

Quark XML Author 2015 October 2018 Update for Microsoft Word — System Admin Guide

Contents

Preface	1
Why Quark XML Author?	1
Features	1
Usability	1
Enterprise Fitness.....	3
Using This Manual	3
Intended Audience	3
Configuration Overview	4
Configuration Synopsis	5
Word Configuration	5
Disabling commands	5
Repurposing commands	5
Special considerations: command and buttons	6
User Interfaces	6
Dynamic Configuration	7
Configuration: ShortcutKeys	9
ShortcutKeys Configuration in the AppConfig File	9
ShortcutKeys Configuration in the DocConfig File	9
Associating Shortcut Keys with Menu Items and CommandBarButton.....	9
ShortcutKey	10
Supported Keys	11
Unsupported Keys	11
InternalClass.....	11
ExtensibilityInterface	12
Global Shortcut Key File: Disabling Shortcut Keys	12
Configuration: Ribbon, Office Menu, and Backstage View	13
Single and Multiple Ribbon Configurations	13
Single Ribbon.....	13
Multiple Ribbons.....	13
Example	14
Ribbon Strings and Resources	14
Word Backstage View	14
Ribbon Nodes	15

Icon IDs for use in imageMso	15
<box>	16
<button>.....	17
<buttonGroup>	18
<checkBox>	19
<command>	20
<commands>.....	21
<contextualTabs>	22
<customUI>	22
<dropDown>	22
<group>.....	23
<menu>	24
<menuSeparator>.....	26
Word Backstage View	26
Quick Access Toolbar (QAT).....	34
<ribbon>.....	34
<separator>	35
<splitButton>	36
<tab>	36
<tabs>	37
<toggleButton>	38
Unique Ids.....	41
Undo	42
Printing	42
Word 2010 and later	42
Hiding/Disabling Ribbon Tabs for Third-Party Software	43
Application Configuration: Internal Classes	44
AcceptRevision	44
AssignAttribute.....	45
ChangeToList.....	45
ChangeToPara	47
ClearUndoStack	47
Copy.....	48
CreateRendition	48
filter	49
StyleSheet Child Nodes	50
Using CreateRendition to Load External Objects	50
Cut	51
DeleteComment.....	51
DeleteTable	52
DeleteTableCol.....	52
DeleteTableRow	52
DocumentAttributes	52
DocumentClose	52

DocumentNew	52
DocumentOpen	53
filter	54
DocumentOpenWord	55
DocumentSave	55
DocumentSaveAs	56
filter	57
EmailRendition	57
filter	59
EmphasisAction	60
EmphasisHandler	61
FindAndReplace	61
FormattingAction	62
FormattingActionChangeCase	64
FormatTableCellShading	64
Selections containing cells with different color attribute definitions	65
IndentElement	65
InlineAttributes	67
InsertXACrossref	67
InsertComment	67
InsertColumnBreak	67
InsertCustomLink	67
InsertElement	68
displayName	68
Insert Emphasis	69
InsertEndNote	70
InsertEntityReference	70
InsertFootnote	70
InsertHyperlink	71
InsertInlineElement	71
InsertPageBreak	72
InsertSectionBreak	72
InsertTable	72
InsertTableColumn	73
InsertTableRow	73
ManageTextEntities	73
MergeComments	73
Selecting Files	74
Automatically Merging All Comments	74
MergeTableCells	74
PageLayout	74
Paste	75
DefaultPasteOptionAsText	75
PreviewRendition	75

filter	77
StyleSheet Child Nodes	78
Using PreviewRendition to Load External Objects	78
Redo.....	78
RejectRevision.....	78
SaveToRepository	79
Tokens	79
Delegates.....	79
SetAccessMode.....	80
User Experience	80
Access Modes and SharePoint.....	81
SetColumns	81
SpellChecker	81
ShowAttributesHandler	81
SplitTableCells.....	82
StyleHandler	82
TableAutoFitBehavior	82
TableBorders	83
TableCellAlignment	83
TableDistributeColumns	84
TableDistributeRows	84
TableRefresh	84
QuarkSubSection	84
ToggleCommentsPane	84
ToggleEmpty.....	84
ToggleEntityView.....	84
ToggleKeepWithNext	84
ToggleTrackChange.....	85
ToggleVisibleElement	85
ToggleWidowOrphanControl	85
Undo	85
UpdateSchema	86
WordDialogEdit	86
Application Configuration: Extensibility Interface	87
Programming for Quark XML Author	87
Calling Quark XML Author from an External Method with Delegates.....	88
Configuring the Extensibility Interface	88
Building the EI Method.....	88
Calling the EI Method.....	93
List of Available Enumerated Values.....	99
XOM Defined	104
List of Available Delegates	104
List of Available Document Events	141

Application Configuration: ResolveReferences	147
The Reference Node	147
Pattern Attribute	147
Assembly and Class Attributes	148
Tokens	148
Application Configuration: Miscellaneous items	149
CultureName	149
Language and Numerical Content.....	149
EnableProgressDialogs	149
ErrorLogging.....	149
ExtendedLoggingInfo	149
LogFilePath	150
UndoHistory	150
Namespaces	150
Namespace	150
NotSupported.....	152
TempFilePath	153
Blocking Shortcut Keys	153
DisableTemplateAddins.....	154
Disable all add-ins.....	156
Disable all add-ins except the specified list.....	156
Disable the specified list of add-ins	156
ImageFilePath	156
Document Level Configuration	158
Portionmarks.....	158
ReferenceShadingColor	160
AllowRestartContinueNumbering	160
DefaultSaveOptions	160
ComponentCopy	160
Commenting.....	161
Attribute.....	161
Break	162
SectionBreak	162
PageBreak	162
ColumnBreak	162
AllowHeaderFooterEdit	163
AllowSoftLinebreaks	163
Smart Paste	163
Reviewing	163
HiddenInsertables	164
Change To Menu	164
ShowInsertBeforeMenu	164
PasteTrackDeletedText	165

ShowInsertAfterMenu	165
EnableFastSave	166
EnableOpenDocxSupport	166
dir	166
Application Configuration: XpressUpdates	167
Overview	167
XpressUpdate.xml	167
Manifest.xml	168
Dialogs	168
Quark XML Author Structure Overview	170
XML Declaration and Processing Instructions	170
Defining Multiple Namespaces.....	170
Root Element	170
External Entities	171
Definition Order	171
Attributes	171
Emphasis Styles.....	172
Elements	172
Comment Lines	172
Attribute Groups	173
Common Attributes	174
Field Attributes	174
Quark XML Author Structure Attribute Definition	187
AttributeDef	187
Restriction	187
Using uniqueidentifier.....	188
Associating Attribute Definitions with Elements and Emphasis	188
SectionBreak, PageBreak, and ColumnBreak Attributes	188
Quark XML Author Structure Emphasis Definition	190
EmphasisDef	190
Specialized	191
Style	199
Substyles	200
Attributes and Attributes Type.....	201
Associating Emphasis with Elements	201
Extensibility Methods.....	202
Restricting Emphasis Nesting	204
Inline Media	205
See section 14.6.2, "Images"	205
Inline OLE	205

Quark XML Author Structure Element Definition	207
ElementDef	207
Section	215
Para	216
Sequence	222
Choice	223
Uniqueidentifier	230
Unmanaged	230
OLE	230
Media	232
Reference	235
EndNote	237
Table	238
Support for language-specific configurations	281
UI controls and features	281
Icons	283
Context Menus	283
Resource Files	283
Modifying file filter resource strings	287
Extensibility Methods	291
 Starter Documents and Templates	 294
Starter Document	294
Document Content	294
Namespace	295
DOCTYPE declarations	295
Microsoft Word Templates	295
Preparing to Use TemplateManager	296
Removing temporary files	296
CollectControls.exe	297
To Launch TemplateManager	298
Application Config Tab	299
Document Config Tab	301
Converting Word 2003 configurations to the latest supported version	305
Enabling the Styles button in standard Word	305
 Structured Authoring	 307
Deleting Content Elements and Structure	307
Backspace and Delete in Structured Authoring	307
Entering / Pasting Text - Advanced	308
Typing in an Empty	308
Tab in Structured Authoring	308
Cut, Copy, Paste in Tables	309
General	309
Pasting Table Elements	310

Pasting Table Elements Into a Selection	310
Pasting Table Elements Without a Selection	311
Creating Tables from Copied Table Elements	313
"strict XML" execution of Cut and Copy	313
Multiple element - Cut, Copy, Paste	314
 Integration with Content Management Systems	 316
SaveToRepository	316
Example	317
Upload() Interface	317
Example 1: Upload without delegates.....	318
Example 2: Upload with delegates	319
Example 3: Upload a document fragment.....	321
Download and Content Reuse	323
Open a CMS Document in Quark XML Author.....	323
Importing content into existing documents.....	325
Download Interface	327
Example 1: Drag and Drop External Content.....	329
Example 2: Using Paste to Insert External Content	329
Example 3: Opening a Document Referencing External Content.....	330
 Configuring Smart Paste	 331
Smart Paste Overview	331
Smart Paste Extensibility Method	332
Configuration-specific transforms	332
Configuration-specific table transforms	333
Enabling Smart Paste	333
The Intermediate Schema	333
TextType Elements	334
Table	335
Image	335
Final Transformation Stylesheet.....	336
Multipart Wrapper	336
Example: Intermediate Heading > Quark XML Author Section.....	337
Example: Handling List Items	338
Example: Handling Table Rows.....	339
 Configuring MetaFormsBridge.....	 340
XpressRun Method	340
XpressRun Tokens	341
MetaForms Files	342
Example	342
 Implementing Cross References.....	 343
AppConfig.....	343

DocConfig	343
EI Method	343
ComponentCopy Node	343
ElementDef	344
EmphasisDef	344
Configuring the Cross-reference dialog	345
Configuring dialog listings based on target element names (XMLNames) or XPath.....	345
Extensibility Interface Use Case Study	348
Inserting Elements Through EI in Quark XML Author	348
Example 1	349
Example 2	354
MathType Integration.....	356
Implementing MathType Support	356
Serialization.....	357
Developer Notes.....	357
OLE Word Document Integration.....	358
Embedded Table User Experience	358
Editing.....	359
Viewing	359
Copy and Paste.....	359
Implementing Embedded Table Support	359
Starter documents.....	361
Sample configuration	361
Images	362
Emphasis.....	362
Metadata.....	364
Smart Content	370
Overview	370
Serialization.....	370
Word Tables	371
Why are there empty values in the smart content?.....	372
Serializing Metadata	372
Images	374
The Word canvas.....	374
Image sizing logic.....	374
Rounding Error Issue.....	374
Implementing a Custom XML Resolver	376
Implementation Overview	376

How Microsoft and Quark Support Languages and Cultures...	378
Bidirectional language support	380
Bidirectional language support in Microsoft Office	380
Bidirectional language support in Quark XML Author	380
Configuring the XAS to allow support of direction attributes for elements	381
Configuring the user interface for bidirectional language support	381
Configuring bidirectional language support at the content level	382
Configuring bidirectional language support at the element level	383
Configuring bidirectional language support at the document level	383
IME based:Supporting New Languages	384
What Must Be Translated	384
Translation Process Overview	384
(External)	385
Icons (Internal)	385
Shortcut keys (Internal)	386
Friendlylies (Internal)	386
Localization: A step-by-step example	386
Pre-requisites	386
The localization procedure	386
Variable Reference Support	390
Section Type Elements	390
Reference Type Elements	390
Additional delegates	391
Installation of Quark XML Author	392
Automated installation on a new machine:	392
Or for manual installation on a new machine:	392
Appendix A – Color Names	394
Appendix B – Change Log	398
Changes in version 6.3	398
Changes in version 6.2	398
Changes in version 6.1	398
Legal notices	400

Preface

Why Quark XML Author?

When given the choice, knowledge workers overwhelmingly opt to use their current authoring tool—Microsoft Word. They would also like to use Word without concern for the technical aspects of XML. It is only natural that your knowledge workers would like Word to create the complex XML documents your organization needs automatically and invisibly.

This is exactly what Quark XML Author for Microsoft Word does. Quark XML Author is an add-on product that seamlessly combines with Microsoft Word and does not affect Word functions—except when the author chooses to create an XML document. Then, Quark XML Author takes over to create an environment where Word menus and functions look and seem to operate normally, and content is presented as the user works—without the distraction of XML tags, structures, or rules. Quark XML Author ensures that content is valid as it is authored—not sometime after the fact.

The result is a natural authoring experience for knowledge workers who can be sure that documents will meet the technical requirements of their organization’s critical initiatives.

Features

Usability

Quark XML Author delivers high-end XML authoring in Word through a host of features,

including:

Quark XML Author delivers high-end XML authoring in Word through a host of features, including:

- A configurable, non-linear mapping between the XML Document Object Model (DOM) and the Word Interface. The “normal” way to add XML elements is from a drop-down list. This method requires the author to know what an XML element is, which element should be used, and how to insert it. Quark XML Author maps element classes to any GUI object. For example, textual elements may be inserted using the Style toolbar. Pictures and other references may be

inserted using the Insert menu along with its normal sub-menus. Table elements may be inserted through the Table menu.

- Continual monitoring of the user's activity and dynamic activation of menu and toolbar functions only when they will create XML that conforms to the DTD or Schema in use. When the user inserts an element, Quark XML Author automatically creates mandatory parent or child elements and transparently places the user within the proper element.
- A powerful data pipe for structured and unstructured data. Users may create external references to graphics or text using drag-and-drop from a Content Management System or they may reference external data sources for runtime creation of virtual documents. Automation of metadata tasks so that the organization can gain search accuracy and efficiency without overburdening users. This includes pre-filling of metadata, automatic inheritance, dynamic roll-up from element to document level, and more.
- Exclusive What You See Is What You Need Interface. XML tags and formatting are hidden and Quark XML Author enhances the Spell Check, Track Changes, and other advanced functions so that they work seamlessly with the visible and invisible layer.
- Loads XML and normal Word documents at the same time – no need to exit and restart Word.
- Just-Write Editing monitors every keystroke and frees the author from thinking about XML rules. Try to edit referenced text? Quark XML Author will ask if you would like to remove the reference and make the text your own. Press enter in the middle of a paragraph? Quark XML Author splits the paragraph and creates all the proper XML automatically – including mandatory related elements and inherited attributes – behind the scenes and out of the author's view. Multimedia and textual referencing Content may be dragged from a Browser-based Content Management System into Quark XML Author – the reference is automatically inserted and traversed to display content in-line.
- Referenced content may be locked to prevent editing, or configured for optional de-referencing and editing by the current author. Either way, the system records the original source and any edits that may occur in the current context.
- Future References allow placeholders to be passed to a Content Management System and work-orders to be passed to a Workflow System with automatic update of content when the work is completed.
- Dramatically reduces the effort required to provide a high level of detailed metadata that ensures accurate retrievals.
- Automatically populates metadata and lists from LDAP, databases, or other operational data sources.
- Provides cascading lists and configurable screens to speed the entry of any remaining metadata.
- Automatically copies metadata from one element to the next and “rolls-up” metadata from elements and referenced content to the document level under configurable rules. Unsurpassed help for metadata-weary users

- Quark XML Author is designed to operate with real-time integration to a content management server, workflow server, or other network resources. The application may also be configured to run in a “briefcase” mode to support mobile or occasionally-connected users.

Enterprise Fitness

Quark XML Author handles the most complex requirements without compromise:

- Support for very complex DTDs or Schemas.
- Real-time connectivity for population from or to enterprise data sources.
- Open XML communication with Content Management Systems, Workflow Systems, and other network resources.
- Version control over referenced content with locking to a fixed version or to the latest version. De-referencing of content with notification and audit trail. Atomic blocks of referenced content ensure contextual integrity.
- Microsoft C# implementation of .NET form mission critical stability monitors every keystroke and mouse click—enhancing or replacing standard Word functions as needed. No fragile macros or VBA to break.

Using This Manual

Configuring and integrating Quark XML Author involves several processes. This manual provides instructions for:

- Configuring the Word File menu and Word ribbon to include only those that conform to a valid XML output.
- Configuring access to the Extensibility Interface, a utility linking to an external process for the retrieval of defined values.
- Defining the messages displayed in Microsoft Word as they relate to the document.
- Defining the structure of a document class using the Quark XML Author Structure (*.xas).
- Configuring the Microsoft Word templates for each document class to determine document layout and presentation.

Intended Audience

This guide has been prepared for persons responsible for configuring and integrating Quark XML Author for Microsoft Word. The reader is expected to have knowledge of XML structure, syntax, and standard terminology, as well as of Microsoft Word.

Configuration Overview

The configuration of XML Author specifies the user interface within Microsoft Word and assigns XML Author features and functionality to those user interface controls. Word 2010 and later expose a user interface called the Backstage view.

At the document level, the user interface for XML Author must be designed for the following scenarios:

- Word with no document or a standard Word document loaded
- Word with an XML Author document loaded

This is true for all supported Microsoft Word versions.

At the control level, the following user interface controls may be customized:

- User interface labels
- Titles
- Descriptions
- Screentips
- Keytips

The text for these is created and maintained in a resource file separate from the configuration files for ease of localization.

Quark XML Author makes it possible to alter or disable Microsoft Word commands and options to maintain the XML structural integrity of a Quark XML Author document. These options can be constrained to prevent Quark XML Author users from invalidating the document against an XML Schema, even when the same action may be allowed in MS Word.

Two files specify how constraints apply to commands and options:

1. AppConfig.xml: applies regardless of what Quark XML Author document class is active.
2. DocConfig.xml: a document-class specific file which allows your organization to develop multiple document classes. Each of these may have a unique interface within Word. The name of this document-level configuration file may vary and is referred here as DocConfig.xml for the sake of convenience.

Configuration Synopsis

The main components for configuration reside in AppConfig.xml and the DocConfig.xml file:

The following components are common to all supported versions of Microsoft Word.

Shortcut Keys. Shortcut keys define keystroke combinations to launch common Quark XML Author or Microsoft Word commands.

Extensibility Interface. This section identifies the arguments required to launch the Extensibility Interface, which provides access to external processes for defining or auto-populating attribute values for content elements.

Details for each component appear in later sections of this document.

Word Configuration

Disabling commands

A Word command can be enabled/disabled at the command level. This enables/disables the command across all occurrences in the Word user interface including: Ribbon, Toolbar, Menus, Backstage and Quick Access Toolbar.

To disable a command, specify the command without a repurpose designation.

For example,

```
<command idMso="MyCommand"/>
```

See “<command>”.

The UI is disabled in XML Author and enabled in Word.

Repurposing commands

A Word command can be repurposed at the command level to a new function. This repurposes the command to a single new function across all occurrences in the Word user interface including: Ribbon, Toolbar, Menus, Backstage, and Quick Access Toolbar.

For more information on repurposing controls, see the [MSDN web page](#):

Search for the phrase “Overriding Commands and Repurposing Controls”.

To repurpose a command, specify the command with a repurpose designation.

For example,

```
<command idMso="FileSave">
  <InternalClass name="DocumentSave" nativeFormat="false"
  filter="Save2007"/>
</command>
```

The UI is enabled in XML Author and Word. In XML Author, the command invokes the repurpose designation. In Word, the command invokes the standard Word feature.

Note: On the QAT, the only Word command that cannot be repurposed/overridden is Undo. See “Quick Access Toolbar (QAT)”.

Special considerations: command and buttons

The visibility of a Word command is not controlled at the command level, but rather at the individual button or menu. See “Common Attributes”.

If there are any UI elements (i.e. Ribbon button) using the same idMso, their visible, enabled, and action override behavior will be inherited from the command. On the other hand, if an idMso is used in a UI element but not a command, then these values can be set explicitly. The enabling/disabling behavior will be inherited from the command and should not be used in the UI element.

Table 2-1: Visibility Settings

Settings may be combined using the pipe character.

For example visible = “Word | OLEWordDocument”.

Visibility	Attribute
visible in all Word and XML Author documents	visible=”true”
not visible in all Word and XML Author documents	visible=”false”
only visible in XML Author documents	visible=”xa”
only visible in Word documents	visible=”Word”
only visible in embedded Word documents	visible=”OLEWordDocument”

Note: If a command is not defined, and a particular idMso is used in multiple UI elements, the visibility of the first declaration found in the configuration will be used, and all other UI elements will inherit from the first declaration.

Note: An action override for an idMso can only be set in a command node, not in a UI element.

XML Author cannot hide commands on the QAT. See “Quick Access Toolbar (QAT)”.

User Interfaces

Ribbon. The ribbon is a component of Word.

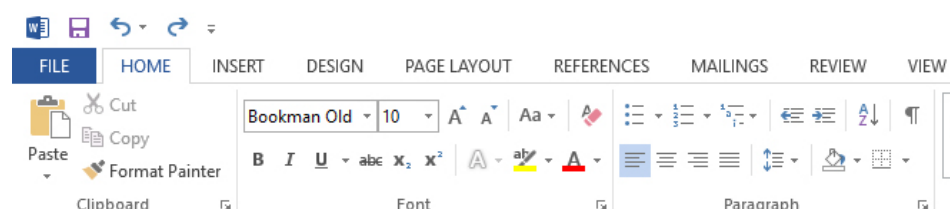


Figure 2-1: The Word 2010 Ribbon

Backstage View. The Backstage view is used only in the File menu of Word 2010 and later.

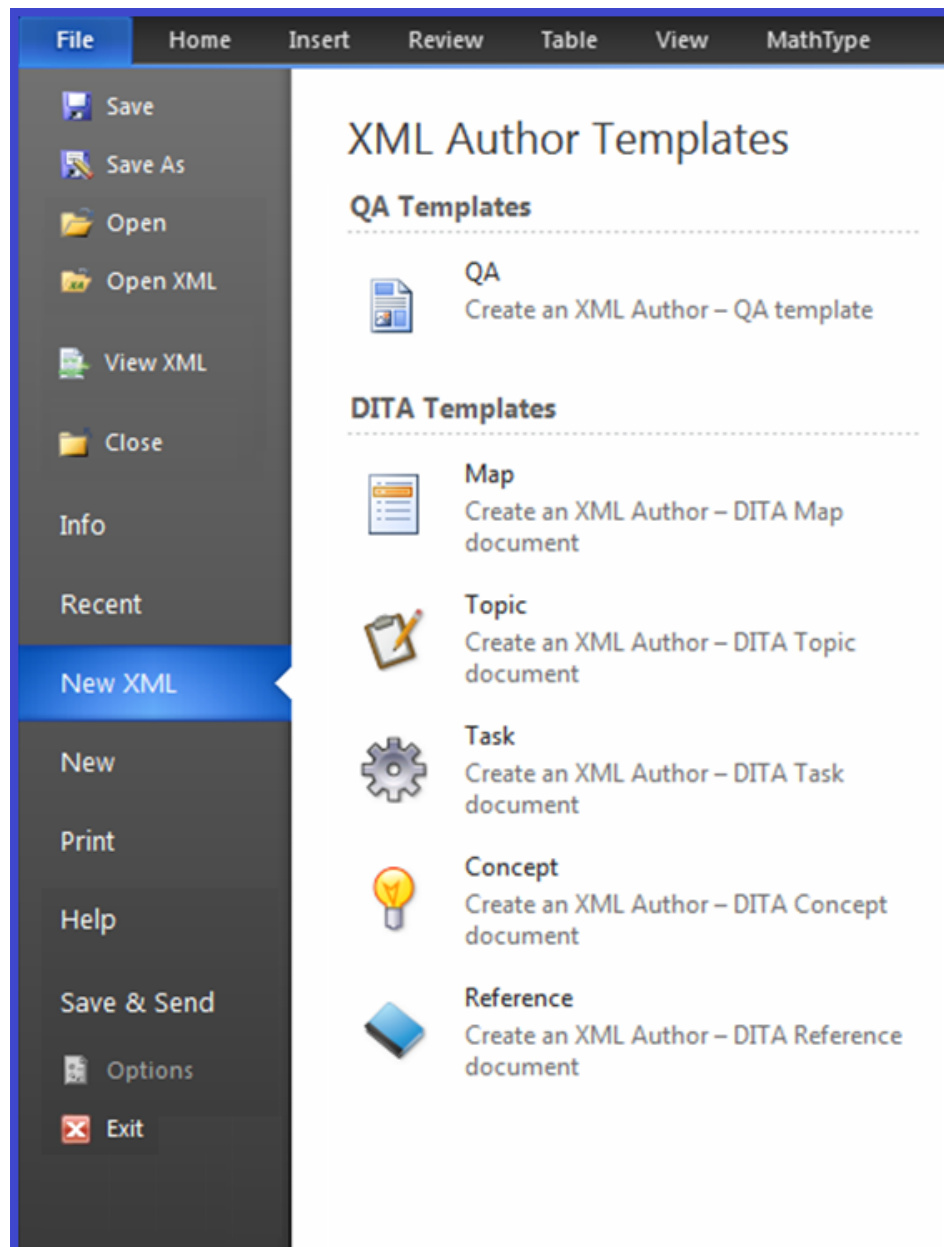


Figure 2-2: File > XML Author submenu

Dynamic Configuration

Dynamic configuration allows the system to query and pull custom component and configuration updates from the server. Configurations can vary based on user roles and can be managed in a Platform collection. Configuration updates can be pushed to clients, and clients can check for configuration updates on launch.

