end_tags_0001	-	-
xml_export_blank_line_before_start_tags_0001	-	-
xml_export_indent_spaces	2	-
xml_export_indent_tags_0001	-	-
xml_export_new_line_after_end_tags_0001	-	-
xml_export_new_line_after_start_tags_0001	-	-
xml_export_new_line_before_end_tags_0001	-	-
xml_export_new_line_before_start_tags_0001	-	-
xml_export_paragraph_child_elms_0001	-	-
xml_file_extensions	xml;ux;ent	-

Glossary

ambiguous content model	A content model in a DTD is ambiguous if an element in a document instance could match more than one part of the content model.
application customization	A customization at the application level. Application-level customizations apply to all documents, regardless of the DTD or schema used.
attribute	A value that is associated with an element but is not part of the content of the element. Many properties are represented by attributes, for example, class or ID.
block element	An element whose content is preceded and followed by line breaks.
browser	A program that communicates with Web servers and is used for retrieving and displaying documents from the World Wide Web or an intranet. Most browsers use a graphical interface to provide access to text, images, audio, and video.
CALS table model	A widely used DTD for table markup, described in the U.S. Department of Defense SGML standard MIL-M-28001B. XMetaL Author supports a definition of the CALS DTD developed by the OASIS consortium and described at www.oasis-open.org .
cascading style sheet (CSS)	A way to specify document formatting that is supported by browsers. XMetaL uses cascading style sheets to format the document pane in Normal and Tags On views. A cascading style sheet generally consists of one or more rules that define element appearance. These style sheets are said to be cascading because several style sheets can be applied to the same document. See www.w3.org for more information.
catalog files	One or more files that map external identifiers for DTDs, rules files, or entities to a filename. Also called OASIS catalog files. For more information on catalog files, see OASIS Technical Resolution 9401:1997 .
CDATA	Character data. A type of content in which any XML or SGML markup delimiters (such as '<' and '&') that appear are treated as ordinary characters. XML and SGML documents can contain CDATA sections; SGML documents can contain CDATA elements.
CDATA section	A markup construct in XML and SGML documents, beginning with the characters '<! [CDATA[' and ending with ']>', inside which all content is treated as character data.
COM interface	Component Object Model. A language-independent interface developed by Microsoft for combining applications under Microsoft Windows. The XMetaL scripting API uses a COM interface.
content model	An expression in a DTD that defines the content of a particular element.
counter	A numerical element prefix that is incremented automatically for each successive occurrence of that element. For example, chapters in a document may be numbered 1, 2, 3, ..., etc. Counters are implemented via a cascading style sheet.
customization	An enhancement to the functionality, behavior, or appearance of XMetaL Author. Customizations can be made at the document level or at the application level.
customization file (CTM)	An XML configuration file specifying a variety of element-based behaviors and properties for an XML document type.
current element	The element containing the insertion point or selection. If an entire element is selected, the current element is the parent of that element, not the selected element itself.

DOCTYPE declaration	Document type declaration. A declaration at the top of an XML or SGML document that specifies which DTD applies to the document, and may contain some extra markup declarations.
document customization	A customization that applies to all documents based on the DTD or schema used for the customization.
Document Object Model (DOM)	The Document Object Model (DOM) is an abstract definition of an API (application program interface) for manipulating XML document structures. The DOM is a Recommendation of the World Wide Web Consortium (W3C) , developed and maintained by the W3C DOM Working group. XMetaL follows the DOM Level 1 Specification . The DOM was designed to represent XML structures, but can represent SGML structures (such as CALS tables) if they are also found in XML.
DTD	Document Type Declaration. A set of declarations, written in a formal notation, that defines the structure of a document.
element	The building blocks of XML and SGML documents. Elements are named according to their function in the document, for example, headings, lists, and paragraphs.
empty element	An element that cannot have any content.
entity	A special character, a block of text or markup, or a file.
entity reference	A reference, using a specific syntax, to an entity. When the document is displayed in a browser or editor, the entity reference is replaced by the text or file that the entity represents.
external entity	A type of entity that represents another XML or SGML file.
external identifier	A way of identifying an external file. External identifiers can appear in a document type declaration and in external entity declarations, where they identify the external file that the entity refers to. In SGML files, external identifiers can consist of a system identifier, a public identifier, or both. In XML files, external identifiers must contain a system identifier, which may be preceded by a public identifier.
external identifier map file	The file, <code>../XMetaL/Author/extid.map</code> , that associates external identifiers with filenames on the system.
followed-by element	An element that is inserted following the occurrence of a specific element. Followed-by elements are configured in XMetaL Developer as part of the customization (<code>.ctm</code>) file.
form	A data-entry interface usually associated with specific elements in an XMetaL document. Forms are designed in the XMetaL Forms Toolkit (XFT) either as dialog boxes that are launched from an XMetaL macro, or as content that appears as part of an element.
general entity	These can be text entities, which represent a piece of text or a single character; external entities, which represent another XML file; and graphic entities, which represent a graphic, audio, or video, etc. file.
generated text	Text that is not part of the document content, but is generated by a display program and displayed at the beginning or end of an element's content.
graphic entity	A type of general entity that represents an external multimedia file, for example, a graphic, video, or audio file.
hypertext	Text that can be used to link to another document or another location in the same document. The viewer can display the linked document or location by clicking the text.
ID	A unique identifier. The value of an ID attribute must not be used for any other ID attribute in the document.