The following components allow for dynamic configuration:

OverrideConfig event. This event enables the custom Extensibility Interface to pull updates to the configuration.

OverrideAppConfigPath delegate. This delegate enables the custom Extensibility Interface to override the application configuration path.

These components can be defined in the AppConfig file as shown in the following example:

```
<ExtensibilityInterface>
<MethodInfo>
  <Method
<Method id="OverrideConfiguration" assembly="external application
assembly
  name" class="external application class name"
  method="OverrideConfiguration">
    <Argument type="Tokens">
      <!--Server ConfigurationPathIdentifier - Specifies the CMS
specific
        identifier for the configuration folder at the server. For
Platform,
        the identifier supports a role based collection path (*
signifies the
        default or fallback role):
          Role|<Role 1>,Path|<Collection Path 1>;Role|<Role
2>,Path|<Collection Path 2>
        LocalConfigurationPath - Specifies the folder path at the
local file
        system where the downloaded configuration if kept.
        CopyFilter - Specifies the recursive file search filter for
files in
        The installation location to be copied to the local
configuration
        path.You can specify multiple filters using "|" as a
separator. This
        filter is required only if the server folder keeps partial
configurations.-->

      <Token>Server ConfigurationPathIdentifier=

Role|*,Path|;Role|Analyst,Path|Home/Configuration/Analyst;
      Role|Editor,Path|Home/Configuration/Editor
    </Token>
    <Token>LocalConfigurationPath=
      %Appdata%\Quark\XML Author\DynamicConfig
    </Token>
    <Token>CopyFilter=
      Schemas/*. *|en/*. *|es/*. *|fr/*. *|ja/*. *
    </Token>
  </Argument>
  <Argument type="Delegates ">
    <Delegate>OverrideAppConfigPath</Delegate>
  </Argument>
</Method>
</MethodInfo>
</ExtensibilityInterface>
```

Configuration: ShortcutKeys

Shortcut keys let you define keyboard combinations to common Quark XML Author commands. These commands are launched by pressing a single keystroke (for example, **F12**) or by pressing a combination of a keystroke and the **Shift** or **Ctrl** or **Alt** key. The **ShortcutKeys** section of the configuration files defines these shortcuts and the commands they launch.

The shortcut keys that should always be blocked are specified in the global **XAShortcutKeysConfig.xml** file. This file is deployed with every installation.

Shortcut keys can be defined and modified both the **AppConfig** and **DocConfig** files, and the procedures are the same in both files.

The order of precedence for key configurations is that the **DocConfig** overrides the **AppConfig** which overrides the global **XAShortcutKeysConfig.xml** file.

ShortcutKeys Configuration in the AppConfig File

ShortcutKeys can be configured in both the **AppConfig** and **DocConfig** files. Configuration in the **AppConfig** file will be reflected in the Word environment regardless of the active document class. Quark recommends that you do not modify **XAShortcutKeysConfig.xml**, but use the **AppConfig** file to configure shortcuts that you want users to be able to use even if they have a standard Word document open.

ShortcutKeys Configuration in the DocConfig File

Use the **DocConfig** file to configure shortcuts that should only be available for a specific document class. You may also need to block standard Word shortcuts in the **DocConfig** file; how to block shortcuts is covered in section 8.9.

Associating Shortcut Keys with Menu Items and CommandBarButton

Menu Items and CommandBar Buttons can both have shortcut keys associated with them through the use of the **ShortcutKey** child element. To assign a shortcut to a Menu Item or CommandBar Button, include an empty **ShortcutKey** element in the **MenuItem** or **CommandBarButton** element.

ShortcutKey

The ShortcutKey element's attributes specify the character to use, and whether the Shift, Ctrl, Alt, or some combination of the three are part of the shortcut key combination. In the example below, the combination Ctrl + O launches the command:

```
<ShortcutKey key="o" shift="false" ctrl="true"/>
```

The Key attribute value is the character to use. The values of shift and ctrl are set to **false** if they are not used in the shortcut key, or true if they are. In the example below, the combination Ctrl + Shift + x launches the menu item:

```
<ShortcutKey key="x" shift="true" ctrl="true"/>
```

Note that the values are case-sensitive.

Table 3-1: ShortcutKey Attributes

Attribute	Type	Required	Definition
alt	boolean	no	If set to true , the Alt key is part of the shortcut key combination. Set to false if the Alt key is not used.
ctrl	boolean	yes	If set to true , the Ctrl key is part of the shortcut key combination. Set to false if the Ctrl key is not used.
key	string	yes	The keyboard key that is used in the shortcut key combination. All keys on the keyboard can be configured except for Backspace and Tab. Numeric keypad keys can be configured using the values shown in Table 3-2. The enter key may be represented by either Return or Enter .
shift	boolean	yes	If set to true , the Shift key is part of the shortcut key combination. Set to false if the Shift key is not used.

Table 3-2: Numeric KeyPad Key Values

Keypad Key	Key Attribute Value
1	NumPad1
2	NumPad2
3	NumPad3
4	NumPad4
5	NumPad5
6	NumPad6
7	NumPad7
8	NumPad8
9	NumPad9
0	NumPad0
/	Divide
*	Multiply
-	Subtract
+	Add
.	Decimal

Table 3-2: Numeric KeyPad Key Values

Supported Keys

Supported keys can be used in any combination with Shift, Ctrl, and Alt keys.

The list of supported keys includes:

+, =, -, |, \, {, }, [,], ~, ` , ? , / , > , < , , (comma), \ , ' , ; , Return, Enter

The list of supported virtual keys includes:

13, 186 – 192, 219 - 222

Unsupported Keys

The list of supported keys includes:

Home, End, Page Up, Page Down, Arrow keys, Tab, Enter

InternalClass

If the shortcut is for an internal command (one launched within Microsoft Word, such as Cut, Copy, or Paste), add an InternalClass child element to the ShortcutKey element.

The InternalClass element identifies the internal class value (for example, Cut) that will launch the command. Enter the internal class name as the value of the name attribute of the InternalClass element, as shown in the example below.

CONFIGURATION: SHORTCUTKEYS

```
<InternalClass name="Cut"/>
```

In the example below, the shortcut key combination Shift + Del will cut the selection from the document and place it on the clipboard.

```
<ShortcutKey Key="DEL" Shift="true" Ctrl="false">
  <InternalClass name="Cut"/>
</ShortcutKey>
```

ExtensibilityInterface

If the shortcut is for an external command (one that launches an external process), add an ExtensibilityMethod child element to the ShortcutKey element.

The ExtensibilityMethod element identifies the ExtensibilityInterface method (such as Popup) that will launch the command. The ExtensibilityMethod element has one required attribute: id. The value of the id attribute is the name of the Extensibility Method specified in the EI node of the configuration file. An example is shown below.

```
<ExtensibilityMethod id="Popup"/>
```

In the following example, the shortcut key F12 launches the Popup ExtensibilityInterface Method.

```
<ShortcutKey Key="F12" Shift="false" Ctrl="false">
  <ExtensibilityMethod id="Popup"/>
</ShortcutKey>
```

ExtensibilityInterface methods are defined in the Extensibility Interface, which is covered in Section 6.

Global Shortcut Key File: Disabling Shortcut Keys

The Global Shortcut Key file contains the list of shortcut keys that are disabled by Quark XML Author in all configurations.

The following is a fragment of the XAShortcutKeysConfig.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<!-- This is a list of universal shortcuts to disable on all
configurations. Keys can be repurposed in DocConfig ShortcutKeys
node. -->
<ShortcutKeys>
  <ShortcutKey key="B" shift="true" ctrl="true" />
  <!-- remove bold -->
  <ShortcutKey key="C" shift="true" ctrl="true" />
  <!-- copy formatting -->
  <ShortcutKey key="D" ctrl="true" />
  <!-- font dialog -->
</ShortcutKeys>
```

Configuration: Ribbon, Office Menu, and Backstage View

Ribbons, Office button menus, menu items and Backstage views can be defined and modified in both the AppConfig and DocConfig files, and the procedures are the same in both files.

Configuring Quark XML Author for Microsoft Word requires adding a new node (<ribbon>) to AppConfig.xml and, if multiple ribbon configuration support is needed, the DocConfig file as well.

Single and Multiple Ribbon Configurations

Single Ribbon

If a single ribbon configuration is required, the entire ribbon can be configured in AppConfig.xml. This includes:

- Defining the tabs that will be visible in the standard Word interface (i.e., no document is loaded or the current active document is not a Quark XML Author document).
- Defining the Quark XML Author tabs that will be visible when a Quark XML Author document is the active document.
- Defining the controls on Quark XML Author tabs that will be visible when a Quark XML Author document is the active document.

Multiple Ribbons

If multiple ribbon configurations are necessary, ribbon configuration is split between AppConfig.xml and the DocConfig file. Global UI definitions are placed in the <Ribbon> node in AppConfig.xml. This includes:

- Defining the tabs that will be visible in the standard Word interface (i.e., no document is loaded or the current active document is not a Quark XML Author document).
- Defining the Quark XML Author tabs which appear in all ribbon configurations when a Quark XML Author document is the active document.
- Defining the controls on Quark XML Author tabs which appear in all ribbon configurations when a Quark XML Author document is the active document.

CONFIGURATION: RIBBON, OFFICE MENU, AND BACKSTAGE VIEW

Each DocConfig file defines a separate ribbon configuration. The <Ribbon> node in the DocConfig file defines only those tabs and controls that are visible for that ribbon configuration. This includes:

- Defining tabs that will only be visible for a specific ribbon configuration.
- Defining controls that will only be visible for a specific ribbon configuration.

Example

For example, assume two document classes. Class A may contain tables. Class B cannot. This necessitates the following configuration structure:

- The <Ribbon> node in AppConfig.xml would define the Quark XML Author version of the Insert tab and specify that it only appears when a Quark XML Author document is the active document.
- The <Ribbon> node in AppConfig.xml would specify that the native Word Insert tab not appear when a Quark XML Author document is the active document.
- The <Ribbon> node in the DocConfig for Class A would contain a button for inserting tables.
- The <Ribbon> node in the DocConfig for Class B would not contain a button for inserting tables.

In this example, the resource file contains the strings for the XML Author Insert tab label and the Insert Table button's label, description, screentip, and keytip. See **"Error! Reference source not found."**.

Ribbon Strings and Resources

In addition, the resource file must be populated with the strings for user interface labels, titles, descriptions, screentips, and keytips. See **"Error! Reference source not found."**.

Specify the various attributes by concatenating the control's id value with an underscore followed by the attribute name. Consider the following menu item:

```
<menu id="XMLAuthor" image="QuarkXMLAuthor.ico"
insertBeforeMso="FileNew" itemSize="large" visible="true">
```

To specify label and title values, add the following entries to the resource file:

XMLAuthor_label = "New XML"

XMLAuthor_title = "New Business Document"

This resource file is created and compiled the same as other resource files.

Word Backstage View

The <backstage> node defines the items that appear in the Word File menu. The <backstage> node is a child of <Ribbon>\<customUI>. See 4.4.14 for details.

The example below is specific for Word 2010 Backstage View

```
<Ribbon>
```

```
<customUI>
  <backstage>
</backstage>
</customUI>
</Ribbon>
```

The example below is specific for the Backstage View in Word 2013 or later.

```
<Ribbon>
<customUI>
  <backstage2013>
</backstage2013>
</customUI>
</Ribbon>
```

Ribbon Nodes

The <Ribbon> node follows immediately after the <CommandBars> element in AppConfig.xml and in the DocConfig file.

The general structure of the <Ribbon> node looks like the XML shown below. The xmlns attribute should be the first attribute.

```
<Ribbon>
  <customUI
xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="OnLoad" loadImage="LoadImage">
    <commands>
      <!-- one or more command nodes -->
    </commands>
    <ribbon>
      <!-- all of the nuts and bolts of the office
menu and ribbon tabs -->
    </ribbon>
  </customUI>
</Ribbon>
```

The following subsections describe each available node in alphabetical order.

Icon IDs for use in imageMso

For several of the user interface controls on the ribbon, the imageMso attribute contains the id value of the image displayed on a ribbon button by Word. These values (or names) may be found in the following Microsoft article:

[“2016 Office System Document: Lists of Control IDs”](#)

The article states “These names are also accessible within the 2016 product by hovering over controls in the QAT customization dialog, and looking at the screenshot of the control.” The article also contains a link to download collection of spreadsheets which contain these values for use in the imageMso attribute.

Or you can view these icons and their values in Microsoft Excel in the Developer tab of the Ribbon.

To display the Developer tab in the Excel Ribbon:

1. Click the Application button, then click **Excel Options**.
2. In the Excel Options dialog, click **Popular**.

- 3. Check Show Developer tab in the Ribbon.
- 4. Click OK.

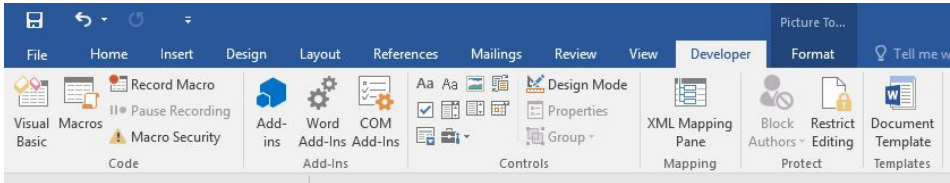


Figure 4-1: Excel Ribbon – Developer tab

- 1. Click a Gallery button.
- 2. In the Gallery drop down, click the desired icon.
- 3. In the imageMso dialog, record the imageMso value for the selected icon. For example “AcceptInvitation”.

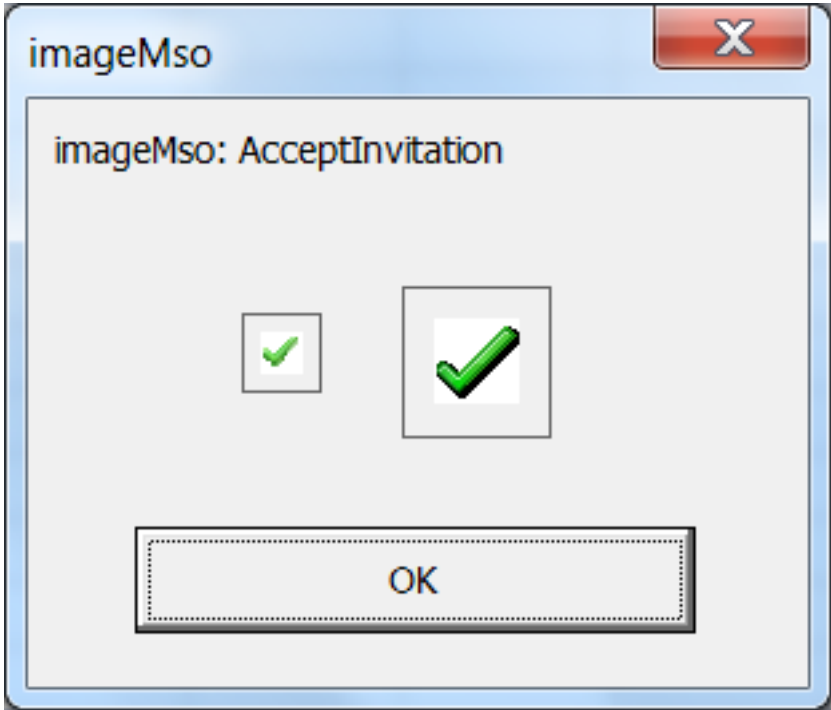


Figure 4-2: The imageMso dialog displays the icon’s value

<box>

The <box> node can be used to collect a series of controls within the <group> node into a vertical or horizontal box.

Common Attributes

The <box> node uses the required attribute *id*.

Unique Attributes

The <box> node uses the unique attribute described in Table 4-1.

Table 4-1: <box> Attributes

Attribute	Value	Required	Description
boxStyle	horizontal vertical	yes	Defines the orientation of the box.

Child of

The <box> node may be a child of the following element:

- <group>

Parent of

The <box> node may contain one or more of the following elements:

- <button>
- <buttonGroup>
- <checkBox>
- <command>
- <menu>
- <splitButton>
- <toggleButton>

Example

The example below contains two dropdown lists and a button group in a horizontal row.

```
<box id="StyleEmphasisDropdowns" boxStyle="horizontal">
  <dropDown id="StylesListXA" showLabel="false"
sizeString="XXXXXXXXXXXXXXXXXXXX">
    <!-- Control details omitted -->
  </dropDown>
  <dropDown id="EmphasisListXA" showLabel="false"
sizeString="XXXXXXXXXXXXXXXXXXXX">
    <!-- Control details omitted -->
  </dropDown>
</box>
```

<button>

The <button> node defines a button control.

To use a native Word button, specify a value for the idMso attribute. The specified Word button is then used without modification; the <button> node for this type of button is a leaf node.

To define a Quark XML Author button, specify a value for the id attribute. Child elements define the button behavior.

Common Attributes

The <button> node uses the following “Common Attributes”:

- id or idMso (required)
- image
- imageMso

- insertAfterMso
- showLabel
- size
- visible

The following common attributes for the <button> node may still be used but have been deprecated:

- description
- label
- screentip

Define these values in the “Resource Files”.

Child of

The <button> node may be a child of the following nodes:

- <box>
- <buttonGroup>
- <group>
- <menu>
- <officeMenu>
- <splitButton>

Parent of

The <button> node may contain the following nodes:

- <InternalClass>
- <ExtensibilityMethod>
- <ShortcutKey>

These nodes define the behavior of the control and are defined in the System Admin Guide.

Example

In the example below, the button invokes the “PasteChart” Extensibility Method and can also be triggered with the shortcut key Ctrl + V.

```
<button id="PasteChartXA" imageMso="Paste" size="large"
label="Paste Chart">
  <ExtensibilityMethod id="PasteChart"/>
  <ShortcutKey key="V" shift="false" ctrl="true"/>
</button>
```

<buttonGroup>

The <buttonGroup> node contains a collection of one or more buttons or toggle buttons.

Common Attributes

The <buttonGroup> node uses the required attribute <id>.

Child of

The <buttonGroup> node may be a child of <box>.

Parent of

The <buttonGroup> node may contain the following nodes:

- <button>
- <toggleButton>

Example

In the example below, the <buttonGroup> contains three toggle buttons.

```
<buttonGroup id="BoldItalicUnderlineGroup">
  <toggleButton id="Bold" imageMso="Bold" showLabel="false">
    <!-- Control details omitted -->
  </toggleButton>
  <toggleButton id="Italic" imageMso="Italic"
showLabel="false">
    <!-- Control details omitted -->
  </toggleButton>
  <toggleButton id="Underline" imageMso="Underline"
showLabel="false">
    <!-- Control details omitted -->
  </toggleButton>
</buttonGroup>
```

In the example below, the Toggle Empties button is used to allow a Quark XML Author user to view Empties on the canvas. This approach enables a legacy Quark XML Author user to visualize the layout.

```
<buttonGroup id="ToggleEmpties">
<toggleButton id="HideEmpties"
  imageMso="SizeToControlHeight"
  showLabel="false">
<InternalClass name="ToggleEmpty"/>
</toggleButton>
</buttonGroup>
```

This **EmptiesInaccessible** element contain a value of either **true** or **false** (default). This allows the user to hide and show Empties.

An example is shown below:

```
<EmptiesInaccessible>true</ EmptiesInaccessible >
```

<checkBox>

The <checkBox> node is used to specify a check box control.

Common Attributes

The <checkBox> node uses the following “Common Attributes”:

- id or idMso (Required)
- image or imageMso
- showLabel
- size

The following common attributes for the <toggleButton> node may still be used but have been deprecated:

- keytip
- label
- screentip

To define these values, use the Resources file. See “**Error! Reference source not found.**”.

Child of

The <checkBox> node may be a child of the following nodes:

- <box>
- <group>

Example

In the example below, the native Word checkbox for viewing the Document Map is defined.

```
<checkBox idMso="ViewDocumentMap"/>
```

<command>

The <command> node defines the behavior for a standard Word command. It can globally override Word functionality or only apply when a Quark XML Author document is active.

The <command> node always uses the idMso attribute rather than the id attribute because it always points to a native Word command.

To disable a command, specify the command without a repurpose designation.

For example,

```
<command idMso="MyCommand"/>
```

To repurpose a command, specify the command with a repurpose designation.

For example,

```
<command idMso="FileSave">
    <InternalClass name="DocumentSave" nativeFormat="false"
        filter="Save2007"/>
</command>
```

The visibility of a command is specified at the individual user interface control.

Common Attributes

The <command> node uses the required attribute *idMso*.

Child of

The <command> node may be a child of the node <commands>

Parent of

The <command> node may contain the following nodes:

- <ExtensibilityMethod>
- <InternalClass>
- <ShortcutKey>

These nodes define the behavior of the control and are defined in the System Admin Guide.

Examples

In the following example, setting visible="xa" indicates that the functionality of the native Word command FileSave will be overridden whenever it is invoked from within a Quark XML Author document.

```
<command idMso="FileSave" visible="xa">
  <ShortcutKey key="S" shift="false" ctrl="true"/>
  <InternalClass name="DocumentSave" nativeFormat="false"
filter="Save2007"/>
</command>
```

In the following example, setting visible="true" indicates that the native Word command will always be available. As a result, "FileClose" would not need to be specifically defined as part of the Office Menu in the <officeMenu> node.