IDispatch object	An object associated with an in-place control that gives a script access to the control's properties, methods, and events.
IDREF	A reference to an ID. Unlike ID attributes, IDREF attributes do not have to be unique: more than one IDREF can refer to the same ID.
inline element	An element that does not have a line break before or after its contents.
in-place control	An ActiveX control that is embedded in the Normal and Tags On document panes in XMetaL Author or XMetaL XMAX, and communicates with XMetaL Author or XMetaL XMAX so that changes in the control can modify the document, and vice versa.
internal subset	An optional part of the document type declaration that may contain markup declarations.
ISO	International Organization for Standardization.
ISO 8859-1 character set	The character set for special or accented characters that is widely used in HTML documents. This character set is also called ISO Latin 1. It includes characters required for most western European languages.
macro	A sequence of XMetaL actions or script commands that can be run as a unit via a keyboard shortcut, a toolbar button, or a menu item. Macros can be recorded from within XMetaL Author, or created by inserting scripting code into a macro file using XMetaL Developer.
marked section	A markup construct in SGML documents that designates the content for special processing. The parameters of the marked section specify the type of processing. The most common uses for marked sections are to cause a portion of the document to be ignored at certain times, and to surround SGML markup constructs that you want to be treated as text, not markup.
marked section parameter	Keywords that determine how to process a marked section in an SGML document. The available keywords are INCLUDE (process the section normally); IGNORE (do not process the section); CDATA (treat elements and entity references in the content as text, not markup); RCDATA (treat elements in the content as text, not markup); and TEMP (the section is temporary).
markup	Special instructions or indicators in a document that specify how the enclosed content is to be processed by an application. Element tags are an example of markup. In an XML or SGML document, the element tags specify the role of the content (heading, title, paragraph, etc.).
MCR file	A macro file. An MCR file is an XML-based customization file containing XMetaL macros, or scripts. For document-level customizations, the MCR filename is named according to the DTD or schema; for application-level customizations, the MCR filename is <code>xmetal.mcr</code> .
modal dialog box	A dialog box that remains active until it is closed. Once it is open, you must complete your task and close it before you can return to the main window and continue working.
modeless dialog box	A dialog box that can remain open while you are working in the main program window.
namespace	A feature of XML that permits documents to contain identically-named elements defined in more than one DTD or schema.
notation	A declaration in a DTD that specifies a file format that can be used for files represented by graphic entities. For example, if GIF files are to be used, a notation declaring the GIF format must be present in the DTD.
OASIS	Organization for the Advancement of Structured Information Standards, a consortium dedicated to promoting structured information standards such as XML and SGML. For more information, see www.oasis-open.org .