```
<command idMso="FileClose" visible="true"/>
```

<commands>

The <commands> node contains a collection of <command> nodes that specifies native Word functionality on a global level.

Common Attributes

The <commands> node uses no attributes:

Child of

The <commands> node may be a child of the node <customUI>

Parent of

The <commands> node may contain the node <command>

Example

```
<commands>
  <command idMso="FileSave" visible="xa">
    <ShortcutKey key="S" shift="false" ctrl="true"/>
    <InternalClass name="DocumentSave"
nativeFormat="false" filter="Save2007"/>
  </command>
  <command idMso="FileClose" visible="true"/>
  <command idMso="ApplicationOptionsDialog"/>
  <command idMso="Help" visible="xa">
    <ExtensibilityMethod id="About"/>
  </command>
</commands>
```

<contextualTabs>

Word has contextual tabs for Table Tools and Pictures Tools. The items on these tabs can break the Quark XML Author document structure and must be suppressed. Add the XML snippet below after the <tabs> node to suppress these contextual tabs when a Quark XML Author document is the active document.

```
<contextualTabs>
    <tabSet idMso="TabSetTableTools" visible="Word"/>
    <tabSet idMso="TabSetPictureToolsClassic" visible="Word"/>
</contextualTabs>
```

<customUI>

The <customUI> node contains all of the command and ribbon definitions.

Common Attributes

The <customUI> node must have the following structure. The xmlns attribute should be the first attribute.

```
<customUI xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="OnLoad" loadImage="LoadImage">
```

Child of

The <customUI> node is the child of <Ribbon>

Parent of

The <customUI> contains the following nodes:

- <commands>
- <ribbon>

Example

```
<customUI
xmlns="http://schemas.microsoft.com/office/2006/01/customui"
onLoad="OnLoad" loadImage="LoadImage">
    <commands>
        <!-- one or more command nodes -->
    </commands>
    <ribbon>
        <!-- all of the nuts and bolts of the office
menu and ribbon tabs -->
    </ribbon>
</customUI>
```

<dropDown>

The <dropDown> node defines the following attributes:

Common Attributes

The <dropDown> node uses the following “Common Attributes”:

- id (Required)
- showLabel
- sizeString

The following common attributes for the <dropDown> node may still be used but have been deprecated:

- label

To define these values, use the Resources file. See “**Error! Reference source not found.**”.

Child of

The <dropDown> node may be a child of the following nodes:

- <box>
- <group>

Parent of

The <dropDown> node may contain the following nodes:

- <InternalClass>

These nodes define the behavior of the control and are defined in the System Admin Guide.

Example

```
<dropDown id="StylesListXA" showLabel="false"
sizeString="XXXXXXXXXXXXXXXXXXXXXXX">
  <InternalClass name="StyleHandler"/>
</dropDown>
```

<group>

Controls on each ribbon tab are organized into logical groups. In Figure 4-3, the Quark XML Author Home tab is shown with three groups: Clipboard, Styles, and Editing.

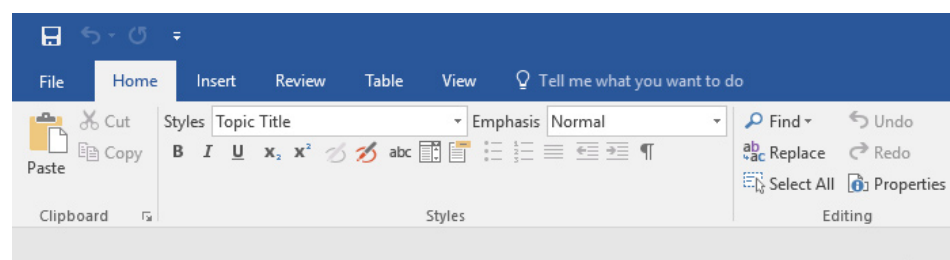


Figure 4-3: Home tab with three groups

The <group> node defines each group on a tab and contains all the other controls.

Common Attributes

The <group> node uses the following “Common Attributes”:

- id or idMso (required)
- insertAfterMso (optional)
- insertBeforeMso (optional)
- visible (optional)

The following common attributes for the <group> node may still be used but have been deprecated:

- label

To define these values, use the Resources file. See “**Error! Reference source not found.**”.

Child of

The <group> node may be a child of the following element:

- <tab>

Parent of

The group node may contain any of the following control nodes.

- <box>
- <button>
- <command>
- <menu>
- <splitButton>
- <toggleButton>

Each of these nodes is discussed in Section 4.2.

Example

The example below shows a group containing three buttons.

```
<group id="CopyPasteXA">
  <button id="PasteXA" imageMso="Paste" size="large">
    <!-- Control details omitted -->
  </button>
  <button id="CutXA" imageMso="Cut">
    <!-- Control details omitted -->
  </button>
  <button id="CopyXA" imageMso="Copy">
    <!-- Control details omitted -->
  </button>
</group>
```

<menu>

The <menu> node defines a menu with the Office menu, a ribbon tab, or a split button.

To use a native Word menu, specify a value for the idMso attribute.

To define a custom menu, specify a value for the id attribute.

Common Attributes

The <menu> node uses the following “Common Attributes”:

- id or idMso (required)
- image or imageMso
- insertBeforeMso

- `itemSize`
- `size`
- `visible`

The following common attributes for the `<menu>` node may still be used but have been deprecated:

- `keytip`
- `label`
- `screentip`
- `title`

To define these values, use the Resources file. See “**Error! Reference source not found.**”.

Unique Attributes

The `<menu>` node may also use the unique attribute described in Table 4-1.

Table 4-2: `<menu>` Attributes

Attribute	Value	Description
<code>itemSize</code>	<code>large</code>	Display size of the menu item. Defaults to small.

Child of

The `<menu>` node may be a child of the following nodes:

- `<box>`
- `<group>`
- `<officeMenu>`
- `<splitButton>`

Parent of

The `<menu>` node may contain the following nodes:

- `<button>`
- `<toggleButton>`

Example

The example below presents a menu with two buttons.

```
<menu id="FileSendXAMenu" image="export-to-xml-icon.ico"
insertBeforeMso="FilePrepareMenu" itemSize="large" visible="XA">
  <button id="FileExportToIE" imageMso="WebPagePreview">
    <!-- Control details omitted -->
  </button>
  <button id="FileSendAsXmlAttachment"
imageMso="FileSendAsAttachment">
    <!-- Control details omitted -->
  </button>
```



```
</  
menu  
>
```

<menuSeparator>

The <menuSeparator> node forces a separation line to appear between menu items.

Common Attributes

The <menuSeparator> node uses the following “Common Attributes”:

- id (Required)

Child of

The <menuSeparator> node may be a child of the following nodes:

- <menu>
- <officeMenu>

Parent of

The <menuSeparator> node may not contain any child nodes.

Example

In the example below, the highlighted node causes a separation line to appear before the FileOpen button.

```
<menu id="XpressAuthor" image="QuarkXMLAuthor.ico"  
insertBeforeMso="FileNew" itemSize="large" visible="true">  
  <button id="NewTopic" image="topic-icon.ico"  
visible="true">  
    <!-- Control details omitted -->  
  </button>  
  <button id="NewTask" image="task-icon.ico" visible="true">  
    <!-- Control details omitted -->  
  </button>  
  <button id="NewConcept" image="concept-icon.ico"  
visible="true">  
    <!-- Control details omitted -->  
  </button>  
  <button id="NewReference" image="reference-icon.ico"  
visible="true">  
    <!-- Control details omitted -->  
  </button>  
  <menuSeparator id="FileOpenSeparator"/>  
  <button id="FileOpen" image="open-xa-icon.ico"  
visible="true">  
    <!-- Control details omitted -->  
  </button>  
</menu>
```

Word Backstage View

Just as we had the ability to customize the Word 2007 Office menu, we have the ability to customize the Word Backstage view (File menu).

The following overview is an excerpt from the “Introduction to the Office 2010 Backstage View for Developers” article ©Microsoft:

“In the 2007 release of the Microsoft Office system, file-level functionality was accessible on the Office menu that was available by clicking the Office button.

“In Microsoft Office 2010 and 2013, the Office button is replaced by a File tab. Clicking the File tab takes you to the Microsoft Office Backstage view.

“Backstage view is fully extensible by developers, permitting organizations to customize the user interface (UI) to suit their own needs. And best of all, the Backstage UI is customizable by using the same files, callbacks, and many of the controls used in the Ribbon. This means that developers already familiar with customizing the Ribbon UI can use those same skills to create a Backstage UI targeted at the needs of their organization.”

Additional information:

[MSDN Library 691833](#)

The <backstage> node defines the items that appear on the Word File menu. The Word File menu may have any combination of menus, buttons, and split buttons. Quark recommends that Quark XML Author configurations mimic, as much as possible, the native Word interface. Thus the Word File menu should be defined to present the same menu items that are present in standard Word, with the Quark XML Author items at the top.

This solution is taken from the VSTO2010 documentation, however unlike the Microsoft samples, Quark XML Author only uses the in-document UI customization.

The following is a fragment from AppConfig file for Backstage 2010 that illustrates where <backstage> resides in the <Ribbon>\<customUI>structure.

```
<Ribbon>
  <customUI
    xmlns="http://schemas.microsoft.com/office/2006/01/customui"
    onLoad="OnLoad" loadImage="LoadImage">
    <commands>
      <!-- one or more command nodes -->
    </commands>
    <ribbon>
      <!-- nuts and bolts of the office menu and ribbon
tabs -->
      <ribbon startFromScratch="false">
        <officeMenu>
        </officeMenu>
        <tabs>
        </tabs>
        <contextualTabs>
        </contextualTabs>
      </ribbon>
    <!-- nuts and bolts of the backstage view -->
    <backstage>
    </backstage>
  </customUI>
</Ribbon>
```

The following is a fragment from AppConfig file for Backstage 2013

that illustrates where <backstage> resides in the <Ribbon>\<customUI>structure.

```
<Ribbon>
  <customUI
    xmlns="http://schemas.microsoft.com/office/2006/01/customui"
    onLoad="OnLoad" loadImage="LoadImage">
```

CONFIGURATION: RIBBON, OFFICE MENU, AND BACKSTAGE VIEW

```
        <commands>
            <!-- one or more command nodes -->
        </commands>
        <ribbon>
            <!-- nuts and bolts of the office menu and ribbon
tabs -->
            <ribbon startFromScratch="false">
                <officeMenu>
                </officeMenu>
                <tabs>
                </tabs>
                <contextualTabs>
                </contextualTabs>
            </ribbon>
        <!-- nuts and bolts of the backstage view -->
        <backstage2013>
            </backstage2013>
    </customUI>
</Ribbon>
```

The following is an example <backstage> element, with screen shots of the user interface that it would create. Note: The schema for <backstage> elements is not the same as ribbon elements. For example, a ribbon group will not work for a Backstage group.

```
<backstage>
    <button id="OpenXMLAuthor" getLabel="GetLabel" image="open-
xa-icon.ico" insertAfterMso="FileOpen" visible="true">
        <InternalClass name="DocumentOpen" filter="OpenXA2007QA" />
    </button>
    <button id="SaveAsXMLAuthor" getLabel="GetLabel"
imageMso="FileSaveAs" insertBeforeMso="FileOpen" visible="XA">
        <InternalClass name="DocumentSaveAs" nativeFormat="false"
filter="Save2007QA" />
        <ShortcutKey key="F12" shift="false" ctrl="false" />
    </button>
    <button id="XAViewAsXML" getLabel="GetLabel" image="export-
to-xml-icon.ico" insertBeforeMso="FileClose" visible="XA">
        <InternalClass name="PreviewRendition" xslt="qa\xml-pretty-
print.xsl" appPath="IExplore.exe" />
    </button>
    <tab id="FileNewXAMenu" insertBeforeMso="TabNew" title="XML
Author Templates" getLabel="GetLabel" visible="true"
columnWidthPercent="40">
        <firstColumn>
            <taskGroup id="XATemplateGroupQA"
allowedTaskSizes="largeMediumSmall">
                <!-- QA Templates -->
                <category id="QA" label="QA Templates">
                    <task id="CreateNewQATask" label="QA"
imageMso="TemplatesMenu" description="Create an XML Author - QA
template" visible="true">
                        <InternalClass name="DocumentNew"
templateXml="QA\blank-template.xml" />
                    </task>
                </category>
                <!-- DITA Templates -->
                <category id="XATemplateGroupDita" label="DITA
Templates">
                    <task id="CreateXADitaMap" label="Map"
imageMso="MeetingsToolAppointmentAgenda" description="Create an
XML Author - DITA Map document" visible="true">
                        <InternalClass name="DocumentNew"
templateXml="DITA\template-map.xml" />
                    </task>
                    <task id="CreateXADitaTopic" label="Topic"
```

```

description="Create an XML Author - DITA Topic document"
image="topic-icon.ico" visible="true">
    <InternalClass name="DocumentNew"
templateXml="DITA/template-topic.xml" />
    </task>
    <task id="CreateXADitaTask" label="Task"
description="Create an XML Author - DITA Task document"
image="task-icon.ico" visible="true">
    <InternalClass name="DocumentNew"
templateXml="DITA/template-task.xml" />
    </task>
    <task id="CreateXADitaConcept" label="Concept"
description="Create an XML Author - DITA Concept document"
image="concept-icon.ico" visible="true">
    <InternalClass name="DocumentNew"
templateXml="DITA/template-concept.xml" />
    </task>
    <task id="CreateXADitaReference" label="Reference"
description="Create an XML Author - DITA Reference document"
image="reference-icon.ico" visible="true">
    <InternalClass name="DocumentNew"
templateXml="DITA/template-reference.xml" />
    </task>
</category>
</taskGroup>
</firstColumn>
<!-- DO NOT EDIT THIS COLUMN UNDER ANY CIRCUMSTANCES!!!!!!
-->
<secondColumn>
    <group id="XAAboutBannerGroup">
        <topItems>
            <imageControl id="XALogo" getImage="GetXABannerImage"
visible="true" altText="Quark XMLAuthor for Microsoft Word"/>
        </topItems>
    </group>
    <group id="XABlurbGroup">
        <topItems>
            <labelControl id="XABlurb" getLabel="GetLabel"/>
        </topItems>
    </group>
    <group id="XAVersionInfoGroup" getLabel="GetLabel">
        <topItems>
            <labelControl id="XAVersionLable"
getLabel="GetVersionString"/>
            <labelControl id="XACopyrightLabel"
getLabel="GetLabel"/>
            <hyperlink id="XAEulaLink" getLabel="GetLabel"
onAction="OnEulaHyperlinkClick"/>
        </topItems>
    </group>
</secondColumn>
<!-- DO NOT EDIT ABOVE!!!!!! -->
</tab>
<tab idMso="TabPrint">
    <firstColumn>
        <group id="XAGroupPrintSettings"
insertBeforeMso="GroupPrintSettings" getLabel="GetLabel"
style="error" visible="XA">
            <primaryItem>
                <button id="XAPageSetup" imageMso="PageSetupDialog"
getLabel="GetLabel" visible="XA">
                    <InternalClass name="PageLayout" />
                </button>
            </primaryItem>
        </topItems>
        <labelControl id="XAPageSetupWarning"

```

CONFIGURATION: RIBBON, OFFICE MENU, AND BACKSTAGE VIEW

```
getLabel="GetLabel"/>
    </topItems>
  </group>
</firstColumn>
</tab>
<tab id="XAShare" getLabel="GetLabel"
insertBeforeMso="ApplicationOptionsDialog" visible="XA">
  <firstColumn>
    <taskFormGroup id="XAGroupShare" getLabel="GetLabel"
allowedTaskSizes="mediumSmall">
      <category id="XAShareCategory">
        <task id="XASendUsingEmailTask"
imageMso="SendAsAttachmentToMailRecipient" getLabel="GetLabel">
          <group id="XASendUsingEmailGroup"
getLabel="GetLabel" visible="XA">
            <topItems>
              <layoutContainer id="SendAsXMLLayout"
layoutChildren="horizontal">
                <button id="SendAsXMLAttachementButton"
image="send-xa-icon.ico" getLabel="GetLabel" style="large">
                  <InternalClass name="EmailRendition"
rendition="xpressFormat" />
                </button>
                <layoutContainer
id="SendAsXMLDescriptionLayout" layoutChildren="vertical">
                  <labelControl id="SendAsXmlDescription1"
getLabel="GetLabel"/>
                  <layoutContainer
id="SendAsXmlDescriptionLayout2" layoutChildren="horizontal">
                    <imageControl id="SendAsXMLBullet1Icon"
imageMso="ColorSilver"/>
                    <labelControl id="SendAsXmlBullet1"
getLabel="GetLabel"/>
                  </layoutContainer>
                  <layoutContainer
id="SendAsXMLBullet2Layout" layoutChildren="horizontal">
                    <imageControl id="SendAsXMLBullet2Icon"
imageMso="ColorSilver"/>
                    <labelControl id="SendAsXmlBullet2"
getLabel="GetLabel"/>
                  </layoutContainer>
                  <layoutContainer
id="SendAsXMLBullet3Layout" layoutChildren="horizontal">
                    <imageControl id="SendAsXMLBullet3Icon"
imageMso="ColorSilver"/>
                    <labelControl id="SendAsXmlBullet3"
getLabel="GetLabel"/>
                  </layoutContainer>
                </layoutContainer>
              </layoutContainer>
            </topItems>
          </group>
          <group id="XASendAsWordUsingEmailGroup" label=" "
visible="XA">
            <topItems>
              <layoutContainer id="XASendAsWordLayout"
layoutChildren="horizontal">
                <button id="XASendAsWordAttachmentButton"
imageMso="SendAsAttachmentToMailRecipient" label="Send as Word
Attachment" style="large">
                  <InternalClass name="EmailRendition"
rendition="nativeFormat" />
                </button>
                <layoutContainer
id="SendAsWordDescriptionLayout" layoutChildren="vertical">
                  <labelControl id="SendAsWordDescription"
```

```

getLabel="GetLabel"/>
        <layoutContainer
id="SendAsWordBullet1Layout" layoutChildren="horizontal">
            <imageControl id="SendAsWordBullet1Icon"
imageMso="ColorSilver"/>
            <labelControl id="SendAsWordBullet1"
getLabel="GetLabel"/>
        </layoutContainer>
        <layoutContainer
id="SendAsWordBullet2Layout" layoutChildren="horizontal">
            <imageControl id="SendAsWordBullet2Icon"
imageMso="ColorSilver"/>
            <labelControl id="SendAsWordBullet2"
getLabel="GetLabel"/>
        </layoutContainer>
    </layoutContainer>
</topItems>
</group>
</task>
<task id="XAViewXMLTask" getLabel="GetLabel"
image="export-to-xml-icon.ico">
    <group id="XAViewXMLGroup" getLabel="GetLabel">
        <topItems>
            <layoutContainer id="XAViewXMLLayout"
layoutChildren="horizontal">
                <button id="XAViewAsXML2" getLabel="GetLabel"
image="export-to-xml-icon.ico" style="large">
                    <InternalClass name="PreviewRendition"
xslt="qa\xml-pretty-print.xsl" appPath="IExplore.exe" />
                </button>
            </layoutContainer>
        </topItems>
    </group>
</task>
</category>
</taskFormGroup>
</firstColumn>
</tab>
</backstage>

```

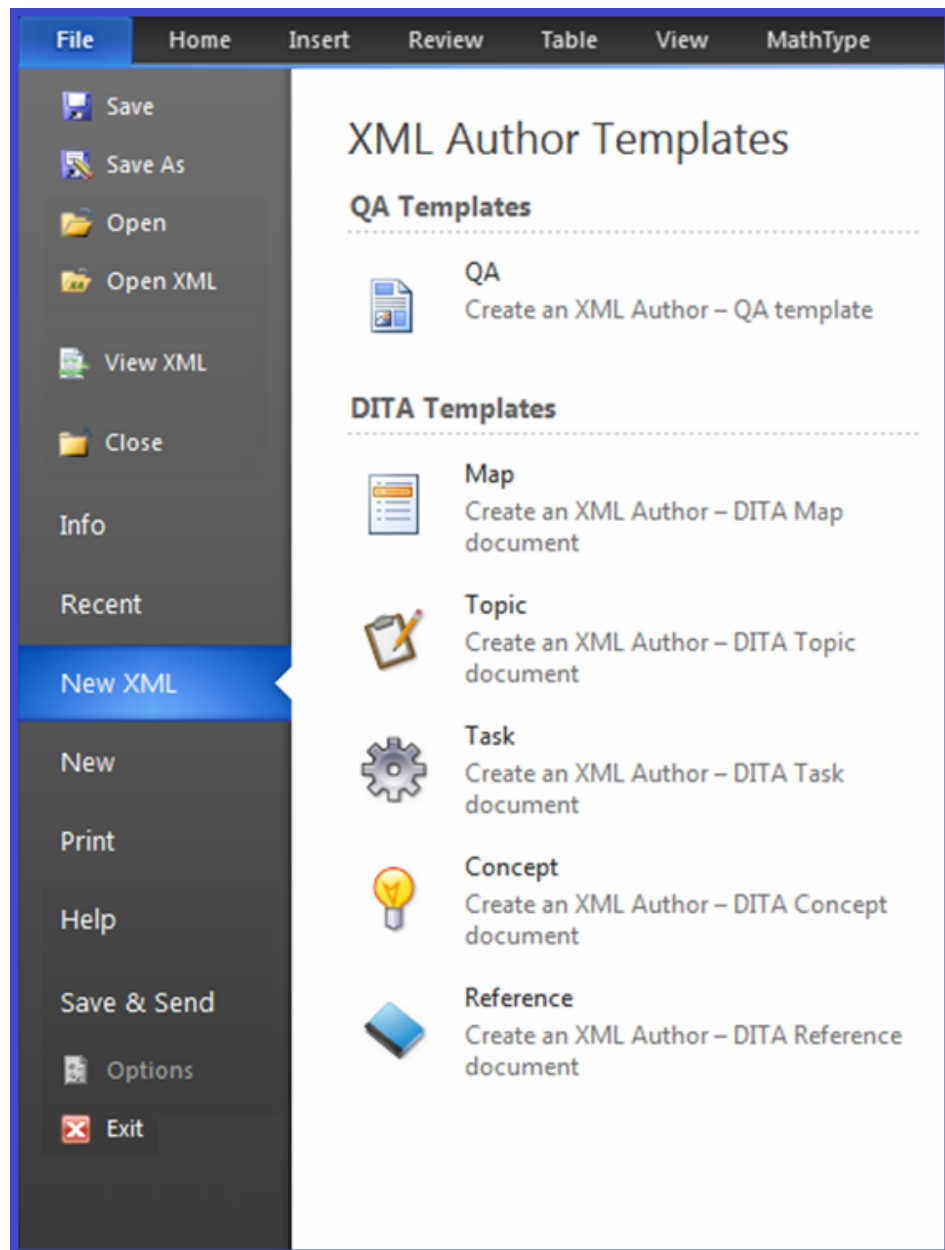


Figure 4-4: File > XML Author submenu

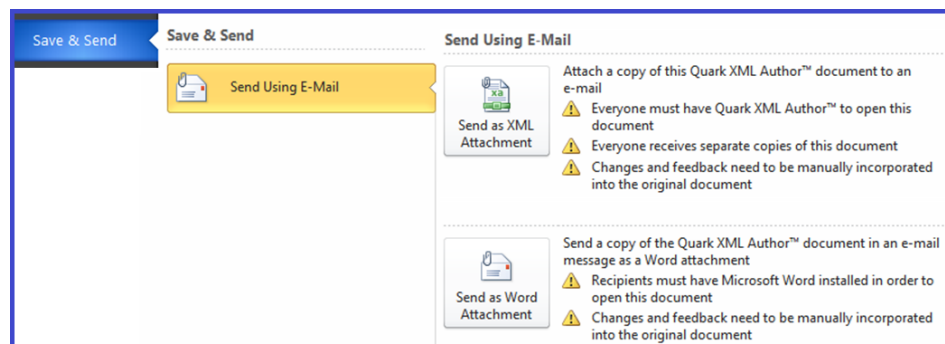


Figure 4-5: File > Save & Send submenu

Hiding menu items in Word 2010 and later

Hiding Items in the Backstage View

This section describes how to hide standard Word menu items in the Backstage view. This is typically done to hide Word features that do not apply to an XML authoring session or that might corrupt or damage an XML Author document.

For controls on the Backstage view, XML Author only supports the following:

- visible in all Word and XML Author documents
- not visible in all Word and XML Author documents

For more information on specifying visibility, see “Common Attributes”.

Template Stub

This is accomplished using a template stub.

In the following example template stub, the TabInfo and TabShare items are hidden by setting the visible attribute to false.

```
<?xml version="1.0" encoding="utf-8"?>
<customUI
xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <backstage>
        <tab idMso="TabInfo" visible="false" />
        <tab idMso="TabShare" visible="false" />
    </backstage>
</customUI>
```

For Word 2010 or 2013, the template stub is named the following:

customUI14.xml

Given that a Word template is essentially a zip file. The following process adds a folder to template (zip) file and then adds the template stub to that new folder.

To incorporate a custom template stub:

1. Launch Windows Explorer.
2. Navigate to the folder where you installed Quark XML Author. This is typically the following folder:

C:\Program Files\Quark\XML Author\

1. Then, navigate to the subfolder that contains the Word Templates for the schema you are authoring in. For example, the template for the Quality Assurance configuration is located in the QA subfolder:

C:\Program Files\Quark\XML Author\QA\

1. Select the desired template. For example, the QA.dotx template file.
2. Rename the template file so that it has an extension of .ZIP.
3. Add a folder name “customUI” to this zip file.

CONFIGURATION: RIBBON, OFFICE MENU, AND BACKSTAGE VIEW

4. Add your customUI14.xml file to the "customUI" folder.
5. Rename the template file back to its original filename by removing the .ZIP extension.

The specified items will be hidden from the Word 2010/2013 Backstage View the next time you open an XML Author document based on this template.

Programmatically closing the Backstage View

The following configuration example shows how to programmatically close the Backstage View. "Set the **isDefinitive** attribute **true** to close the Backstage view and return to the workbook when you click the **Save & Close** button."

```
<button id="DocumentAttributes" imageMso="FileProperties"
insertBeforeMso="ApplicationOptionsDialog" visible="XA"
isDefinitive="true">
  <InternalClass name="DocumentAttributes"/>
</button>
```

Related information:

[Adding Custom Commands and Changing the Visibility of Controls in the Office 2010 Backstage View](#)

Quick Access Toolbar (QAT)

XML Author has the following limitations and considerations in regard to the QAT. The user can add any Word command to the QAT.

XML Author cannot hide commands on the QAT. See "Common Attributes". XML Author can repurpose/override all commands on the QAT except Undo. See "Repurposing commands" and "Undo". XML Author can disable any command on the QAT. See "Disabling commands".

<ribbon>

The <ribbon> node, not to be confused with its ancestor, <Ribbon>, contains all of the information for the appearance of Word's Office menu and the ribbon.

Common Attributes

The <ribbon> node uses one attribute, defined in Table 4-3.

Table 4-3: <ribbon> Attributes

Attribute	Value	Description
startFromScratch	true false	Set to true if the entire built-in Word ribbon structure is to be discarded. Set to false if the existing ribbon is to be modified rather than built from scratch. Defaults to false . Quark recommends setting this

Attribute	Value	Description
		attribute to false.

Child of

The <ribbon> is the child of <customUI>.

Parent of

The <ribbon> contain the following nodes:

- <officeMenu>
- <tabs>
- <contextualTabs>

Example

The structure of the node is shown below.

```
<ribbon startFromScratch="false">
  <officeMenu/>
  <tabs/>
  <contextualTabs/>
</ribbon>
```

<separator>

The <separator> node specifies that a space will appear between two controls in the ribbon. This is typically used to create a space between two button groups.

Common Attributes

The <separator> node uses the following “Common Attributes”:

- id (Required)

Child of

The <separator> node may be a child of the following nodes:

- <group>

Parent of

The <separator> node may not contain any child nodes.

Example

In the example below, the highlighted node creates a space between the box and menu.

```
<box id="WindowButtonBox" boxStyle="vertical">
  <!-- Control details omitted -->
</box>
<separator id="GroupWindowSeparator2"/>
<menu idMso="WindowSwitchWindowsMenuWord" size="large">
  <!-- Control details omitted -->
</menu>
```

<splitButton>

The <splitButton> node defines a Split Button. A split button can be clicked like a regular button, but also makes available to the user a menu of one or more additional commands if the drop-down arrow is clicked. A child <button> node indicates the button-click behavior. A child <menu> node contains the additional commands that may be performed.

To use a native Word split button, specify a value for the idMso attribute.

To define a custom split button, specify a value for the id attribute.

Common Attributes

The <splitButton> node uses the following “Common Attributes”:

- id or idMso (Required)
- size
- visible

Child of

The <splitButton> node may be a child of the following nodes:

- <box>
- <buttonBox>
- <group>
- <officeMenu>

Parent of

The <splitButton> node may contain the following nodes:

- <button>
- <menu>
- <toggleButton>

Example

In the example below, the split button acts as a Selection button when clicked, but the user can also access a menu with a different option.

```
<splitButton id="SelectXA">
  <button id="SelectButtonXA" imageMso="SelectMenu"/>
  <menu id="SelectMenuXA">
    <button idMso="SelectAll"/>
  </menu>
</splitButton>
```

<tab>

Each <tab> node contains one or more <group> nodes, which in turn contain the array of control definitions that are presented on the ribbon tab.

Common Attributes

The <tab> node uses the following “Common Attributes”:

- id or idMso (required)
- visible (required)

The following common attributes for the <tab> node may still be used but have been deprecated:

- keytip
- label

To define these values, use the Resources file. See “**Error! Reference source not found.**”.

Child of

The <tab> node may be a child of the following nodes:

- <tabs>

Parent of

The <tab> node may contain the following nodes:

- <group>

Example

A native Word tab is defined by identifying it with the idMso attribute. The visible attribute must be set to **Word** so that it will not be shown when a Quark XML Author document is active. The example below is for the native Word **Home** tab.

```
<tab idMso="TabHome" visible="Word"/>
```

Native Word tabs may have child items in order to disable or enable specific items while a Quark XML Author document is the active document. In the example below, the Word Print Preview tab has been redefined so that the second group only displays when a Quark XML Author document is not active.

```
<tab idMso="TabPrintPreview" visible="true">
  <group idMso="GroupPrintPreviewPrint" visible="true"/>
  <group idMso="GroupPrintPreviewPageSetup" visible="Word"/>
  <group idMso="GroupZoom" visible="true"/>
  <group idMso="GroupPrintPreviewPreview" visible="true"/>
</tab>
```

Tabs specific to Quark XML Author use the id attribute. The visible attribute must be set to **XA** so that they will *only* be shown when a Quark XML Author document is active. The example below shows the structure of a Quark XML Author tab.

```
<tab id="TabXAHome" visible="XA">
  <group id="CopyPasteXA">
    <!-- Control definitions omitted -->
  </group>
  <!-- Additional groups omitted -->
</tab>
```

Appearance order for the tabs matches the definition order.

<tabs>

The <tabs> node contains definitions for all of the ribbon tabs available in the application.

Common Attributes

The <tabs> node uses no attributes:

Child of

The <tabs> node may be a child of the following nodes:

- <ribbon>

Parent of

The <tabs> node may contain the following nodes:

- <tab>

Example

Each ribbon tab is represented by a <tab> node within <tabs>. An example is shown below.

```
<tabs>
  <tab id="TabXAHome" visible="XA">
    <!-- Control definitions ommitted -->
  </tab>
  <tab id="TabXAInsert" visible="XA">
    <!-- Control definitions ommitted -->
  </tab>
  <tab id="TabXAReview" visible="XA">
    <!-- Control definitions ommitted -->
  </tab>
  <tab id="TabXATable" visible="XA">
    <!-- Control definitions ommitted -->
  </tab>
  <tab id="TabXAView" visible="XA">
    <!-- Control definitions ommitted -->
  </tab>
  <tab idMso="TabHome" visible="Word"/>
  <tab idMso="TabInsert" visible="Word"/>
  <tab idMso="TabPageLayoutWord" visible="Word"/>
  <tab idMso="TabReferences" visible="Word"/>
  <tab idMso="TabMailings" visible="Word"/>
  <tab idMso="TabReviewWord" visible="Word"/>
  <tab idMso="TabView" visible="Word"/>
  <tab idMso="TabDeveloper" visible="Word"/>
  <tab idMso="TabAddIns" visible="Word"/>
  <tab idMso="TabPrintPreview">
    <!-- Control definitions ommitted -->
  </tab>
  <tab idMso="TabBlogInsert" visible="Word"/>
  <tab idMso="TabBlogPost" visible="Word"/>
</tabs>
```

<toggleButton>

The <toggleButton> node defines a toggle button.

To use a native Word toggle button, specify a value for the idMso attribute.

To define a custom toggle button, specify a value for the id attribute.

Common Attributes

The <toggleButton> node uses the following “Common Attributes”:

- id or idMso (Required)
- image or imageMso

- showLabel
- size

The following common attributes for the <toggleButton> node may still be used but have been deprecated:

- keytip
- label
- description
- screentip

To define these values, use the Resources file. See “**Error! Reference source not found.**”.

Unique Attributes

The <toggleButton> node uses one required attribute, defined in Table 4-4.

Table 4-4: <toggleButton> Attributes

Attribute	Value	Description
getPressed	GetPressed	Defaults to no value. When set, allows internal classes to set the Pressed state of a ribbon button. Currently used by the following internal event handlers: ChangeToList, ChangeToPara, InlineAttributes, and ToggleCommentsPane. This is always set programmatically and is not specified by a configuration engineer.

Child of

The <toggleButton> node may be a child of the following nodes:

- <box>
- <buttonGroup>
- <group>
- <menu>
- <splitButton>

Parent of

The <toggleButton> node may contain the following nodes:

- <InternalClass>
- <ExtensibilityMethod>
- <ShortcutKey>

These nodes define the behavior of the control.

Example

A sample toggleButton is shown below.

```
<toggleButton id="HideEmpties" imageMso="LineSpacing" showLabel="false">
  <!-- Control details ommitted -->
</toggleButton>
```

Common Attributes

Table 4-5: Common Attributes

Attribute	Value	Description
description	String displaying the control description.	Deprecated. Specify the control description in "Resource Files".
id	String to identify the control.	Must be unique within the configuration. See "Unique Ids".
idMso	Name of a native Word control.	Word control names can be found in WordRibbonControls.xlsx.
image	Name of the image to display as an icon for the control.	Icons are internal to Quark XML Author (Xpress.dll).
imageMso	Name of a native Word control; the icon from the control is used for the Quark XML Author control.	Word control names can be found in WordRibbonControls.xlsx.
insertAfterMso	Name of a native Word control.	The control will be placed after the named Word control in the menu or group.
insertBeforeMso	Name of a native Word control.	The control will be placed before the named Word control in the menu or group.
label	String to be displayed as the control's label.	Deprecated. Specify the control label in the Resources file. See "Error! Reference source not found.".
keytip	1-3 alphanumeric characters to be displayed as the control's keytip.	Deprecated. Specify the control screen tip in the Resources file. See "Error! Reference source not found.".
screen tip	String to be displayed as the control's screen tip.	Deprecated. Specify the control's screen tip in the Resources file. See "Error! Reference source not found.".

Attribute	Value	Description
showLabel	false	Defaults to true . If set to false , hides the control's label.
size	large	Display size of the button, split button, or toggle button. Defaults to small.
sizeString	One or more X characters	Specifies the size of the drop-down control. For example, a value of XXXX indicates a width of four characters.
title	String to be displayed as the control's title.	Deprecated. Specify the control's title in the Resources file. See "Error! Reference source not found." .
visible	false true Word xa	<p>Determines when the control is available. Settings may be combined using the pipe character.</p> <p>false: control is not visible in all Word and XML Author documents.</p> <p>true: control is visible in all Word and XML Author documents.</p> <p>Word: control is only visible in Word documents.</p> <p>xa: control is only visible in XML Author documents.</p> <p>OLEWordDocument: control is only visible in embedded Word documents.</p>

Unique Ids

The Ribbon configuration does not support duplicate IDs, therefore a feature that is present more than once in the ribbon must have a unique ID for each occurrence of that feature.

For example, in the DITA configuration, the Properties button exists on the Home and View ribbon. Both occurrences of the button call the same Extensibility Method "DocumentProperties".

```
<button id="DocumentPropertiesXA" label="Properties" keytip="PR"
imageMso="FileProperties">
    <ExtensibilityMethod id="DocumentProperties" />
</button>
```


CONFIGURATION: RIBBON, OFFICE MENU, AND BACKSTAGE VIEW

```
<button id="ViewDocumentPropertiesXA" imageMso="FileProperties"
label="Properties" keytip="DP">
    <ExtensibilityMethod id="DocumentProperties" />
</button>
```

Undo

The user must not be allowed to invoke the Undo feature in Microsoft Word because:

Standard options/techniques for preventing a user from invoking a Word command include hiding it, disabling it, and repurposing or overriding it. However, Undo is a unique and special case which must be handled differently than other Word commands.

On the QAT, the only Word command that cannot be repurposed/overridden is Undo.

QAT is the only UI in which we are unable to hide Word commands. Now we can disable a Word idMso command across all user interfaces in Word including: Ribbon, Toolbar, Menus, and Quick Access Toolbar.

Printing

Several of the Word page setup features are not XML aware and therefore should be excluded from the user interface. If that is not possible, then the user must be warned not to use those features.

Word 2010 and later

In the Backstage View for Print, the Quark XML Author Page Setup feature is provided. This feature is used to specify page orientation and layout settings specific to XML Author documents. In that view, Word's Orientation button and Page Setup link are not XML aware and therefore should not be used. The user is provided that warning

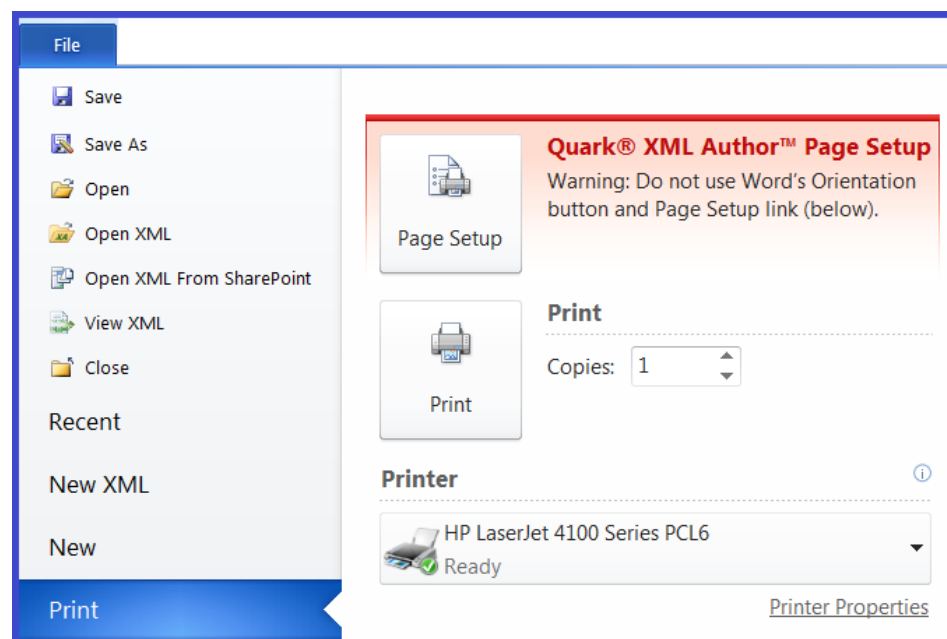


Figure 4-6: Backstage View for Print

Hiding/Disabling Ribbon Tabs for Third-Party Software

The Quark XML Author add-in cannot influence the position or visibility of the ribbon tabs of a third-party add-in. Therefore, disabling the add-in is the only way to remove its ribbon tab when Quark XML Author is in focus/active.

This is accomplished using `DisableTemplateAddins`. See “`DisableTemplateAddins`”.

Application Configuration: Internal Classes

Internal Classes are functions that are built into Quark XML Author. In many cases, they replace similar native Word features, such as Insert Table.

The InternalClass element's attributes contain the name of the application class that will handle the operation when the user selects the menu item/clicks the button/invokes the shortcut key associated with the command. The name attribute is required. Depending on what class is used, there may be other required and optional attributes. This chapter defines the internal classes supported by Quark XML Author and the additional parameters, if any, that can be used with them. The example below shows how DocumentOpen can be implemented.

```
<InternalClass name="DocumentOpen" filter="Word Documents (*.doc)|*.doc|Quark XML Author Documents (*.xml)|*.xml"/>
```

As you can see, DocumentOpen requires an additional parameter, which is specified as the InternalClass element's filter attribute.

AcceptRevision

Accepts the currently selected document revision or, if the **acceptAll** parameter is present and set to **true**, accepts all document revisions. If neither parameter is present, any revision(s) in the current selection will be accepted.

Table 5-1: AcceptRevision Parameters

Parameter	Required	Definition
acceptAll	no	Boolean. Set to true to accept all revisions in the document. Defaults to false .
keyCode	no	Integer. Specifies the keyCode for a specific document node. If keyCode is a valid node, AcceptRevision will accept revision(s) for the node specified.

AssignAttribute

Assigns a value to a document attribute or document element attribute. If the attribute is not found on the current document, the nearest ancestor element which contains this attribute is located and the attribute value is assigned to that element (if found).

Table 5-2: AssignAttribute Parameters

Parameter	Required	Definition
attributeName	yes	String. The document attribute or document element attribute, as specified in the document class schema, to which a value will be assigned.
attributeValue	no	String. Specifies the value that will be assigned to the attribute named by the attributeName parameter.
generateUndo	no	Specifies whether or not an undo event should be generated for this attribute assignment. Defaults to false .
namespace	no	String. Specifies the namespace of the attribute named as the value of attributeName.
toggleGroup	yes	String. Specifies the name of the toggle group to which the menu item containing the InternalClass belongs.
xpath	no	String. If an xpath for a particular element is provided, it is used as the starting point for finding the containing attribute specified by the attributeName. If it is not provided the XomCurrentNode is used.

ChangeToList

Changes the selected element to a list item. If the selection contains multiple elements, changes all of them to list items. There are some restrictions depending on the selection of elements in the user interface:

- Top-level selection elements have to be at the same level and of same parent. In other words, the user cannot select list items of different lists, nor can the user select a para type element with depth=x and another non-descendant para with depth <> x.

- The top-level selection can ONLY include para or list type selections AND cannot include both para type and list type selections.
- The selection for list type elements must include the first or the last list item, or both.
- Top level selections must not be defined with minOccurs=1, maxOccurs=1/1+.
- The presence of any required child elements (for list type elements, both required children under list container as well as required children under the para type child) disqualifies them from being changeType candidates.
- Both of these element types must appear as optional unbounded (minOccurs=0, maxOccurs=unbounded) children under their parent definitions.
- There can be any **optional** descendant structure amongst the para/list types. In that case, if the current selection does have a descendant structure for each of the top-level para/list type elements, each of those instance structures would have to be “equivalent” to the schema descendant structure of the corresponding para or list type for a target to be a valid change type

Other specific behavior depends on the values of the parameters.

Table 5-3: ChangeToList Parameters

Parameter	Required	Definition
merge	no	Boolean. Set to true to force new list items to merge with the adjacent list (if one exists) upon conversion rather than being inserted as the sole child of a new list. Defaults to false .
target	yes	String. Comma-delimited list of list elements that may be used. Specifies the XML name of the list element to use as the parent of the changed element. Multiple levels of list can be specified by providing a comma separated list of list elements. The exact list element that will be used is dependent on the XAS and the location of the element to be changed. In the example below, the selected element could be changed to an item in a bullet list or a sub-list. <pre><InternalClass name="ChangeToList" target="bulletul,subbulletul" merge="true"/></pre>

Parameter	Required	Definition
		<p>Note that there is no space after each comma.</p> <p>Technically, a “list” type element is a specialized element identified by the following schema pattern in the schema: Invisible Section with a 1+ sequence of (visible) para type element children. Although list elements are discussed here, the target element can be any element as long as it follows the same pattern.</p>

ChangeToPara

Changes the selected element (usually a list item) to a paragraph. If the selection contains multiple list items, all are converted to separate paragraphs.

Table 5-4: ChangeToPara Parameters

Parameter	Required	Definition
target	yes	<p>String. Specifies the XML name of the paragraph element. In the example below, the paragraph element is <p>.</p> <pre><InternalClass name="ChangeToPara" target="p"/></pre> <p>A paragraph type element is identified in the schema by the presence of the <para> tag in the schema. Presence of the excludeFromChangeToContext Menu attribute on an ElementDef can be used to filter out any unwanted items from being potential ChangeTo candidates.</p>

ClearUndoStack

Clears the stack of undo actions used by the Quark XML Author’s Undo function. Does not use parameters.

Copy

Copies the selection to the clipboard. Does not use parameters.

CreateRendition

Creates a document rendition according to the values specified by its parameters.

Table 5-5: CreateRendition Parameters

Parameter	Required	Definition
destination	no	String. Specifies the path to which the exported file and embedded media should be saved. The path may be absolute or relative to the application folder. It may also include a default file name. If the name is omitted, the Save As dialog will display, targeted at the specified location. If this attribute is omitted, the Save As dialog will display and the user will have to specify the location and file name. Supports two patterns: {cd} (Current location of active document) and {ApplicationData} (Environment variable-APPDATA). The destination can either be one of these patterns or it can start with them.
expandRef	no	Boolean. Indicates whether or not the reference elements should be expanded into the components they reference before processing the XSLT. Set to true , to expand all references. Defaults to false .
filter	no	String. When specified, if destination is omitted, OR if {cd} is specified and one doesn't exist (document hasn't been saved), user is prompted with a Save As dialog. Within this dialog, the 'Save as type' filter is specified by this attribute. See below for more

Parameter	Required	Definition
		information.
params	no	String. Used to supply parameter values to the stylesheet. Parameters are specified as name=value pairs separated by the pipe character (). For example: "name=value name2=value2 ..."
supportPath	no	String. When a document is rendered/previewed, all files under the folder specified will be copied to the final destination of the rendered document. This process will overwrite any files of the same name and skip over files that it cannot copy. Specified as either a full path to an existing directory or a relative path to an existing directory under the Quark XML Author application folder.
xpath	no	String. Apply the transform to a fragment of the current active document pointed to by the supplied xpath instead of the entire document. The xpath must be relative to the ExportedRoot.
xslt	no	String. Specifies the path and name of the transform stylesheet. The path may be a URL, an absolute path, or a path relative to the application folder.

filter

The Save As dialog is displayed only if the destination parameter has no value. The optional filter attribute, if populated, determines what filters will be present in the **Save as Type** drop-down box in the **Save As** dialog. The value of the filter attribute is a resource ID in the Resources file. See **"Error! Reference source not found."**.

APPLICATION CONFIGURATION: INTERNAL CLASSES

The resource ID identifies a string list of filter types. Each filter type comprises two parts: the filter text and the filter definition, which are separated by the pipe character (|). For example, to specify a filter to create a html rendition, you would use the pair shown below as the value of the “HTML Rendition” resource ID:

```
HTML Rendition = HTML Document (*.html)|*.html
```

This would produce the filter shown in Figure 5-1 .

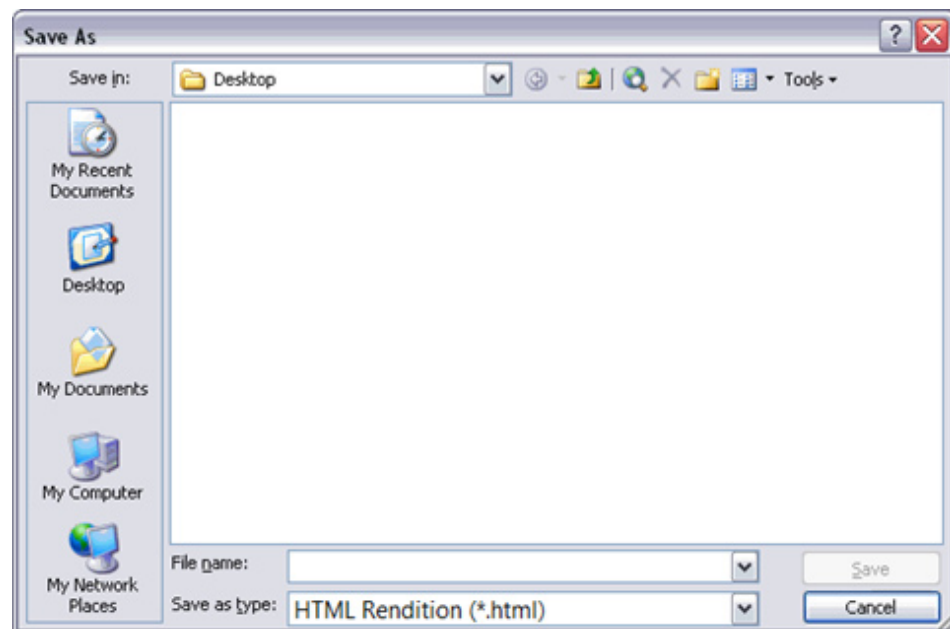


Figure 5-1: Save As dialog

Of course, the actual resource ID is arbitrary and can have any name you want. The filter is specified in this way to aid in localization.

If neither destination nor filter are specified, the rendition is saved to the Quark XML Author temp directory with a Quark XML Author-generated filename, e.g. preview_1.html.

StyleSheet Child Nodes

CreateRendition may have any number of StyleSheet child nodes. This provides the ability to perform chained transformations with the output of one stylesheet provided to the next. Any XSLT parameters that need to be supplied to the individual stylesheet can be specified by the params attribute as shown in the example shown below.