parameter entity	An entity that represents one or more marked section parameter keywords.
PCDATA	Parsed character data. The most common form of text content in XML and SGML documents. In PCDATA text, any markup, such as element start and end tags and entity references, is interpreted with its normal meanings.
pretty-printing	Saving a file that contains markup so that it is easily readable, for example, by indenting lists to reflect a nested structure.
PRE-like elements	Elements that are formatted to look like the HTML <code>PRE</code> element; that is, with all whitespace preserved exactly as it was entered.
processing instruction	An instruction that is not interpreted as part of the document's content, but rather interpreted by an application that is processing the file.
public identifier	A system-independent string that is used to represent a DTD or entity file. Part of an external identifier.
RCDATA	Replaceable character data. SGML files can have RCDATA elements and marked sections, in which any element start- or end-tags that occur are interpreted as text, but any entity references are interpreted in the normal way.
required attribute	An attribute that must be present in order for the document to be valid.
required element	An element that must be present in order for the document to be valid.
remote file	A file on an http or Web server.
rules checking	An XMetaL feature that ensures that you do not break the required structure as you edit your document; it does this by allowing you to insert only valid elements. Rules checking is less stringent than validation in that it checks that no errors have been made, but does not check that the markup is complete.
rules file	An XMetaL-specific alternative to a DTD. All of the files comprising a DTD are compiled into a single binary rules file. Rules files can be issued to XMetaL users who are not authorized to modify the DTD.
schema	An XML standard for defining the structure, content, and semantics of an XML document.
SDATA entity	A type of text entity whose content is specific to a particular processing application or platform. These entities are often used to represent platform-specific characters, and codes for formatting systems (such as troff or TEX). SDATA entities are permitted only in SGML files.
selector	In a cascading style sheet, a selector is an expression, representing one or more elements, that a style property can be associated with. A selector can represent an element, several elements, an element with a specific ancestor, an element in a particular class, etc.
semantic tables	A group of elements that is not marked up with one of the supported table models (CALs and HTML) that can be formatted as a table.
SGML	Standard General Markup Language. A standard for describing the structure of a document using markup. SGML is described by the ISO 8879 standard (1986). HTML and XML are applications of SGML.
SGML declaration	An SGML declaration is a file associated with a DTD that contains information about the character set, markup delimiters, quantity settings, and special markup features that are available in documents that use that DTD.
SQDIR	The variable that is used to represent the folder in which XMetaL is installed.

standalone document	An XML document that an application can parse without referring to an external DTD. A standalone document may still require a DTD in some situations, for example, if it is being edited.
system identifier	Part of an external identifier. A system identifier is generally the filename of the file (for example, a DTD or entity set) that the external identifier refers to. In XML documents, system identifiers are required in external identifiers, and are interpreted as URLs.
tags	An element in an XML or SGML file begins with a start-tag (for example, <PRE>) and ends with an end tag (for example, </PRE>).
TBR	Toolbar file. Toolbar and menu customizations are saved in TBR files. When XMetaL Author loads a TBR file, it looks for the file in the user's personal settings folder first. If it does not find the file, it then looks for the file in the folder specified by the <code>tbr_path</code> variable in the XMetaL configuration file. By default, this variable is set to the <code>rules_path</code> value. If it does not find the TBR file in either of these places, it looks for it in the folder that contains the DTD or schema, if this is a folder other than the <code>Rules</code> folder.
text entity	A type of general entity that stands for one or more text characters.
Unicode	A standard for electronically encoding the characters of many of the scripts used to write the world's languages, as well as special symbols such as mathematical symbols. Unicode is the character encoding specified by XML. For more information, see www.unicode.org .
URL	Uniform Resource Locator. A URL is the address of a file, written in a format that can be interpreted by a Web server.
valid document	An XML or SGML document is valid if it is well-formed and if it conforms to the rules in the DTD or XML schema.
VBScript	Visual Basic Script. A scripting language implemented by Microsoft Visual Basic. VBScript is one of the languages that can be used to configure XMetaL with the COM interface .
W3C	World Wide Web Consortium, an industry association for the development of World Wide Web technologies. For more information, see www.w3.org .
WebDAV	World Wide Web Distributed Authoring and Versioning, the Internet Engineering Task Force standard for collaborative authoring on the Web. WebDAV is a set of extensions to the Hypertext Transfer Protocol (HTTP) and Secure Hypertext Transfer Protocol (HTTPS) that facilitates collaborative editing and file management between users located remotely from each other on the Internet.
well-formed document	An XML document that is structurally correct according to the XML standard. There are several aspects to well-formedness, the most important of which are: the document must have only one top-level element, and all elements must be properly nested.
whitespace	One or more space, tab, carriage return, or line feed characters, in any combination.
XAC	XMetaL Application Customization file. A compiled, deployable customization file. It contains all the files created or copied during the build process, and is recognized by XMetaL Author and XMetaL XMAX as a customization.
XFT	XMetaL Forms Toolkit. A set of form creation and form layout tools that developers can use to design and implement embedded forms and modal dialog boxes.
XML	Extensible Markup Language. An easy-to-implement subset of SGML, originally designed for displaying content over the Internet. XML is an initiative of the W3C. For more information, see www.w3.org/XML .