```
<StyleSheet xslt="stylesheet name"
params="name1=value1|name2=value2"/>
```

Using CreateRendition to Load External Objects

CreateRendition supports configurable stylesheet extensions that allow the execution of .NET code from within the transform itself. When CreateRendition is invoked, Quark XML Author inspects the top of the XSLT for <?XpressExtension?> processing instructions. For each instruction that is found (multiple instructions are allowed), the .NET object is loaded and made available to the transform. Additional

parameters required by the .NET object can be supported using the “params” attribute.

The format of the XSLT processing instruction is shown below:

```
<?XpressExtension namespace="ns" assembly="Assembly"
object="Invision.objectname" ?>
```

Cut

Cuts the selection from the document and copies it to the clipboard. Does not use parameters.

DeleteComment

Deletes the currently selected comment (or comment for the current selection).

Table 5-6: DeleteComment Parameters

Parameter	Required	Definition
deleteAll	no	If a value is specified for the query attribute, then deleteAll is ignored. Set to true to delete all comments in the current document. If set to false , the first comment in the current selection is deleted. Defaults to false .
query	no	Value is a regular expression that allows extracting selective comment nodes. For example: <pre><InternalClass name="DeleteComment" query="author='Some Author Another Author' && initial='AA' customAttr='customValue1 customValue2 customValu e3' /></pre> <p>The query attribute value is expressed as a series of property=value pairs, where the named property is either a property of Word comments or any custom attributes defined in the configuration. Property values must be enclosed in single quotation marks. AND (&&) and OR () operators can</p>

Parameter	Required	Definition
		be used in the expression.

DeleteTable

Deletes the currently selected table (or the table in which the cursor is presently located). Does not use parameters.

DeleteTableCol

Deletes the table column in which the cursor is located. Does not use parameters.

DeleteTableRow

Deletes the table row in which the cursor is located. Does not use parameters.

DocumentAttributes

Opens the Attribute Panel for the root element of the Document. Does not use parameters.

DocumentClose

Closes the document.

Table 5-7: DocumentClose Parameters

Parameter	Required	Definition
filename	no	String. Specifies the name of a file to be closed rather than the currently active document.
ignoreChanges	no	Boolean. Set to true to allow the user to close the document without being prompted to save changes. Set to false to prompt the user to save. Defaults to false .

DocumentNew

Creates a new document of the specified type.

Table 5-8: DocumentNew Parameters

Parameter	Required	Definition
defaultName	no	String. Specifies a default name

Parameter	Required	Definition
		for the new document. The default name is created from a regular expression given in curly brackets. A string pattern can be appended to the regular expression value. For example, {NOW(yyyyMMdd)}-???.xml The pattern in {} is a regular expression. NOW specifies today and the string in brackets specifies the date pattern to be used. The string after the } is appended to the date. {NOW(yyyyMMdd)}-INVISION.xml specifies the current date, with – INVISION.xml appended to the date. If defaultName is not specified, the value to be used is pulled from the NewDocumentTitle config resource and the appropriate number (1,2,3 etc.) is added to the string. See “Resource Files” for information about the NewDocumentTitle resource value.
templateXml	yes	String. Specifies the path and filename of the template file from which the document will be created. The path may be absolute or relative to the application folder.

DocumentOpen

Invokes the **Open** dialog.

Table 5-9: DocumentOpen Parameters

Parameter	Required	Definition
filter	no	String. See below for more information.
fullfilepath	no	String. Specifies the fully qualified path of the document to open or specifies the folder that should be displayed in the

Parameter	Required	Definition
		Open dialog.

filter

If it is populated, the Open dialog box's **Files of type** drop-down box will contain filters the attribute value specifies. The value of the filter attribute is a resource ID in the Resources file. See “**Error! Reference source not found.**”.

The resource ID identifies a string list of filter types. Each filter type comprises two parts: the filter text and the filter definition, which are separated by the pipe character (|). For example, to specify a filter to display only Word documents, (.DOC and .DOCX) you would use the pair shown below as the value of the “OpenWordDocs” resource ID:

`OpenWordDocs = Word Document (*.doc)|*.doc` This would produce the filter shown in Figure 5-2.

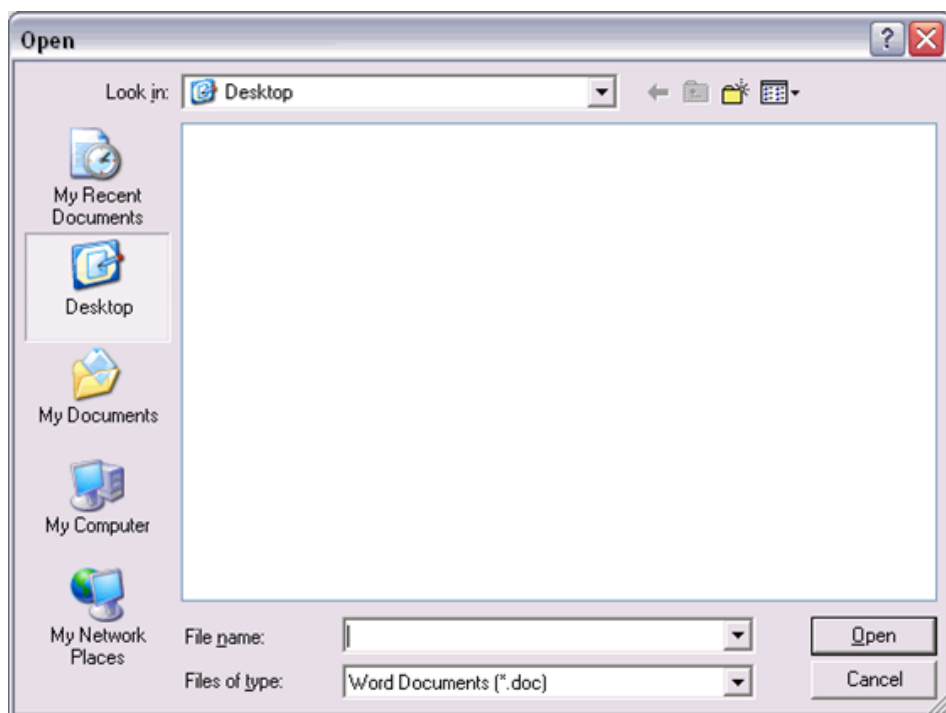


Figure 5-2: Open Dialog

Each filter set is likewise separated by a pipe character. For example, if we wanted to add Quark XML Author Documents (*.xml)|*.xml to the example already given above, it would look like the example below.

`OpenWordDocs = Word Document (*.doc)|*.doc|Quark XML Author Document (*.xml)|*.xml`

It is always wise to include a filter that shows All Files (*.*). Adding this filter to the example above produces the example below.

`OpenWordDocs = Word Document (*.doc)|*.doc|Quark XML Author Document (*.xml)|*.xml|All Files (*.*)|*.*`

Of course, the actual resource ID is arbitrary and can have any name you want. The filter is specified in this way to aid in localization.

DocumentOpenWord

Invokes the native Word FileOpen dialog for opening documents. The dialog uses default Word filters and does not use any Quark XML Author document filters. Primarily used for accessing a file on a remote web server via native Word's native WebDAV capabilities. For a remote file, the function attempts access using the Windows default credentials to download the file and if unsuccessful, prompts for username/password. The login prompt repeats if an Unauthorized status code is received on a WebDav open request. Does not use parameters.

DocumentSave

Saves the current document.



Note that Quark XML Author leverages Word's native AutoSave functionality so that documents are automatically saved periodically according to the setting in the user's Word environment.

Using DocumentSave by itself simply saves the document. However, the following parameters can be passed to modify the save function.

Table 5-10: DocumentSave Parameters

Parameter	Required	Definition
defaultFilepath	no	String. If the document has never been saved, then the SaveAs feature is invoked and this parameter is used. Specifies the folder that should be used as the Save In folder in the Save As dialog. The dialog uses this path as the current folder of the dialog. The user is not restricted to this path.
transform	no	String. Specifies the location and filename of the default stylesheet for the document class. The value of this parameter may be a URL, an absolute path, or a path relative to the application folder. The file name must be

Parameter	Required	Definition
		included.
filter	no	String. If the document has never been saved, then the SaveAs feature is invoked and this parameter is used. See “DocumentSaveAs” for more information.

Also these parameters overwrite any values specified in the DefaultSaveOptions node.

DocumentSaveAs

Invokes the Save File dialog box so that the user can select a location and filename for the document. Supports WebDAV saves.

Table 5-11: DocumentSaveAs Parameters

Parameter	Required	Definition
filter	no	String. If the document has never been saved, then the SaveAs feature is invoked and this parameter is used. See below for more information.
defaultFilepath	no	String. Specifies the folder that should be used as the Save In folder in the Save As dialog. The dialog uses this path as the current folder of the dialog. The user is not restricted to this path.
transform	no	String. Specifies the location and filename of the default stylesheet for the document class. The value of this parameter may be a URL, an absolute path, or a path relative to the application folder. The file name must be included.
useWordDialog	no	Boolean. Specifies whether or not the Word Dialog should be displayed. Set to true to allow the user to save the Quark XML Author document as a Word document.

Parameter	Required	Definition
fullfilepath	no	String. Specifies the fully qualified path of the document to open or specifies the folder that should be displayed in the Open dialog.

filter

The optional filter attribute, if populated, determines what filters will be present in the **Save as Type** drop-down box in the Save As dialog. The value of the filter attribute is a resource ID in the Resources file. See “**Error! Reference source not found.**”.

The resource ID identifies a string list of filter types. Each filter type comprises two parts: the filter text and the filter definition, which are separated by the pipe character (|). For example, to specify a filter to save as a Word document, you would use the pair shown below as the value of the “SaveWordDocs” resource ID:

```
SaveWordDocs = Word Document (*.doc) | *.doc
```

This would produce the filter shown in Figure 5-3.

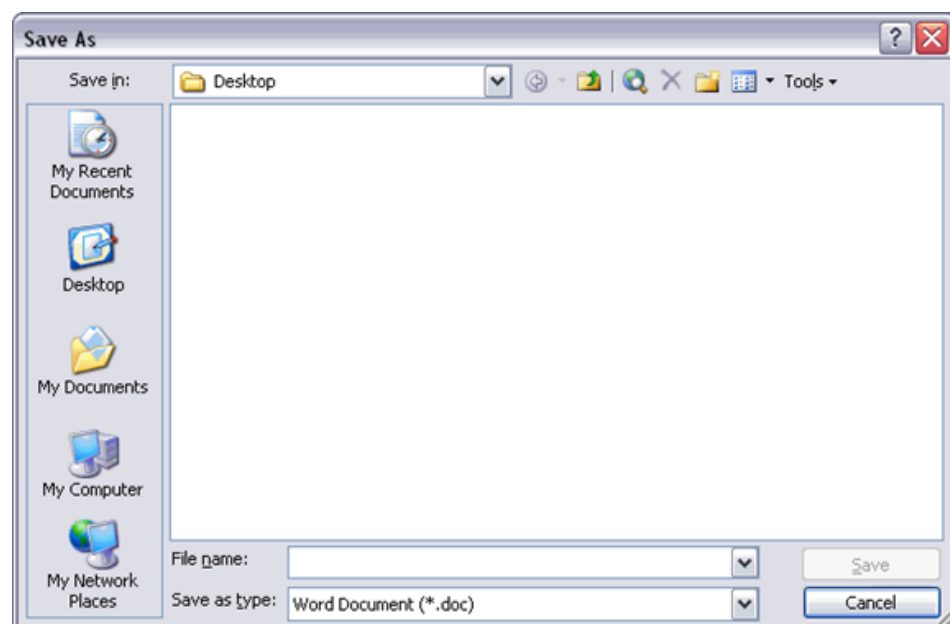


Figure 5-3: Save As Filter

Of course, the actual resource ID is arbitrary and can have any name you want. The filter is specified in this way to aid in localization.

EmailRendition

Creates a document rendition according to the values specified by its parameters and invokes Send to Mail Recipient with the document rendition as an attachment.



All of EmailRendition's parameters are optional. However, if the rendition parameter is omitted, at least one of the other parameters must be used.

Table 5-12: EmailRendition Parameters

Parameter	Required	Definition
destination	no	String. Specifies the path to which the exported file and embedded media should be saved. The path may be absolute or relative to the application folder. It may also include a default file name. If the name is omitted, the Save As dialog will display, targeted at the specified location. If this attribute is omitted, the Save As dialog will display and the user will have to specify the location and file name.
expandRef	no	Boolean. Indicates whether or not the reference elements should be expanded into the components they reference before processing the XSLT. Set to true , to expand all references. Defaults to false .
filter	no	String. See below for more information.
params	no	String. Used to supply parameter values to the stylesheet. Parameters are specified as name=value pairs separated by the pipe character (). For example: "name=value name2=value2 ..."
rendition	no	String. Specifies the file format value of the Mail Attachment. Valid values are nativeFormat or xpressFormat : nativeFormat, which sends as a

Parameter	Required	Definition
		.doc file; or xpressFormat, which attaches the Quark XML Author .xml file. For example: rendition="xpressFormat" or rendition="nativeFormat" Using rendition is preferred over omitting it. However, if it is omitted, at least one of the other parameters must be included.
xslt	no	String. Specifies the path of the transform stylesheet The path may be a URL, an absolute path, or a path relative to the application folder.

filter

The Save As dialog is displayed only if the destination parameter has no value. The optional filter attribute, if populated, determines what filters will be present in the **Save as Type** drop-down box in the Save As dialog. The value of the filter attribute is a resource ID in the Resources file. See **"Error! Reference source not found."**.

The resource ID identifies a string list of filter types. Each filter type comprises two parts: the filter text and the filter definition, which are separated by the pipe character (|). For example, to specify a filter to save as a Word document, you would use the pair shown below as the value of the "SaveWordDocs" resource ID:

```
SaveWordDocs = Word Document (*.doc) | *.doc
```

This would produce the filter shown in "Filter Figure".

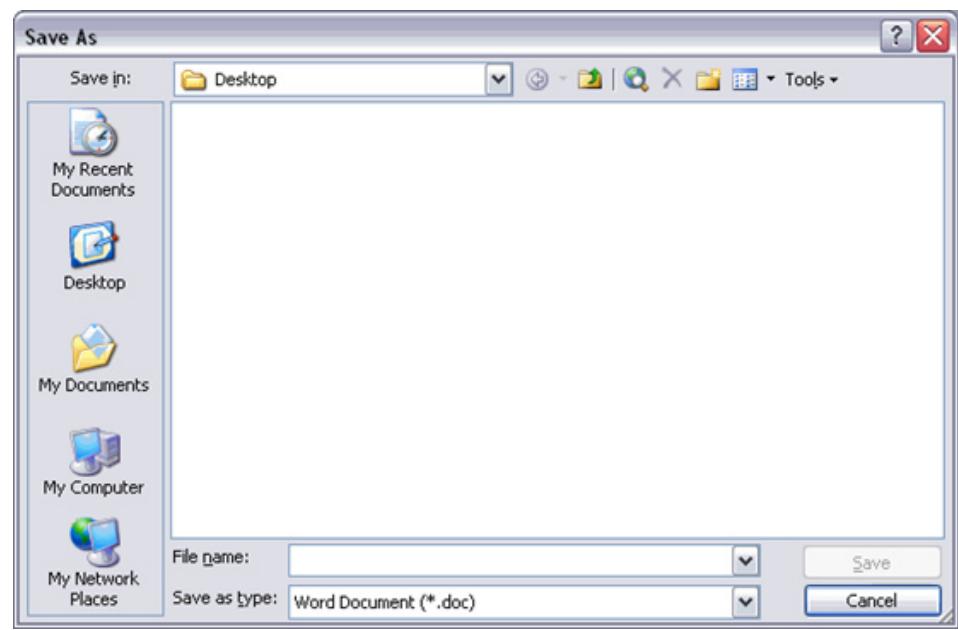


Figure 5-4: Save As Filter

Each filter set is likewise separated by a pipe character. For example, if we wanted to add Quark XML Author Document (*.xml)|*.xml to the example already given above, it would look like the example below.

```
SaveWordDocs = Word Document (*.doc)|*.doc|Quark XML Author
Document (*.xml)|*.xml
```

It is always wise to include a filter that shows All Files (*.*). Adding this filter to the example above produces the example below.