XML declaration	A processing instruction that appears at the start of an XML document. This processing instruction indicates the XML version being used, and may specify the character encoding and whether the document needs an external DTD.
XSLT	Extensible Stylesheet Language: Transformations. A language for describing how to transform an XML document into another XML document.

Index

.rls/.rlx/.rld files 38
@import statement 64
#GLOBAL element properties 16

A

adding
 buttons 8
 customization items 10
 existing macros 33
 existing, functions 33
 items 10
 menu items 8
 new functions 31
 new macros 31
ambiguous content model 38
arguments
 command 18
asset catalog file 73
Asset catalog file 76
asset display file 73
Asset display file 75
Asset Manager 73
 Asset catalog file 76
 Asset display file 75
 configuration files 73
 creating new asset types 75
 fxindex.htm file 75
 fxitems.xml file 76
 fxmaster.xml file 77
 master catalog file 77
 methods 73
 remote assets 78
 script variables 73
 text and text file assets 77
associating
 XFT forms, with elements 12
attribute declarations (schema support) 37
attribute description file 37, 49
attribute group definition (schema support) 37
attributes
 case 47
 description file 37, 49
 modifying 38
 selectors 61
AttributeUses (schema support) 37
autonumbering 69

B

backup_ext
 xmetal50.ini 81
building
 configuring the build environment 18
 customization 18
bulleted lists 15

buttons
 adding 8

C

cascading style sheet editor 60
cascading style sheets 60
 :after pseudo-element 69
 :before pseudo-element 69
 browser support 60
 counters 69
 examples 72
 extensions 65
 imported style sheets 60
 introduction 60
 left indent property 65
 location 60
 miscellaneous properties 65
 prefix options 65
 priority 61
 rule ordering 61
 standards 60
 Structure View options 65
 style properties 64
 supported properties 66
 table formatting 71
case, names 47
catalog
 OASIS 41, 42
catalog file entries
 resolving 42
CATALOG keyword 44
catalog, OASIS 45
 CATALOG keyword 44
 DELEGATE keyword 44
 finding 44
 keyword precedence 41
 keyword summary 41
 locations 44
 OVERRIDE keyword 44
catalogs
 support in schemas 43
change list 12
CLASS attribute 72
command arguments 18
commands
 menu 8
compiling a DTD 38
complex type definitions (schema support) 37
configuration file 80
configuring
 structure view CSS 60
configuring the build environment 18
content types 38
converting
 doc to xml 35

- counters
 - counter-increment 69
 - counter-reset 69
 - initializing 69
 - multi-level 69
 - styles 69
 - creating
 - DTD, overview 6
 - forms 51, 53
 - rules 8
 - schema, overview 8
 - solution 7
 - XML template 8
 - XSD, overview 8
 - CSS
 - custom selectors 64
 - debugging 19
 - structure view 60
 - supported properties 66
 - table formatting 71
 - validation 60
 - CTM
 - debugging 19
 - Custom tabs (in Resource Manager) 73
 - customization
 - adding an item to 10
 - building 18
 - debugging 19
 - modifying 10
 - new 7
 - opening items in 10
 - removing an item from 10
 - saving 10
 - customization components 8
 - customization file element properties 11
 - customization support 22
 - Customizations
 - advanced 51
 - customizing
 - overview 6
 - with xft 51
 - customizing XMetaL
 - adding new toolbar icons 80
 - alias property 11
 - change list 12
 - customizing XMetaL
 - designating elements as paragraphs 15
 - designating toggling elements 15
 - description property 11
 - element properties 11
 - elements
 - assigning to buttons 15
 - toggling 15
 - followed-by element 13
 - images
 - defining in Customizations dialog box 15
 - image elements 15
 - introduction 6
 - macros
 - creating 15
 - paragraphs
 - defining in Customizations dialog box 15
 - customizing XMetaL (*continued*)
 - text layout 15
 - toggling elements 15
 - toolbars
 - customizing 15
- ## D
- data source
 - form 56
 - object 56
 - data source object
 - XFT 56
 - database import
 - required components 34
 - debugging
 - CSS and CTM 19
 - declaration subset 40
 - default content 11
 - default font (plain text)
 - xmetal50.ini 81
 - definition lists 15
 - DELEGATE keyword 44
 - dialog boxes
 - C++ 22
 - editor 51
 - disable text layout
 - in global element properties 16
 - display properties
 - supported CSS selectors and elements 66
 - DLLs 22
 - DOCTYPE
 - internal subset 40
 - document type declaration 39
 - document type definition
 - viewing 37
 - document type name 39
 - DTD
 - creating 37
 - creating, overview 8
 - language support 47
 - modifying 37
 - viewer 37
 - dtd hierarchy
 - mapping 54
 - DTDs 37
 - compiling 38
 - creating 37
 - internal 40
- ## E
- editing
 - functions 32
 - macros 31, 32
 - editing properties
 - functions 31
 - macros 31
 - element declarations (schema support) 37
 - element properties 11
 - elements
 - case 47

- elements (*continued*)
 - modifying 38
 - selectors 61
- encodings 10
- Enter key 13
- entities
 - case 47
- events 50
- examples
 - XFT forms 59
- Excel spreadsheets
 - importing 34
- external identifier map 45, 46
- extid.map file 45, 46

F

- filename
 - for XAC 18
 - output 18
 - XAC file 18
- files
 - DTDs 37
 - rules files 37
 - SGML declaration 47
- filetypes (UseAs) 9
- followed-by elements 13
- form editor 51
- Form Layout Editor 51
- forms 8
 - content mapping model 50
 - creating 51, 53
 - design tips 51
 - filename extension 50
 - forms
 - interfaces 50
 - introduction 50
 - samples 59
 - script events 50
 - testing 50
- forms editor
 - dialog box editor 51
- functions
 - adding existing 33
 - editing 32
 - editing properties 31
 - importing 33
 - new, adding 31
 - testing 34
- fxindex.htm (asset display file) 73
- fxitems.xml (asset catalog file) 73
- fxmaster.xml (master asset catalog file) 73

G

- generated text
 - counters 69
- getting started 6
- Global element
 - properties 16
- Global element properties 16
- global settings 80

H

- html 35
 - preview 35
 - print 35
- html printing
 - customize 35

I

- icons, toolbar 80
- icons.ini 80
- identifier
 - external 39
 - public 39
 - system 39
- importing
 - databases 34
 - Excel spreadsheets 34
 - functions 33
 - macros 33
 - scripts 33
- In-Parent entries 17
 - customization 17
- ini variables
 - complete list 81
- internal DTD 40
- internal subset 40
- introduction 6

L

- language support 47
- limitations
 - schema 37
 - wildcards 37
- list
 - ini variables, all 81
- list elements 15
- lists
 - defining in Customizations dialog box 15
- log file
 - xmetal50.ini 81

M

- macros
 - editing 31, 32
 - editing properties 31
 - existing, adding 33
 - importing 33
 - language 81
 - new, adding 31
 - testing 34
- mapping
 - dtd hierarchy 54
- master asset catalog file 73
- MDAC 34
- menu items
 - adding 8

- menus
 - working with 8
- meta-inf.xml 18
- mkrls 38
- model group definitions (schema support) 37
- model groups (schema support) 37
- modifying
 - attributes 38
 - customization 10
 - elements 38
- MS Word 35
- multi-language
 - support 47

N

- NAMECASE (SGML declaration) 47
- names 47
- namespace extensions 73
- namespace-aware selectors 64
- namespaces 16
- new
 - customization 7
- next element 13
- noscrollbars.txt 73
- notation declarations (schema support) 37
- numbered lists 15

O

- OASIS catalog 41
- object
 - data source 56
- Object Bar 51
- object events
 - XFT 50
- options
 - Plain Text 15
- output
 - filename, changing 18
- OVERRIDE keyword 44
- overview
 - customizing 6

P

- paragraph order
 - in global element properties 16
- Particles (schema support) 37
- pdf 35
 - preview 35
 - print 35
- personal settings 80
- planning
 - workflow 7
- post-build
 - properties 18
- pretty printing 15
- printing
 - html 35