```
SaveWordDocs = Word Document (*.doc)|*.doc|Quark XML Author
Document (*.xml)|*.xml|All Files (*.*)|*.*
```

Of course, the actual resource ID is arbitrary and can have any name you want. The filter is specified in this way to aid in localization.

If neither destination nor filter are specified, the rendition is saved to the Quark XML Author temp directory with a Quark XML Author-generated filename, e.g. MyDoc.doc.

EmphasisAction

Used to apply emphasis.

Table 5-13: EmphasisAction Parameters

Parameter	Required	Definition
emphasis	yes	String. Specifies the name of the emphasis that should be applied. Must be an emphasis style defined in the document schema. In the example below, the style bold would be

Parameter	Required	Definition
		<p>applied to the selected text.</p> <pre> <InternalClass name="EmphasisAction" emphasis="bold"/> <InternalClass name="EmphasisAction" emphasis="emphasis"> <SubStyle key="style" value="bold"/> </InternalClass> </pre>
key	no	String. Specifies sub-style values for a particular emphasis.
value	no	String. Specifies sub-style values for a particular emphasis.

EmphasisHandler

Invokes the Quark XML Author emphasis handler for displaying a combo box of allowed emphasis items at the current location. Does not use parameters.

FindAndReplace

Provides Find, Replace, and navigation functionality for Quark XML Author documents similar to native Word functionality. There are some important differences in functionality:

- The Replace button is disabled for find phrases which the schema makes illegal to be replaced. A tooltip shows the reason for disabling the button.
- The Replace All button will only replace selections which are editable and ignore all non-editable selections.
- The Goto list box is restricted to the following 6 types: Page, Section, Line, Comment, Table, and Graphic.
- The Enter page number text box in the Goto tab has the following restrictions:
 - The text must be a numeric value.
 - The numeric value can be preceded with a + or a - to indicate a relative Goto.

This behavior is a subset of the features Word offers which supports a wider variety of formats in the text box. For example, s2p5 is a valid value and takes the user to the page five in section two of the Word document.

- The replace/Replace all will not work on read-only (i.e. accessMode=Review) documents.
- The gotoNext() and GotoPrevious() functionality still invokes the Word's Goto/Find Next. The browse object in the lower right corner of the documents on the vertical scrollbar (the round icon and the two arrow buttons around it) still invoke Word functions.

Table 5-14: FindAndReplace Parameters

Parameter	Required	Definition
tab	yes	String. The required tab parameter may have one of three values: Find , Replace , and Goto . These values refer to the three tabs available in the Find and Replace dialog. The value specified defines the default tab to be presented when the dialog is displayed to the user.

FormattingAction

Used to apply/remove character formatting, font color or highlight color for the selected word, phrase or text elements.

Typography such as bold and italic may be applied to a word, phrase or multiple text elements. This formatting feature is similar to the formatting feature in Word.

From the ReadMe description: In Business Document solutions, the ability to apply stylistic emphasis such as bold and italics has been replaced with new paragraph styling in a new feature called Character Formatting. This applies to: Bold, Italic, Underline, Strikethrough, Superscript, and Subscript. These styles are applied using the same user controls as in previous versions of XML Author. For example, the Bold button on the Home ribbon. Different from the previous inline elements version of these styles, metadata may not be applied to these new styles. In Business Document solutions, text foreground color may be applied to a word, phrase or multiple text elements. This is done using the Font Color feature. In Business Document solutions, text background color may be applied to a word, phrase or multiple text elements. This is done using the Text Highlight Color feature.

Table 5-15: FormattingAction Parameters

Parameter	Required	Definition
formattingName	yes	String. Specifies the name of the formatting that should be applied. Must be a name of the formatting style defined in the document schema. From the Business Documents

Parameter	Required	Definition
		<p>XAS schema:</p> <pre> <Formatting> <Bold name="b" friendly="Bold"/> <Italic name="i" friendly="Italic"/> <Underline name="u" friendly="Underline"/> <Strikethrough name="strikethrough" friendly="Strikethrough" /> <Subscript name="sub" friendly="Subscript"/> <Superscript name="sup" friendly="Superscript"/> <FontColor name="color" friendly="Font Color" colorAttributeName="text forecolor"/> <HighlightColor name="highlight" friendly="Highlight Color" colorAttributeName="high lightcolor"/> </Formatting> </pre> <p>These definitions are available to all configurations and are enabled by default in Business Documents configuration. Serialization name is “name”. For example, “b” for Bold. However, custom configurations can customize the name and friendly name of formatting definitions..</p> <p>Example calls:</p> <pre> <InternalClass name="FormattingAction" </pre>

Parameter	Required	Definition
		<pre>formattingName="bold" colorValue="null"/> <InternalClass name=" FormattingAction " formattingName ="color" colorValue="0000FF" /></pre>
colorValue	no	String. For formattingName equals <i>color</i> or <i>highlight</i> , this specifies the color name or RGB value of the color that should be applied to the selection. If the selection already has a color and the current color is different than this becomes the new color. The color value could be in hex string or color name string. Example: "0000FF", "blue" etc.

FormattingActionChangeCase

Used to change case for the selected word, phrase or text elements. This formatting feature is similar to the formatting feature in Word.

Table 5-16: FormattingActionChangeCase Parameters

Parameter	Required	Definition
changeCase	yes	<p>String. Specifies the name of the formatting that should be applied.</p> <p>Example calls:</p> <pre><InternalClass name=" FormattingAction ChangeCase" changeCase ="titlesentence" /></pre>

FormatTableCellShading

Invokes the Cell Shading dialog, which allows users to assign a background color to the current selection of table cells. Does not use parameters.

This requires that cells of a table section are defined with an attribute of the datatype `tableCellBackgroundColor`. See "Table 11-8". For this discussion, let's refer to this as simply the "color" attribute.

The user's color selection is applied to the selected cells on the Word canvas and is applied to the color attribute in the XML for each selected cell. However, if the user selects "No Color", then the default color is applied to the Word canvas and the color attribute is removed in the XML.

In the user interface, the toolbar or ribbon button is enabled if the current cell or selected cells are editable and are defined with a color attribute.

In the Cell Shading dialog, the colors available can be specified in the attribute definition. This is typically used to limit the user to a specific set of colors. For example, only three specific colors are available for header cells. If no colors are specified in the attribute definition, then the colors available are Word's stock colors for cell shading.

Selections containing cells with different color attribute definitions

In this situation, the colors available in the Cell Shading dialog are the intersection of the colors available for each different color attribute definition. The colors that are found to be in common are displayed in the user interface. If the intersection is zero colors, then no colors are available and the only action the user can take is to select "No Color. The system removes the color attribute for all selected cells.

IndentElement

Triggers indent/decrease indent behavior for the current element, depending on the value of the **indent** parameter.

Table 5-16: IndentElement Parameters

Parameter	Required	Definition
indent	yes	<p>String. Valid values are: IncreaseIndent and DecreaseIndent. When the value of indent is IncreaseIndent, IndentElement changes the selected element to the element type specified by the element definition's increaseIndent attribute. When the value of indent is DecreaseIndent, IndentElement changes the selected element to the element type specified by the element definition's decreaseIndent attribute.</p> <p>Consider the following element structure:</p> <pre><ElementDef name="List2"</pre>

Parameter	Required	Definition
		<pre> xmlns="list" friendly="Sub List" visible="false"> <Section> <Sequence maxOccurs="unbounded"> <SectionType name="ListItem2" increaseIndent="ListItem 3" decreaseIndent="ListItem "/> </Sequence> </Section> <Attributes/> </ElementDef> </pre> <p>If a ListItem2 element is selected, invoking the first example below would change the element to a ListItem3. Invoking the second example below would change the element to a ListItem.</p> <p>Example 1: Increase Indent</p> <pre> <InternalClass name="IndentElement" indent="IncreaseIndent"/ > </pre> <p>Example 2: Decrease Indent</p> <p>Note that multiple targets for Increase/Decrease Indent may be specified in the increaseIndentand decreaseIndent attributes of SectionType. See “SectionType” on page 232 for details.</p>

InlineAttributes

Toggles the display of portionmarks on or off.

Table 5-17: InlineAttributes Parameters

Parameter	Required	Definition
Type	no	String. Valid values are PreText and PostText which specifies the type of portionmark to be affected. Defaults to PreText . See Section 9.1 for information about portionmarks.

InsertXACrossref

This applies to embedded Word documents only.

Inserts a cross-reference link at the current cursor location in the embedded Word document if a valid target is on the clipboard. The valid target is a destination in an XML Author document. The destination may be in the document that is the parent of the embedded Word document or it may be another XML Author document.

Does not use parameters.

Example:

```
<!-- Internal class to insert Cross-Reference when working in an
embedded OLE document -->
<button id="InsertXACrossRef" imageMso="CrossReferenceInsert"
size="large">
  <InternalClass name="InsertXACrossRef"/>
</button>
```

InsertComment

Inserts a comment into the document. Does not use parameters.

InsertColumnBreak

Toggles the **columnBreak** attribute value on the next visible element following the cursor location between **true** and **false**. See section 12.3 for information on the **columnBreak** attribute. Does not use parameters.

InsertCustomLink

Inserts a cross-reference link at the current cursor location if a valid target is on the clipboard. Does not use parameters.

Example:

```
<!-- Internal class to insert Cross-Reference when working in an
XML Author document -->
<button id="InsertCustomLink" imageMso="CrossReferenceInsert"
size="large" getEnabled="GetEnabled">
```

```
<InternalClass name="InsertCustomLink"/>
</button>
```

InsertElement

The internal class InsertElement is used only within the DocConfig file to specify elements to be inserted, and what child elements to automatically insert. For example, a Section element might have a mandatory Title element, to be followed by one or more Paragraph elements. When the user inserts a Section element, the syntax of the displayName value will instruct Quark XML Author to insert the required child elements. You can also specify optional elements to be automatically inserted.

displayName

Required. String. The displayName parameter value uses the following syntax:

```
<InternalClass name="InsertElement"
displayName="Element[Child[Grandchild/Grandchild/Grandchild]]">
```

Brackets

The [] brackets signifies a parent-child relationship, and the slash indicates sibling relationships. So the example cited above, of a Section with mandatory Title and one or more paragraphs, would look like the following example:

```
<InternalClass name="InsertElement"
displayName="Section[Title/Paragraph()] ">
```

Parenthesis

Parenthesis can be used to indicate content objects:

Table type element

A tilde (~) indicates a table type element.

```
<InternalClass name="InsertElement" displayName="CALS Table(~)"
externalMethodFriendly="true"/>
```

MediaContent object

An asterisk (*) indicates a MediaContent object, such as shown below.

```
<InternalClass name="InsertElement" displayName="Quark
Extensions[Local Media[Media(*)/Media Caption]"/>
```

Default text value

Parenthesis can also contain text that will indicate a default text value. This value will override any default value specified in the XAS:

```
<InternalClass name="InsertElement" displayName="Quark
Extensions[Local Media[Media(*)/Media Caption(Cartoon here)"/>
```

Reserved Characters

When specifying `displayName` for either “InsertElement” or “Insert Emphasis” note that there are several special characters reserved by the system. If these characters are used as part of the `displayName` value, you must replace them with the corresponding code shown in the table below.

Table 5-18: Control Codes for Reserved Characters

Character	Description	Code
/	Forward slash	%#fs%#
[Left Square Bracket	%#ls%#
]	Right Square Bracket	%#rs%#
{	Left Curly Bracket	%#lc%#
}	Right Curly Bracket	%#rc%#
*	Star	%#st%#
~	Tilde	%#ti%#
>	Greater Than	%#gt%#
<	Less Than	%#lt%#
'	Apostrophe	%#apos%#
"	Quotation Mark	%#quot%#
&	Ampersand	%#amp%#

For example, the `InsertEmphasis` Internal class below would insert **CH2(parens)** into the document:

```
<InternalClass name="InsertEmphasis"
  displayName="Symbol (CH2%#lc%#parens%#rc%#) " />
```

These codes can also be used in the schema when specifying default text for an `ElementDef`. (See section 14).

Insert Emphasis

Inserts an emphasized text node into the specified XML node at the cursor position.

Table 5-19: InsertEmphasis Parameters

Parameter	Required	Definition
<code>displayName</code>	yes	The friendly name of the emphasis as defined in the schema and any text to insert are provided in the required <code>displayName</code> attribute. The friendly name is provided first, with text to insert in

Parameter	Required	Definition
		<p>parenthesis. For example:</p> <pre><InternalClass name="InsertEmphasis" displayName="Company ([Co mpany Name]) "></pre> <p>The example above will insert [Company Name] at the cursor location where the Company emphasis is allowed.</p> <p>Can be used as a placeholder for XML data when transforming Quark XML Author document to output.</p> <p>The value of displayName for InsertEmphasis has the same restrictions regarding special characters as it does for InsertElement. See Table 5-18: Control Codes for Reserved Characters.</p>

InsertEndNote

Inserts an end note emphasis named in the parameter. End notes can be inserted in any element type. There must be a corresponding EndNote element definition and an emphasis style defined for the notes themselves. See section 14.8 for a full discussion about configuring end note capability.

Table 5-20: InsertEndNote Parameters

Parameter	Required	Definition
emphasisName	yes	String. Specifies the name of the emphasis defined for the note.

InsertEntityReference

Invokes the Insert Entity Reference dialog, which allows users to select an entity reference to insert into the document. Does not use parameters.

InsertFootnote

Inserts a footnote at the current cursor location.

Table 5-21: InsertFootNote Parameters

Parameter	Required	Definition
noteType	no	String. Specifies whether the inserted note should be a Footnote or Endnote . Defaults to Footnote .

InsertHyperlink

Invokes the Quark XML Author Insert Hyperlink dialog. The resulting **Text to display** is inserted into the document, emphasized, and its href attribute assigned the value of the **Address**. As with Word, when the current selection is within the link emphasis, invoking InsertHyperlink allows editing the link. Does not use parameters.

InsertInlineElement

Allows inserting of an inline element (media or OLE) for a para type element.

Table 5-22: InsertInlineElement Parameters

Parameter	Required	Definition
displayName	yes	<p>String. Specifies the friendly name of the inline element that should be inserted.</p> <pre><!-- displayName is the friendly name of any inline element (any element that is defined as an OLE or media(reference or embedded)).For reference media only invisible containers with a required Media child is supported as a valid inline element--></pre> <pre><InternalClass name="InsertInlineElement" displayName="Math Equation"/></pre>

InsertPageBreak

Toggles the **pageBreak** attribute value on the next visible element following the cursor location between **true** and **false**. See section 12.3 for information on the **pageBreak** attribute. Does not use parameters.

InsertSectionBreak

Toggles the **sectionBreak** attribute value on the next visible element following the cursor location between **true** and **false**. See section 12.3 for information on the **sectionBreak** attribute.

Table 5-23: InsertSectionBreak Parameters

Parameter	Required	Definition
breakType	no	String. Specifies the type of section break to be inserted. May be set to either sectionBreaknextPage or sectionBreakContinuous . Defaults to sectionBreaknextPage if not specified.
columnCount	no	String. Specifies the number of columns in the section break. The value for this attribute should be an integer from 1 to 12. When columnCount is used, it is recommended (not required) that breakType attribute is set to sectionBreakContinuous . See section 5.555.55, SetColumns,

InsertTable

Displays the Insert Table dialog. For more information about elements that affect this dialog, see “TableStyle” and “Template”.

Table 5-24: InsertTable Parameters

Parameter	Required	Definition
definition	yes	String. Specifies the name of the table definition that should be used for the new table. See “Table” for more information.
style	no	String. Specifies the name of

Parameter	Required	Definition
		the TableStyle that should be used for the new table. See TableStyle for more information.
template	no	String. Specifies the name of the table template that should be used for the new table. See “Templates” for more information.

InsertTableColumn

Insert a new column in the table.

Table 5-25: InsertTableColumn Parameters

Parameter	Required	Definition
direction	yes	String. Specifies where the new column should be inserted relative to the current selection. Valid values are: right and left .

InsertTableRow

Insert a new row in the table.

Table 5-26: InsertTableRow Parameters

Parameter	Required	Definition
direction	yes	String. Specifies where the new row should be inserted relative to the current selection. Valid values are: above and below .

ManageTextEntities

Invokes the Manage Text Entities form, which allows users to add, edit, and remove content entities and values. Does not use parameters. This applies to internal content entity definitions. External content entities defined outside of the document are not displayed in the form.

MergeComments

Used to merge into the current document the comments of other documents.

Selecting Files

To allow the user to select those documents, do not use the <Token> element and MergeComments will display the **MergeComments** dialog.

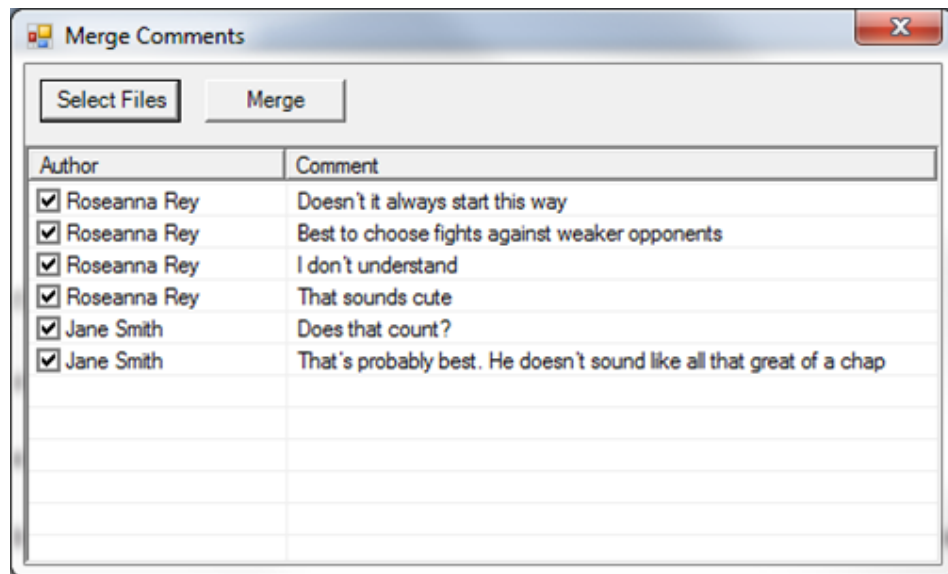


Figure 5-5: Merge Comments dialog

To provide the list of other documents programmatically, use the optional <Token> element.

```
<InternalClass name="MergeComments">
  <Token>filepath=...</Token>
  <Token>filepath=...</Token>
  ...
</InternalClass>
```

Each of the filepath tokens must point to a valid file.

Automatically Merging All Comments

Use the optional <AutoMerge> element to automatically merge all comments rather than allowing the user to choose on a comment-by-comment basis which comments should be merged. Only used in conjunction with <Token>.

```
<InternalClass name="MergeComments">
  <Token>filepath=...</Token>
  <Token>filepath=...</Token>
  <AutoMerge/>
  ...
</InternalClass>
```

MergeTableCells

Merges the selected table cells. Does not use parameters.

PageLayout

Invokes the interface for setting page orientation. Does not use parameters.

Paste

Paste the clipboard contents into the document at the current location. Does not use parameters

DefaultPasteOptionAsText

Allows you to specify whether to or not to automatically paste copied content as Unicode text without formatting when using the CTRL-V short cut.

```
<DefaultPasteOptionAsText>true</DefaultPasteOptionAsText>
```

NOTE: If this parameter is not defined, the default value is **false**. When the value is **false**, the text that is pasted is based on the current cursor position as well as the data available on the clipboard. It could be pasted as unicode text, formatted text or an <element>.

PreviewRendition

Creates a document rendition according to the values specified by its parameters and displays the rendition in the specified application. PreviewRendition uses the same set of parameters as CreateRendition with the addition of the **appPath** parameter.

Table 5-27: PreviewRendition Parameters

Parameter	Required	Definition
appPath	no	String. Location of the launch application.
destination	no	String. Specifies the path to which the exported file and embedded media should be saved. The path may be absolute or relative to the application folder. It may also include a default file name. Supports two patterns: {cd} (Current location of active document) and {ApplicationData} (Environment variable-APPDATA). The destination can either be one of these patterns or it can start with them.
expandRef	no	Boolean. Indicates whether or not the reference elements should be expanded into the components they reference before processing the XSLT. Set

Parameter	Required	Definition
		to true , to expand all references. Defaults to false .
filter	no	String. When specified, if destination is omitted, OR if {cd} is specified and one doesn't exist (document hasn't been saved), user is prompted with a Save As dialog. Within this dialog, the 'Save as type' filter is specified by this attribute. See below for more information.
params	no	String. Used to supply parameter values to the stylesheet. Parameters are specifies as name=value pairs separated by the pipe character (). For example: "name=value name2=value2 ..."
supportPath	no	String. When a document is rendered/previewed, all files under the folder specified will be copied to the final destination of the rendered document. This process will overwrite any files of the same name and skip over files that it cannot copy. Specified as either a full path to an existing directory or a relative path to an existing directory under the Quark XML Author application folder.
xpath	no	String. Apply the transform to a fragment of the current active document pointed to by the supplied xpath instead of the entire document. The xpath must be relative to the ExportedRoot.
xslt	no	String. Specifies the path and name of the transform stylesheet. The path may be a URL, an absolute path, or a path relative to the application folder.

filter

The Save As dialog is displayed only if the destination parameter has no value. The optional filter attribute, if populated, determines what filters will be present in the **Save as Type** drop-down box in the Save As dialog. The value of the filter attribute is a resource ID in the Resources file. See “**Error! Reference source not found.**”.

The resource ID identifies a string list of filter types. Each filter type comprises two parts: the filter text and the filter definition, which are separated by the pipe character (|). For example, to specify a filter to save as a Word document, you would use the pair shown below as the value of the “SaveWordDocs” resource ID:

```
SaveWordDocs = Word Document (*.doc)|*.doc
```

This would produce the filter shown in “Figure 5-6”.

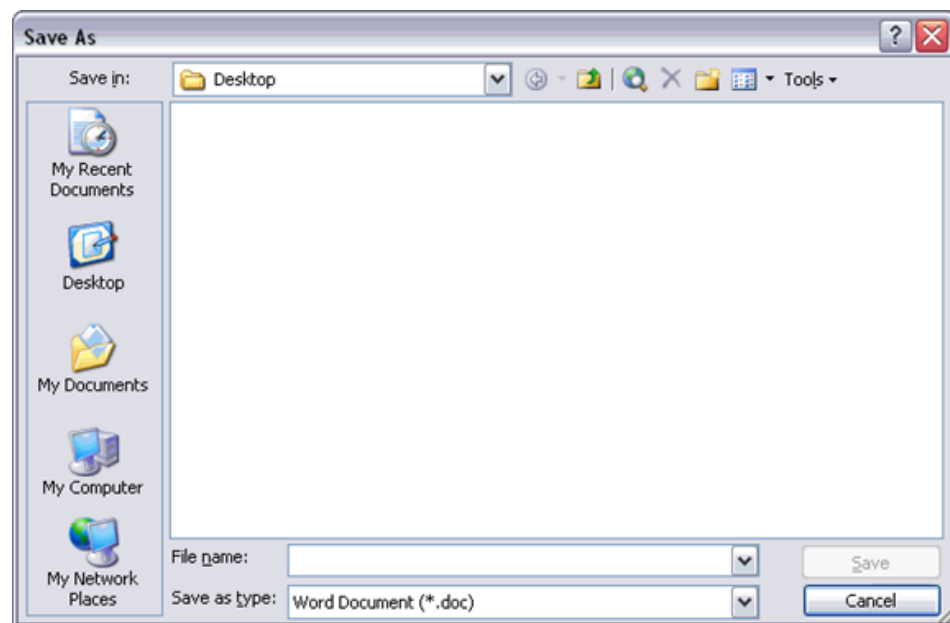


Figure 5-6: Save As Filter

Each filter set is likewise separated by a pipe character. For example, if we wanted to add Quark XML Author Document (*.xml)|*.xml to the example already given above, it would look like the example below.

```
SaveWordDocs = Word Document (*.doc)|*.doc|Quark XML Author
Document (*.xml)|*.xml
```

It is always wise to include a filter that shows All Files (*.*). Adding this filter to the example above produces the example below.

```
SaveWordDocs = Word Document (*.doc)|*.doc|Quark XML Author
Document (*.xml)|*.xml|All Files (*.*)|*.*
```

Of course, the actual resource ID is arbitrary and can have any name you want. The filter is specified in this way to aid in localization.

If neither destination nor filter are specified, the rendition is saved to the Quark XML Author temp directory with a Quark XML Author-generated filename, e.g. MyDoc.doc.

StyleSheet Child Nodes

PreviewRendition may have any number of StyleSheet child nodes. This provides the ability to perform chained transformations with the output of one stylesheet provided to the next. Any XSLT parameters that need to be supplied to the individual stylesheet can be specified by the **params** attribute as shown in the example shown below.

```
<StyleSheet xslt="stylesheet name"
params="name1=value1|name2=value2"/>
```

Using PreviewRendition to Load External Objects

PreviewRendition supports configurable stylesheet extensions that allow the execution of .NET code from within the transform itself. When PreviewRendition is invoked, Quark XML Author inspects the top of the XSLT for <?XpressExtension?> processing instructions. For each instruction that is found (multiple instructions are allowed), the .NET object is loaded and made available to the transform. Additional parameters required by the .NET object can be supported using the “params” attribute.

Table 5-28: PreviewRendition Parameters

Parameter	Required	Definition
params	no	String. Used to supply parameter values to the stylesheet. Parameters are specifies as name=value pairs separated by the pipe character (). For example: “name=value name2=value2 ...”

The format of the XSLT processing instruction is shown below:

```
<?XpressExtension namespace="ns" assembly="Assembly"
object="Invision.objectname" ?>
```

Redo

Invokes Quark XML Author’s Redo function. Does not use parameters.

Similar to Word, the Redo stack is cleared each time an action is pushed onto the Undo stack.

RejectRevision

Rejects the currently selected document revision or, if the **rejectAll** parameter is present and set to **true**, rejects all document revisions. If neither parameter is present, any revision(s) in the current selection will be rejected.

Table 5-29: RejectRevision Parameters

Parameter	Required	Definition
keyCode	no	Integer. Specifies the keyCode for a specific document node. If keyCode is a valid node, RejectRevision will reject revision(s) for the node specified.
rejectAll	no	Boolean. Set to true to reject all revisions in the document. Set to false to reject all revisions in the current selection. Defaults to false .

SaveToRepository

Saves the current document to the specified repository.

Table 5-30: SaveToRepository Parameters

Parameter	Required	Definition
assembly	yes	String. Specifies the file name of the .NET class library of the external process, without the .dll extension. For example, About.dll would be identified simply as About . A relative path to the assembly may be supplied.
class	yes	String. Specifies the fully namespace-qualified class name within the .dll file being called by the Save function. For example, Invision.Xpress.Extensible.MultiSelectPopup.

Tokens

Token child elements can be used to specify additional parameter information. See section 6.2.1.2 for more information on using tokens.

Delegates

Indicates that the Argument will have one or more Delegate child nodes that will make a call to an external DLL for information. The syntax is shown in the example below.

```
<Delegate>InvokeInternalClass</Delegate>
```

Delegates refer to methods in the external DLL that may be called to perform functions within the application. Some are named similarly to methods in other assemblies, and may be called by them. See section 6.4, for a list of available delegates and information on how to use them.

SetAccessMode

Allows user to change the accessMode property of a document dynamically.

Table 5-31: SetAccessMode Parameters

Parameter	Required	Definition
accessMode	yes	String. The required accessMode parameter specifies the access mode to which the author can change the document. The parameter may be set to one of the values listed in Table 5-32.

Table 5-32: accessMode Values

Value	Definition
Author	User can perform all editing and authoring functions.
Comment	User cannot edit document content. User can view, add, modify and delete all comments including comments created by other authors.
RestrictedComment	User cannot edit document content. User can view their own comments in Balloons or use the Reviewing Pane to add, modify and delete comments, and see other's comments.
Review	Read-only access. User can view the document, but cannot change any content or any comments.
Revise	Does not affect the editing experience, but is used to indicate when menu options are available.

User Experience

The user experience for working with comments is currently not documented in an out-of-the-box user guide so we are describing the experience here for archival purposes.

Editing Comments

Comments may only be edited in Word's Reviewing Pane.

Viewing Comments

In Word's comment balloons, only the comments of the current author are displayed.

In Word's Reviewing Pane, the comments of all authors are displayed.

Access Modes and SharePoint

For details on modeling access modes in SharePoint see the "QXA_Technical_Reference_Manual_Adapter_for_SharePoint_EI_4.0" manual.

SetColumns

Launches the Quark XML Author **Columns** dialog. Does not use parameters. See the user guide for details on the user experience.

SpellChecker

Launches the Quark XML Author spelling checker. Does not use parameters.

ShowAttributesHandler

Displays the Attributes Panel dialog.

Table 5-33: ShowAttributesHandler Parameters

Parameter	Required	Definition
xpath	no	<p>String. An xpath value that describes the element that will be affected. The xpath is from the XomRoot.</p> <p>If an xpath for a specific element is provided, the visible attributes for that element are displayed in the Attributes Panel dialog.</p> <p>If an xpath is not provided, the XomCurrentNode is used as the starting point in a search for an element that has visible attributes. If the current element does not have visible attributes, the nearest ancestor element is located and</p>

Parameter	Required	Definition
		evaluated. If not found, this process continues recursively up the document tree. In regards to the user selection, the container of that selection should be evaluated. For example, if text selection is within an emphasis, then the containing emphasis is the starting point.

SplitTableCells

Splits the currently selected table cell. Does not use parameters.

StyleHandler

Invokes the Quark XML Author style handler. Does not use parameters.

TableAutoFitBehavior

Sets the column width behavior for the selected table.

Table 5-34: TableAutoFitBehavior Parameters

Parameter	Required	Definition
value	yes	<p>Specifies whether columns have fixed width or a width proportional to the number of columns in the table. Valid values are Fixed and Proportional. If set to Fixed, invoking TableAutoFitBehavior will set the table to fixed-width columns. The table width will change as columns are inserted into or deleted from the table but the column widths will not. The table may extend past the edge of the Word canvas as a result.</p> <p>If set to Proportional, invoking TableAutoFitBehavior will set the table to proportional-width columns. The table width will not</p>

Parameter	Required	Definition
		change as columns are inserted into or deleted from the table. Columns will be resized as columns are added to or removed from the table so that their widths are proportional to the table size. That is, if a table has four columns and a fifth is added, each column will be 1/5 the width of the table. If a table has four columns and one is removed, the remaining columns will be resized to 1/3 the width of the table.

TableBorders

Invokes the toolbar drop-down table borders tool, which allows the user to apply borders to table cells.

Table 5-35: TableBorders Parameters

Parameter	Required	Definition
borderType	yes	String. Specifies the border type to apply to the currently selected table cells.

TableCellAlignment

Invokes the Table Cell Alignment handler.

In Word, the handler uses the following parameter:

Table 5-36: TableCellAlignment Parameters

Parameter	Required	Definition
align	yes	String. Specifies the alignment of content within the cell. The first character species the vertical alignment: t =top, m =middle, and b =bottom. The second character species the horizontal alignment: j =justify, r =right, c =center and l =left.

For example,

```
<toggleButton id="TableCellAlignTopLeftXA" label="Align Top Left"
imageMso="TableCellAlignTopLeft" showLabel="false">

    <InternalClass name="TableCellAlignment"
align="tl"/>

</toggleButton>
```

TableDistributeColumns

Distributes currently selected table columns so that all are equal width. Does not use parameters.

TableDistributeRows

Distributes currently selected table rows so that all are equal height. Does not use parameters

TableRefresh

Redraws the selected table. Does not use parameters.

QuarkSubSection

Displays Text Direction dialog. Allows user to change the text direction for the table cell. Does not use parameters.

ToggleCommentsPane

Toggles the Comments Pane opened and closed. Does not use parameters.

ToggleEmpty

Toggles the Empty style between its standard font height/background color and the specified values.

Toggles the Empty style between its standard font height/background color and the specified values.

ToggleEntityView

Toggles the display of content entities in the document between the entity names and their values. Does not use parameters.

ToggleKeepWithNext

For the selected element, toggles the value of the **keep-with-next** attribute between **true** and **false**. Does not use parameters.

ToggleTrackChange

For the current document, toggles the Track Changes feature between **true** and **false**. Does not use parameters.

ToggleVisibleElement

Renders an element, emphasis, or set of elements or emphasis, identified by the **xpath** parameter, visible or hidden, depending on the value of the **state** parameter. Invoking this Internal Class sets the value of the **inv:visible** attribute for the specified element(s) to **true** if **state="Show"** or to **false** if **state="Hide"**.

Only elements that are defined as visible in the XAS can be hidden/shown in this manner.



Children of the table node (table heading rows, table body, and so forth) cannot be hidden/shown independently of the parent table

Table 5-38: ToggleVisibleElement Parameters

Parameter	Required	Definition
state	yes	String. Valid values are: Show or Hide . If set to Show , existing inv:visible attributes for the specified element(s) are removed (because this attribute defaults to true.) If set to Hide , the specified element(s) are assigned the inv:visible attribute with the value false .
xpath	yes	String. An xpath value that describes the element or elements that will be affected.

ToggleWidowOrphanControl

For the selected element, toggles the value of the **widow-control** attribute between **true** and **false**. Does not use parameters

Undo

Invokes Quark XML Author's Undo function. Does not use parameters.

Similar to Word, the Redo stack is cleared each time an action is pushed onto the Undo stack.

UpdateSchema

Dynamically changes the Quark XML Author Structure (XAS) and re-renders the document. This change only affects the current document. Instead of parameters, uses one or more child `<ElementDef>` nodes similar to the XAS definition element of the same name. For example, consider an `ElementDef` for a node that is initially visible in the document:

```
<ElementDef name="heading-instruction" friendly="Instruction"
visible="true" style="Heading Instruction">
  <!-- Element structure defined -->
</ElementDef>
```

To dynamically change the definition's visible value from **true** to **false**, the Internal Class would be defined as shown below:

```
<Internal Class="" name="UpdateSchema">
  <ElementDef name="heading-instruction" friendly="Instruction"
visible="false" style="Heading Instruction"/>
</InternalClass>
```

Note that the `<ElementDef>` node is a leaf node only.

The following `ElementDef` attributes may be changed: visible, style, excludeFromContextMenu.

WordDialogEdit

Invokes the Word dialog named in the `dialogName` parameter.

Table 5-39: WordDialogEdit Parameters

Parameter	Required	Definition
dialogName	yes	String. The Word dialog that should be invoked. For example, <code>wdDialogInsertSymbol</code> .

Application Configuration: Extensibility Interface

Quark XML Author was designed to be extensible, so that it can be easily integrated with custom and third party (external) applications. Quark XML Author communicates with external applications via the **Extensibility Interface (EI)** section of the application and document configuration files. Functions that are document type specific are specified in the DocConfig file. Functions that may be invoked for all document types are specified in the application-level configuration file (AppConfig).

External applications typically are in the form of a Dynamic Link Library (DLL) or an EXE. A DLL contains a set of functions, each of which has its own parameters. EI is the mechanism by which these functions are called. EI must be told the details of “when” and “how” a function is called.

This chapter covers the Extensibility Interface from two major viewpoints: programming external methods for use with Quark XML Author, and configuring Quark XML Author to make use of those external methods.



Section 17 of this manual, “Integration with Content Management Systems”, discusses integrating Quark XML Author with Content Management Systems, and contains several examples of creating Extensibility Interface methods. In addition, chapter 18, “Configuring Smart Paste”, contains an EI method example.

Programming for Quark XML Author

Quark XML Author can be configured to call any external application that exposes public methods. The person configuring Quark XML Author simply needs to know the assembly, class, and method name and what parameters the method expects. For example, the following method requires that an XML node and an array of Delegates be passed to it:

```
public bool IsEditableElement(XmlNode node, Delegate[] delegates)
{
    try
    {
        //read all the delegates here.
    }
}
```

```
//The namespaces for all delegates are
Quark.XA.ExtensibilityDelegates.
IsEditableElement isEditableElementDelegate = delegates[0] as
IsEditableElement;
if(isEditableElementDelegate != null && node != null && node
is XmlElement)
{
    // This delegate indicates whether a node is editable or
not.
    // This is based on values associated to a proprietary
    // Invision attribute called inv:access where
    // xmlns:inv="urn:xpressauthor:xpressdocument".This
attribute is usually
    // applied in another EI method. A typical example would be
a document coming
    // from a CMS system with certain nodes
    // tagged as non-editable, On an open, this EI method would
read the non-editable
    // tag from this document and add a Quark XML Author
    //tag (inv:access="read-only") identifying it as a readOnly
chunk.
    return isEditableElementDelegate(node);
}
}
catch(Exception e)
{
    System.Diagnostics.Debug.WriteLine("IsEditableElement
Exception :" + e.Message + "\nStackTrace:" + e.StackTrace);
}
return false;
}
```

As long as the assembly and class to which this method belongs are known, Quark XML Author can be configured to supply the XML node and any delegates that the method requires. The method would do its work and return a Boolean value to Quark XML Author to be acted upon.

Calling Quark XML Author from an External Method with Delegates

In addition to being able to call external methods, Quark XML Author exposes its own methods that can be called by external methods via delegates. Delegates are function pointers to Quark XML Author methods that can be passed on to external DLLs. The External DLLs may then call the methods to perform operations within the application. Custom methods can accept delegates by implementing the Delegate[] argument type.

Configuring the Extensibility Interface

Configuring Quark XML Author to use external methods is a matter of telling Quark XML Author how and when to call them. “How” is covered in Section 6.2.1, “Building the EI Method”. “When” is covered in Section 6.2.2, “Calling the EI Method”.

Building the EI Method

The ExtensibilityInterface node of both the AppConfig and DocConfig files contains the Method definitions that are used to call external applications. (It also can contain instructions about when to call them, which is covered in Section 6.2.2.1,

“Calling the EI Method from a Document Event”). The ExtensibilityInterface node contains a MethodInfo child element. MethodInfo in turn contains one or more Method nodes. The attributes of Method identify the external application to be called. The child nodes of Method provide the parameter values to be passed to the external application, as shown below:

```
<ExtensibilityInterface>

  <MethodInfo>

    <Method

      <Method id="name for the EI method" assembly=" external
application assembly name" class="external application class
name" method=" external application method name">

        <!-- Any number of <Argument> nodes -->

      </Method>

    </MethodInfo>

  </ExtensibilityInterface>
```

In the following subsections, we will explain the attributes and arguments and build an example EI method that calls the external method provided as an example in section 6.1.

Method Node Attributes

The attributes for the Method node define the .NET class library of the external process and how it is executed. Table 6-1 defines these attributes, all of which are mandatory.

Table 6-1: Method Attributes

Attribute	Definition
assembly	Specifies the file name of the .NET class library of the external process, without the .dll extension. For example, About.dll would be identified simply as About . A relative path to the assembly may be supplied if the assembly is placed in the root or a subdirectory of the Quark XML Author application folder.
class	Specifies the fully namespace-qualified class name within the .dll file (for example, Invision.Xpress.Extensible.MultiSelectPopup) being called for the external process.
id	Defines the name by which the method will be referenced.
method	Specifies the method executed when the class is called.

In the example below, we are calling a method named **InvokeMethodIdTest**. This is a method of **MiscellaneousDelegates.Misc**, which is provided in the sample Visual Studio .NET project file.

```
<Method id="IsEditableEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc" method="IsEditableElement">

<!-- Arguments to be added -->

</Method>
```

We have assigned the **id** value **IsEditableEI** to this EI method; whenever we invoke this method, the **id** value is used to reference it. The **id** value can be anything you want it to be provided that it is unique.

Method Node Arguments

Now that we've identified the external method to be called, we have to supply it with the parameter values it expects. Here is the signature for the method again:

```
public bool IsEditableElement(XmlNode node, Delegate[] delegates)
```

IsEditableElement expects to receive an XML node and at least one delegate. We will supply this data with a collection of **Argument** elements.

Enumerated Values

At its simplest, **Argument** is an empty element with one attribute: **type**. The value of **type** is an enumerated value that supplies information about the Quark XML Author document. For the example we're working with, we need to supply an XML node to the external class. Quark XML Author has several enumerated values that supply XML fragments. The one we need to use here is **XomCurrentNode**, which provides the document node corresponding to the user's current selection. We would add an **Argument** element to our method definition that names **XomCurrentNode** as the argument type, as shown in the example below:

```
<Method id="IsEditableEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc" method="IsEditableElement">

  <Argument type="XomCurrentNode"/>

</Method>
```

A complete list of the Enumerated Value names that Quark XML Author makes available is provided in section 6.3, "List of Available Enumerated Values".

Delegates

As mentioned previously, delegates are function pointers to Quark XML Author methods that can be passed on to external DLLs. The external method sample we are using expects to be passed a delegate that it can use to determine if the node it has been passed may be edited. We'll supply this delegate with another **Argument** node.

In this case, the type attribute for the Argument node is set to **Delegates**, and Argument contains a collection of one or more Delegate nodes:

```
<Method id="IsEditableEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc" method="IsEditableElement">

  <Argument type="XomCurrentNode"/>

  <Argument type="Delegates">

    <!-- One or more <Delegate> nodes go here -->

  </Argument>

</Method>
```

The content of the Delegate node is the name of the delegate to be supplied. For this example, we will supply the `IsEditableElement` delegate, shown below:

```
<Method id="IsEditableEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc" method="IsEditableElement">

  <Argument type="XomCurrentNode"/>

  <Argument type="Delegates">

    <Delegate>IsEditableElement</Delegate>

  </Argument>

</Method>
```

The method is complete.

A complete list of the delegates that Quark XML Author makes available, along with the signatures that must be used when calling them, is provided in section 6.4, “List of Available Delegates”.

Tokens

In some cases, neither enumerated values nor delegates will be adequate to supply the correct information to the external class. In these cases, you set the Argument type value to **Tokens**. With type set to **Tokens**, the Argument node must then contain one or more Token child nodes. The content of the Token node can be any data that needs to be supplied to the external application.

For example, consider the following external method, which belongs to the same assembly and class as the previous example:

```
public void AssignSingleAttributeTest(XmlNode xomNode, string[]
tokens)

{

    if(xomNode is XmlElement && tokens != null && tokens.Length
> 0 && xomNode.Attributes.Count > 0)

    {
```

```
        XmlAttribute attr = xomNode.Attributes[tokens[0]];

        if(attr != null)
        {
            //Simply display the attribute value in the
            Console. This could be more complex like displaying
            value.
            //a custom form for choosing/changing an attribute
            value.

            Console.WriteLine("Attribute Name:" + attr.Name + "
            Value:" + attr.Value);
        }
    }
}
```

This method requires an XML node and tokens to be supplied to it. The EI method would be constructed similarly to the one we constructed previously, but the second Argument node type would be set to **Tokens**:

```
<Method id="AssignSingleAttributeEI"
assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc"
method="AssignSingleAttributeTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Tokens">

        <!-- Token elements go here -->

    </Argument>

</Method>
```

In this case, the external method is being used to get an attribute value from a custom Quark XML Author control, so we need to assign the attribute name as the content of a Token node:

```
<Method id="AssignSingleAttributeEI"
assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc"
method="AssignSingleAttributeTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Tokens">

        <Token>inv:attr1</Token>

    </Argument>

</Method>
```

The content of Token can be any string value or an xml fragment required by the external application.

Tag

Tag is a special Argument type that supplies the label of an item selected in the user interface. It is typically used with EI methods that intercept an attempt to insert an element by the user. For example, if the following method was invoked when the user clicked an element with the label **Section**, the string Section would be passed to the external method:

```
<Method id="getImage" assembly="ImageHandler" class="
ImageHandler.InsertElementManager" method="GetImage">

  <Argument type="Tag"/>

</Method>
```

Tag is used in the case study presented in Chapter 21.

Calling the EI Method

Once you have defined a method in the Extensibility Interface, it still won't do anything unless it is invoked. EI methods may be invoked through a document event, through a Quark XML Author command (menu item, commandbar button, or shortcut key) defined in the application or document configuration files, or by an element, emphasis, or attribute definition in the Quark XML Author Structure

Calling the EI Method from a Document Event

Previously, we have discussed the MethodInfo node of the ExtensibilityInterface element in the configuration files. However, ExtensibilityInterface can also contain any number of nodes named for document events. These nodes allow you to specify EI methods to be called when document events are triggered.

To invoke an EI method from a document event, include an ExtensibilityMethod element as a child of the document event element. The ExtensibilityMethod element has one required attribute: id. The value of the id attribute is the name of an EI Method defined in the MethodInfo node.

For example, suppose we had defined a method that looks for document variables called **header** and **footer** in the Word document and sets them to a specified value:

```
<Method id="AssignDocVariableEI" assembly="UIManipulation"
class="UIManipulation.UIDelegates"
method="AssignDocVariableTest">

  <Argument type="Tokens">

    <!-- the header and footer are doc variables defined in the
dot file of the document. -->

    <!-- In this simple example, the docVariables are set to a
hardcoded value. -->

    <Token>header=This is the Header</Token>

    <Token>footer=This is the footer</Token>

  </Argument>
```

```
<Argument type="Delegates">

    <Delegate>AssignDocVariable</Delegate>

    <Delegate>RefreshDocVariables</Delegate>

</Argument>

</Method>
```

Now suppose that we wanted to call this method whenever the document was saved. We would add the following node as a child of the ExtensibilityInterface element:

```
<Save>

    <ExtensibilityMethod id="AssignDocVariableEI"/>

</Save>
```

When the document is saved, the AssignDocVariableEI method would be invoked before the Save completes. A document event can contain an unlimited number of ExtensibilityMethod child elements, each of which would call a different EI method.

A complete list of available document events is provided in section 6.5, "List of available Document events".

Calling the EI Method from a Quark XML Author User Command

Menu items, commandbar buttons, and shortcut keys can all be configured to invoke EI methods. Section 4 of this manual discusses the use of the ExtensibilityMethod child element for each type of user interface component, but for ease of reference, those instructions are summarized here.

To invoke an EI method from a menu item, commandbar button, or a shortcut key combination, include an ExtensibilityMethod element as a child of the appropriate element in the configuration file. The ExtensibilityMethod element has one required attribute: id. The value of the id attribute is the name of an EI Method defined in the MethodInfo node. In each example below, an EI Method name SampleTableImport is invoked:

From a menu item:

```
<MenuItem resourceId="Table Import">

    <ExtensibilityMethod id="SampleTableImport"/>

</MenuItem>
```

From a commandbar button:

```
<CommandBarButton resourceId="Table Import">
```

```
<ExtensibilityMethod id="SampleTableImport"/>

</CommandBarButton>
```

From a shortcut key:

```
<ShortcutKey Key="T" Shift="true" Ctrl="true">

  <ExtensibilityMethod id="SampleTableImport"/>

</ShortcutKey>
```

For more information on configuring menu items, commandbar buttons, and shortcut keys, see Chapter 4 of this manual.

Multiple EI Methods from a Single Command

You may call multiple EI methods from a single command by adding a separate `ExtensibilityMethod` node for each method you wish to call. Keep in mind, however, that if an EI method returns a value of **false** or throws an exception, any subsequent EI method calls will be cancelled.

Calling the EI Method from an Element or Emphasis Definition

With some elements or emphasis styles (which are simply in-line elements), you may want to provide a way for users to access a tool such as a metadata wizard via a command in the context menu. Element and Emphasis definitions may use Extensibility methods to provide this functionality.

To use EI methods with either an element definition or emphasis definition, add an `ExtensibilityMethods` child element to the definition. This node will contain one or more `ExtensibilityMethod` elements, one for each EI method to be called.

The syntax for using Extensibility Methods in these contexts is shown below:

```
<ExtensibilityMethods>

  <ExtensibilityMethod id="method name" friendly="friendly name"
    showInComponentContextMenu="true or false"
    showInContextMenu="true or false" faceID="#" />

</ExtensibilityMethods>
```

Note that for element and emphasis definition, the `ExtensibilityMethod` element has more attributes available to it. They are defined in Table 6-2.

Table 6-2: ExtensibilityMethod Attributes

Attribute Name	Required	Definition
toggleXPath	no	Toggle the state of the 'toggle button' based on the xpath mentioned in the attribute.
enableXPath	no	Value is an XPath expression which evalutes to a node-set in

Attribute Name	Required	Definition
		<p>which false is returned for an empty node-set and true otherwise. Used to toggle availability of menu items, commandbar buttons, and shortcut keys. If the value evaluates to false, the item is disabled, or enabled if the value evaluates to true.</p> <p>Namespaces are not supported in the xpath syntax.</p> <p>For the Ribbon, enableXPath evaluates against root nodes. Against current node is not supported.</p>
faceID	no	Allows you to specify an icon that will be displayed next to the context menu item. The value of faceId is an integer corresponding to the ID number of an icon in the Word template FaceID.dot.
friendly	no	Specifies a name for the context menu command.
id	yes	Specifies the ID of the EI method to be called.
showInComponentContextMenu	no	Defaults to true . When set to false , the command will not appear in the Component context menu.
showInContextMenu	no	Defaults to true . When set to false , the command will not appear in the main context menu, but may still appear in the Component context menu if showInComponentContextMenu is set to true .
showXPath	no	Value is an XPath expression which evaluates to a node-set, and when applied to the current element, indicates whether to include the method on the context menu: if a node-set is returned, the method is included; if an

Attribute Name	Required	Definition
		<p>empty node-set is returned, the method is omitted. The XPath expression is applied to the XOM therefore the names are XOM element names. The XPath expression will be evaluated against the current runtime element and thus must be relative to it.</p> <p>Namespaces are not supported in the xpath syntax. For example, showXPath="self::node()[local-name()='Section']".</p> <p>XML Author does not support dynamic show/hide of Ribbon items, therefore showXPath is not applicable for Ribbon items.</p>
prependElementFriendly	No	<p>Boolean. Defaults to true. When set to false, the element friendly name will not be appended before the extensibility method name.</p>

See section 13 for information on how to define Emphasis and section 14 for information about how to define Elements.

Calling the EI Method from an Attribute Definition

With some document element attributes, you may want to provide a way for users to access a tool such as a metadata wizard. Attribute definitions may invoke an EI method to provide this functionality. Simply add the externalMethodId attribute to the AttributeDef element in the XAS file. The value of the attribute is the id of the EI method. An example is shown below:

```
<AttributeDef namespace="www.invisionresearch.com" prefix="inv"
name="attr1" externalMethodId="AssignSingleAttributeEI"
visible="true"/>
```

Calling the EI Method from an Element Definition

The ElementDef element has an externalMethodId attribute which can define an EI for inserting new elements. For such definitions, the insertable context menu

displays a special icon to indicate that this is an EI insert. See “Table 14-1” for information on this attribute

Using One EI Method to Call Another

You can also call EI methods from the external application. In the example below, the InvokeAnotherEI method passes the name of another EI method (JustAnotherEI) to the external method it calls:

```
<Method id="InvokeAnotherEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc" method="InvokeMethodIdTest">

  <Argument type="Tokens">

    <Token>JustAnotherEI</Token>

  </Argument>

  <Argument type="Delegates">

    <Delegate>InvokeMethodId</Delegate>

  </Argument>

</Method>
```

Here’s the second EI method:

```
<Method id="JustAnotherEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc" method="InvokeTest">

  <Argument type="XomCurrentNode"/>

  <Argument type="Delegates">

    <Delegate>SelectNode</Delegate>

  </Argument>

</Method>
```

The first EI method invokes this external method:

```
public void InvokeMethodIdTest(string[] tokens, Delegate[]
delegates)
{
    InvokeMethodId invokeMethodIdDelegate = delegates[0] as
InvokeMethodId;

    if(invokeMethodIdDelegate != null && tokens.Length > 0 &&
tokens[0] != String.Empty)
    {

        //This delegate simply calls another EI method defined in
the config files.