- processing instructions
 - styling using CSS 63
- project
 - properties 18
- project properties 18
- project wizard 7
- projects
 - types of 7
- properties
 - of a project 18
- pseudo-classes 61
- pseudo-elements 61
- public identifier 39
- PUBLIC keyword 39

Q

- quantities, SGML 47

R

- removing
 - customization items 10
 - items 10
- replacement text 14
- Resource Manager 73
 - namespace extensions 73
- rules checking 81
- rules files 37
 - creating 38
- Rules files 38
- Rules folder 38
- Rules Maker 38

S

- samples
 - XFT forms 59
- saving
 - customization 10
 - encodings 10
 - Plain Text layout options 15
- schema
 - creating 37
 - creating, overview 8
 - language support 47
 - limitations 37
 - modifying 37
 - viewer 37
- schemas
 - catalog support 43
 - limitations 37
 - rixml support 37
 - support for 37
- script
 - editor 31
- scripts
 - importing 33
- selectors
 - attributes 61
 - elements 61

- selectors (*continued*)
 - namespace-aware 64
 - pseudo-classes 61
 - pseudo-elements 21
 - XMetaL-specific 63
- semantic tables 71
- settings
 - global 80
 - personal 80
- SGML declaration 37
- simple type definitions (schema support) 37
- solution
 - creating 7
- solution explorer
 - working with 10
- structure view
 - CSS, configuring 60
- style properties 64
- style sheets 60
- styles
 - supported properties 66
- support
 - languages 47
- support for schemas 37
- system identifier 39
- SYSTEM keyword 39

T

- tables
 - formatting elements as 71
- TBR
 - purpose 8
- template
 - creating 8
- testing
 - functions 34
 - macros 34
- text layout 15
- toolbars
 - enable_toolbar_customization 81
 - working with 8
- transformations 35
 - html 35
 - pdf 35

U

- UseAs types 9
- user name
 - xmetal50.ini 81

V

- validation
 - CSS 60
- validation disabled
 - xmetal50.ini 81
- viewing
 - DTDs or schemas 37

- viewing and saving
 - html 35
- virtual element 17
 - customization 17
- Visual C++ 22
- Visual Studio .NET Solution Explorer 10

W

- W3C schema
 - viewing 37
- well-formed editing
 - disabling 81
- wildcards
 - schema 37
- Windows Scripting Engine 34
- Windows shell namespace extensions 73
- wizard
 - Import Script Wizard 33
 - project 7
- workflow
 - planning 7

X

- XAC
 - purpose 18
- XAC file
 - filename, specifying 18
- XFT 51
 - design tips 51
 - Form Layout Editor 51
 - forms toolkit 51
 - Object Bar 51
 - object events 50
- XFT forms
 - Advanced Display Type property 12
 - associating with elements 12
 - Display As property 12
 - examples 59
 - in-place controls
 - printing 12
 - printing
 - in-place controls 12
 - samples 59
 - Use Bitmap Printing 12
- xm-replace_text processing instruction 14
- XMAX 20
 - customization 20
- XMAX customization
 - debugging 20
- XMetaL Forms Toolkit 51
- XMetaL Rules Maker 38
- XMetaL-specific selectors 63
- xmetal.ini
 - complete list of ini variables 81
 - configuration 81
 - configuration variables 81
 - variables 81
- xmetal50.ini 80
 - assets_path 81
 - backup_ext 81

- xmetal50.ini (*continued*)
 - default font (plain text) 81
 - default_structure_view_width 81
 - enable_open_as_wellformed 81
 - enable_toolbar_customization 81
 - entity extension 81
 - expand_entities_in_tags_on_view_too 81
 - extid_map 45
 - log file 81
 - log_file 81
 - make_log_file 81
 - OASIS_override 44
 - rules_checking_always_off 81
 - tbr_path 81
 - use_extid_mapping 45
 - user name 81
- xmetal50.ini (*continued*)
 - validate_before_export 81
 - validation disabled 81
 - XML
 - xml:lang 47
 - XML template
 - creating 8
 - XMmkrules 38
 - XSD
 - creating 37
 - creating, overview 8
 - modifying 37
 - viewer 37
 - XSL 60
 - XSLT 60