        //The name for the method comes from a token. In this case
we are invoking the InvokeTest EI method.
```

```

        invokeMethodIdDelegate(tokens[0]);
    }
}

```

This method receives the token value **JustAnotherEI** and the **InvokeMethodId** delegate. It then uses **InvokeMethodId** to call the EI method named in the token, which is the second EI Method. That EI method invokes an entirely different external method:

```

public void InvokeeTest(XmlNode currentNode, Delegate[]
delegates)
{
    SelectNode selectNodeDelegate = delegates[0] as SelectNode;
    if(selectNodeDelegate != null && currentNode != null)
    {
        selectNodeDelegate(currentNode);
    }
}

```

And this method selects the node passed to it by the **XomCurrentNode** enumerated value.

You can daisy-chain any number of EI methods and external methods in this manner should you need to.

Use Cases

For additional use cases and examples see “Extensibility Interface Use Case Study”.

List of Available Enumerated Values

The enumerated values defined in Table 6-3 may be used as Extensibility Interface argument types. Several enumerated values refer to the XOM, which is explained following the table.

Table 6-3: Enumerated Values

Enumerated Value	Type	Definition
AccessMode	string	The access mode for the current document. AccessMode is available in an (embedded) OLE Word Document. When an OLE Word Document is the active document, this is the access mode of the parent XML Author document that owns

Enumerated Value	Type	Definition
		the OLE Word Document.
DeleteNodes	2-d array of DeleteNode-ParentNode pairs: XmlNode[element,parent]	<p>Special argument type that is only valid for an EI method specified under the BeforeComponentDelete and ComponentDelete events. It holds a list of elements that either will be deleted (when used with BeforeComponentDelete) or have been deleted (when used with ComponentDelete) and their parent nodes. For example, suppose you were to delete <Title/> and <Para/> from:</p> <pre><Section> <Title/> <Para/> </Section></pre> <p>DeleteNodes would supply an array containing the following:</p> <ul style="list-style-type: none"> • Title-Section • Para-Section <p>Note that when DeleteNodes is used as an argument for an EI method under BeforeComponentDelete, the first element in each pair is still part of the XOM.</p> <p>When DeleteNodes is used with ComponentDelete, the first element in each pair is no longer part of the XOM.</p>
DocNumber	Int	The repository document number for a checked out document. A new document will have a DocNumber of 0. Used by Xpress Server[1].
DocumentType	string	This argument may be assigned a value by the Xpress Server based on taxonomy (document classification) or catalog

Enumerated Value	Type	Definition
		information. For example, this would allow a process to differentiate between 'Document' and 'Template' document types.
ExportedCurrentNode	XML Node	With this argument type, only the currently selected element is serialized. In other words, a new XmlNode is created that will contain element content, but will not have references to parent or child nodes.
ExportedCurrentNodeTree	XML Node	This argument has the same characteristics as ExportedCurrentNode, but also includes all descendent elements.
ExportedRoot	XML Node	Indicates a schema-valid XML document. This document contains element content and is fully de-referenced including CALS-standard tables. Because this argument type requires normalizing the Quark XML Author and Word object models, it may require processing time for larger documents.
Filename	string	Provides an EI method with the currently active document's full name, including paths.
PasteNode	XML Node	Special argument type used exclusively by any BeforePaste and InsertComponent EI methods. Contains the node being pasted (for any EI methods in BeforePaste event) or the node after it has been pasted (for any EI methods inside InsertComponent event).
Revision	int	The repository revision number for a checked out document. When a document is checked in, it will be assigned a Revision 1 greater

Enumerated Value	Type	Definition
		than this value. A new document will have a Revision of 0. Used by Xpress Server.
StyleList	ArrayList of strings	Special argument type used exclusively by the BuildStyleList EI methods. Contains a list of currently insertable elements. Used to facilitate business rules that cannot be supported by configuration. For example, a business rule states that only one instance of a certain content element can exist in a given document.
ToggleState	boolean	Special argument type used exclusively for external method wired to a toggle menu item. Indicates the current state of the toggle button. Pressed=true Depressed=false)
UserName	string	The username of the individual authoring or revising the document as specified in the document header. As is the case with all Quark XML Author header information, it is populated by the Xpress Server (servlet).
WordUserName	string	The UserName property of the Word application.
XomCurrentNode	XML Node	The XOM node corresponding to the user's current selection. If the current selection is on an empty style or it spans multiple paragraphs, XomCurrentNode will be null. See section 6.3.1, "XOM Defined" for a definition of XOM. XOMCurrentNode is available in an (embedded) OLE Word Document. When an OLE Word Document is the active

Enumerated Value	Type	Definition
		document, the XOM node corresponds to a node within the Psuedo XOM of the OLE Word Document.
XomRoot	XML Node	The XOM node corresponding to the current document root. See section 6.3.1, “XOM Defined” for a definition of XOM. XOMRoot is available in an (embedded) OLE Word Document. When an OLE Word Document is the active document, the XOM node corresponds to the root of the parent XML Author document that owns the OLE Word Document.
XomSelectedNodes	XML Node	The XOM nodes corresponding to the user’s current selection when the selection spans multiple elements. Note that only nodes with a defined relationship to the external method are included. To define a relationship with the external method, the method must be referenced in the ElementDef’s ExtensibilityMethods section. See Section 14.10 for more information about the ExtensibilityMethods section of ElementDef. See section 6.3.1, “XOM Defined” for a definition of XOM.
XomPreviousNode	XML Node	The XOM node corresponding to the previous visible node if selection is on an empty. Otherwise null. See section 6.3.1, “XOM Defined” for a definition of XOM.
StyleListParents	Arraylist of XML Node	A special argument type to be used exclusively by the BuildStyleList EI methods. This contains a list of parents of currently insertable elements provided by the styleList

Enumerated Value	Type	Definition
		argument with a one-on-one linkage with stylelist items. This argument can help the EI developer to distinguish between insertables under different parents with the same name.

XOM Defined

XOM means Quark XML Author Object Model. It is the Quark XML Author XML DOM that is used to manage the structure of a document and its attributes. The XOM includes reference nodes and their de-referenced descendants. XOM contains only the document structure. It does not contain any element content. All element content is managed in the Microsoft Word Object Model. A custom process provided with an `<Argument type="XomRoot"/>` may traverse the current document's structure, as well as browse and/or manipulate attribute values.

XOM and OLE Word Documents

XOMCurrentNode is available in an (embedded) OLE Word Document. XOMRoot is available in an (embedded) OLE Word Document.

See "Table 6-3: Enumerated Values".

List of Available Delegates

The available delegates and their signatures are defined below.

AssignDocVariable

```
public delegate void AssignDocVariable(string docVariableName, string
docVarValue)
```

For the given document variable, supplies the string variable specified.

Configuration Example

```
<!-- This method looks for doc variables called "header" and
"footer" in the Word document and sets them to a specific value--
>

<Method id="AssignDocVariableEI" assembly="UIManipulation"
class="UIManipulation.UIDelegates"
method="AssignDocVariableTest">

  <Argument type="Tokens">

    <!-- the header and footer are doc variables defined in the
dot file of the document. -->

    <!-- In this simple example, the docVariables are set to a
```

```

hardcoded value. -->

    <Token>header=This is the Header</Token>

    <Token>footer=This is the footer</Token>

</Argument>

<Argument type="Delegates">

    <Delegate>AssignDocVariable</Delegate>

    <Delegate>RefreshDocVariables</Delegate>

</Argument>

</Method>

```

AssignPostText

```

public delegate void AssignPostText(XmlNode visibleNode, string
postTextString)

```

Assigns PostText portionmark values for a given visible node.

Configuration Example

```

<!-- PreText delegates test method definition -->

<!-- This method assigns a randomly generated string to the
PreText area of a visible node-->

<Method id="PreTextEI" assembly="UIManipulation"
class="UIManipulation.UIDelegates" method="PreTextPostTextTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Delegates">

        <Delegate>GetNodeProperties</Delegate>

        <Delegate>GetPreText</Delegate>

        <Delegate>AssignPreText</Delegate>

        <Delegate>GetPostText</Delegate>

        <Delegate>AssignPostText</Delegate>

        <Delegate>RefreshDocVariable</Delegate>

    </Argument>

</Method>

```

AssignPreText

```

public delegate void AssignPreText(XmlNode visibleNode, string
preTextString)

```

Assigns PreText portionmark values for a given visible node.

Configuration Example

```

<!-- PreText delegates test method definition -->

<!-- This method assigns a randomly generated string to the
PreText area of a visible node-->

<Method id="PreTextEI" assembly="UIManipulation"
class="UIManipulation.UIDelegates" method="PreTextPostTextTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Delegates">

        <Delegate>GetNodeProperties</Delegate>

        <Delegate>GetPreText</Delegate>

        <Delegate>AssignPreText</Delegate>

        <Delegate>GetPostText</Delegate>

        <Delegate>AssignPostText</Delegate>

        <Delegate>RefreshDocVariable</Delegate>

    </Argument>

</Method>

```

AssignTextNode

```

public delegate void AssignTextNode(XmlNode visibleParaNode,
string fragment, TriValue enableTrackChanges)

```

Replaces the content of the visibleParaNode with the contents of the fragment. Only the text portion of the node (and any emphasis applied to that text) is replaced. Any other child elements of the visibleParaNode are left intact. The value of enableTrackChanges indicates whether to record TrackChanges for the change. The possible values are Quark.XA.ExtensibilityDelegates.TriValue.False (do not record TrackChanges), Quark.XA.ExtensibilityDelegates.TriValue.True (record TrackChanges), and Quark.XA.ExtensibilityDelegates.TriValue.NoValue (the current value of TrackChanges for the document is used).

Configuration Example

```

<!-- AssignTextNode delegate test method definition -->

<!-- This method replaces the text in the XomCurrentNode with
text from the location specified in the token-->

<Method id="AssignTextEI" assembly="XomManipulation"
class="XomManipulation.XomDelegates" method="AssignTextNodeTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Tokens">

        <!-- location of the text chunk which is being inserted -->

```

```

    <Token>c:\temp\AssignTextNodeTest.txt</Token>

  </Argument>

  <Argument type="Delegates">

    <Delegate>AssignTextNode</Delegate>

  </Argument>

</Method>

```

Configuration Example

```

<Method id="UndoRecordingTest" assembly="UndoRecording"
class="ExtensibilityInterface.Delegates.UndoRecording"
method="UndoRecordingTest">

  <Argument type="XomCurrentNode" />

  <Argument type="Delegates">

    <Delegate>AssignTextNode</Delegate>

    <Delegate>BeginUndoRecording</Delegate>

    <Delegate>EndUndoRecording</Delegate>

  </Argument>

</Method>

```

BeginUndoRecording

```
public delegate void BeginUndoRecording()
```

Begins recording undo objects.

Configuration Example

```

<Method id="UndoRecordingTest" assembly="UndoRecording"
class="ExtensibilityInterface.Delegates.UndoRecording"
method="UndoRecordingTest">

  <Argument type="XomCurrentNode" />

  <Argument type="Delegates">

    <Delegate>AssignTextNode</Delegate>

    <Delegate>BeginUndoRecording</Delegate>

    <Delegate>EndUndoRecording</Delegate>

  </Argument>

</Method>

```

CreateContextMenu

```
public delegate void CreateContextMenu(XmlNode node,
ContextMenuItem[] items);
```

Supplies the list of items to be placed in a context menu. Used typically for custom context menus in post-text and pre-text portionmark areas.

Configuration Example

```
<!-- EI Method tests CreateContextMenu - the EI will use the
current xom node and build the context menu. For this to work -
assign a shortcut key to the ribbon button-->

<Method id="CreateContextMenuTest" assembly="CreateContextMenu"
class="ExtensibilityInterface.Delegates.CreateContextMenuTest"
method="TestCreateContextMenu">

  <Argument type="XomCurrentNode"/>

  <Argument type="Tokens">

    <Token>Corporate Governance</Token>

    <Token>Environment, Safety & Health</Token>

    <Token>Facilities</Token>

    <Token>Finance</Token>

    <Token>Human Resources</Token>

    <Token>Information Management</Token>

    <Token>Integrated Safeguards & Security</Token>

    <Token>Mission Execution</Token>

    <Token>Supply Chain Management</Token>

  </Argument>

  <Argument type="Delegates">

    <Delegate>CreateContextMenu</Delegate>

  </Argument>

</Method>
```

DeleteNode

```
public delegate void DeleteNode(XmlNode nodeToBeDeleted, bool
displayWarning = false);
```

Delete the node specified. This operation will also delete any schema-dependent nodes. For instance, if the node to be deleted is a required child of a parent as defined in the schema, the parent and all its other siblings will be deleted. Thus, the operation will fail if the final delete root happens to be the root of the document.

DeleteNode Parameters

Parameter	Required	Definition
nodeToBeDeleted	yes	XmlNode. The XOM Node to be Deleted.
displayWarning	no	Boolean. Displays a warning before the node is deleted. Defaults to false.

Configuration Example

```

<!-- DeleteNode delegate test method definition -->

<!-- This method tries to delete the incoming node -->

<Method id="DeleteEI" assembly="XomManipulation"
class="XomManipulation.XomDelegates" method="DeleteNodeTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Delegates">

        <Delegate>DeleteNode</Delegate>

    </Argument>

</Method>

```

EmphasisAction

```

public delegate void EmphasisAction(XmlNode node, string
emphasisName, string subStyleKey, string subStyleValue)

```

Used to apply emphasis.

For more information, see InternalClass"EmphasisAction".

The delegate is available for use in XML Author documents and embedded Word documents.

Configuration Example

```

<Method id="TestEmphasisAction" assembly="XOMCurrentTest"
class="XOMCurrentTest.XOMTest" method="TestEmphasisAction">

    <Argument type="XomCurrentNode"/>

    <Argument type="Delegates">

        <Delegate>EmphasisAction</Delegate>

    </Argument>

    <Argument type="Tokens">

        <Token>emphasisName=example</Token>

        <!-- provide the name of an EmphasisDef that exists as child
to the OLEWordDocument/ParaType/Emphasis -->

    </Argument>

```

```
</Method>
```

EndUndoRecording

```
public delegate void EndUndoRecording()
```

Ends registering undo objects and also registers a compound Undo for all operations that took place so that they can be undone with one Undo call.

Configuration Example

```
<Method id="UndoRecordingTest" assembly="UndoRecording"
class="ExtensibilityInterface.Delegates.UndoRecording"
method="UndoRecordingTest">
```

```
  <Argument type="XomCurrentNode" />
```

```
  <Argument type="Delegates">
```

```
    <Delegate>AssignTextNode</Delegate>
```

```
    <Delegate>BeginUndoRecording</Delegate>
```

```
    <Delegate>EndUndoRecording</Delegate>
```

```
  </Argument>
```

```
</Method>
```

FormattingAction

```
public delegate bool FormattingAction(string formattingName,
string colorValue)
```

Used to apply/remove character formatting, font color and highlight color on current selection.

For more information, see InternalClass"FormattingAction".

The delegate is available for use in XML Author documents.

GetExportedNode

```
public delegate XmlNode GetExportedNode(XmlNode xomNode)
```

Returns the exported node for the specified XOM node.

Configuration Example

```
<!-- EI Method tests GetExportedNode, and
GetExtensibldUserInterface - should display xml of current xom
node in Word taskpane -->
```

```
<Method id="GetExtensibleUserInterfaceTest"
assembly="ExtensibleUserInterface"
class="ExtensibilityInterface.Delegates.ExtensibleUserInterface"
method="GetExtensibleUserInterfaceTest">
```

```
  <Argument type="XomCurrentNode" />
```

```

<Argument type="Delegates">

    <Delegate>GetExportedNode</Delegate>

    <Delegate>GetExtensibleUserInterface</Delegate>

</Argument>

</Method>

```

GetExtensibleUserInterface

```

public delegate IExtensibleUserInterface
GetExtensibleUserInterface(ExtensibleUserInterfaceType type,
UserController control, string title, Guid guid = null, DockPosition
dockPosition = DockPosition.dockPositionRight, bool
createIfNotFound = true);

```

For a given custom UserControl, returns an object implementing the ITaskpane interface which provides the incoming control with a parent form. Available parameters provide support to show multiple custom task panes simultaneously, set default dock position and to get an Extensible User Interface (EUI) object that has already been created.

GetExtensibleUserInterface Parameters

Parameter	Required	Definition
type	yes	ExtensibleUserInterfaceType. Permissible values are Taskpane and Dialog.
control	yes	UserController. The control to be added to the task pane.
title	yes	String. The title of ExtensibleUserInterfaceType.
guid	no	GUID. An EI can now specify a GUID to create a separate task pane. If no GUID is specified, the task pane will be created as a legacy task pane. Defaults to Null.
DockPosition	no	String. the default dock position can be specified when creating the task pane. Defaults to DockPosition.dockPositionRight.
createIfNotFound	no	Boolean. If true, it will create a new EUI if not already created. If false, it will get the already created EUI. Defaults to true.

Example :

```
GetExtensibleUserInterface(ExtensibleUserInterfaceType.Taskpane,  
taskPane, "Navigation Pane", guid, DockPosition.dockPositionLeft,  
true);
```

GetNextNode

```
public delegate XmlNode GetNextNode(XmlNode node)
```

Returns the next visible node in the document as an XML node. If provided node is null, returns the first visible node in the document.

Configuration Example

```
<!-- GetNextNode delegate test method definition -->  
  
<Method id="GetNextNodeEI" assembly="Doc_Navigation_Selection"  
class="Doc_Navigation_Selection.Selection_Navigation"  
method="GetNextNodeTest">  
  
  <Argument type="XomCurrentNode"/>  
  
  <Argument type="Delegates">  
  
    <Delegate>GetNextNode</Delegate>  
  
    <Delegate>SelectNode</Delegate>  
  
  </Argument>  
  
</Method>
```

GetNodeProperties

```
public delegate void GetNodeProperties(XmlNode node, Hashtable  
ht)
```

Gets all properties defined for an attribute, element, or emphasis in the schema. Currently the following properties are supported: NodeType (NodeType identifies the type of incoming element-Element, Attribute or Emphasis), namespace, prefix, xmlname, friendly, style, defaultHyperlink, displayAttribute, displayElement, insertAfter, increaseIndent, decreaseIndent, excludeFromContextMenu, keyCode (only for NodeType=Element, contains keyCode - the actual value if it is visible or -1 if it is not), submenu, contextId, Unmanaged, UniqueIdentifier, externalMethodId, default, numbering, visible, readonly, datatype, format, attrGroupName, required, fixed, and contextFriendly. The contextFriendly property is the friendly name that is displayed in the context menu/style list.

Configuration Example

```
<!-- PreText delegates test method definition -->  
  
<!-- This method assigns a randomly generated string to the  
PreText area of a visible node-->  
  
<Method id="PreTextEI" assembly="UIManipulation"  
class="UIManipulation.UIDelegates" method="PreTextPostTextTest">
```

```

<Argument type="XomCurrentNode"/>

<Argument type="Delegates">

    <Delegate>GetNodeProperties</Delegate>

    <Delegate>GetPreText</Delegate>

    <Delegate>AssignPreText</Delegate>

    <Delegate>GetPostText</Delegate>

    <Delegate>AssignPostText</Delegate>

    <Delegate>RefreshDocVariable</Delegate>

</Argument>

</Method>

```

GetPageNumber

```
public delegate string GetPageNumber(XmlNode node)
```

Returns the page number for the start of the given XML node.

Configuration Example

```

<!-- EI Method tests GetPageNumber - should display pagenumber of
current xom node in a MessageBox -->

<Method id="GetPageNumberTest" assembly="GetPageNumber"
class="ExtensibilityInterface.Delegates.PageNumberTest"
method="GetPageNumberTest">

    <Argument type="XomCurrentNode" />

    <Argument type="Delegates">

        <Delegate>GetPageNumber</Delegate>

    </Argument>

</Method>

```

GetPostText

```
public delegate void GetPostText(XmlNode node, ref string
prevPostTextValue)
```

Retrieves the value of a PostText portionmark for a given node and stores it in the incoming prevPostTextValue parameter.

Configuration Example

```

<!-- PreText delegates test method definition -->

<!-- This method assigns a randomly generated string to the
PreText area of a visible node-->

<Method id="PreTextEI" assembly="UIManipulation"
class="UIManipulation.UIDelegates" method="PreTextPostTextTest">

    <Argument type="XomCurrentNode"/>

```



```

    <Argument type="Delegates">

        <Delegate>GetNodeProperties</Delegate>

        <Delegate>GetPreText</Delegate>

        <Delegate>AssignPreText</Delegate>

        <Delegate>GetPostText</Delegate>

        <Delegate>AssignPostText</Delegate>

        <Delegate>RefreshDocVariable</Delegate>

    </Argument>

</Method>

```

GetPreText

```
public delegate void GetPreText(XmlNode node, ref string
prevPreTextValue)
```

Retrieves the value of a PreText portionmark for a given node and stores it in the incoming prevPreTextValue parameter.

Configuration Example

```

<!-- PreText delegates test method definition -->

<!-- This method assigns a randomly generated string to the
PreText area of a visible node-->

<Method id="PreTextEI" assembly="UIManipulation"
class="UIManipulation.UIDelegates" method="PreTextPostTextTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Delegates">

        <Delegate>GetNodeProperties</Delegate>

        <Delegate>GetPreText</Delegate>

        <Delegate>AssignPreText</Delegate>

        <Delegate>GetPostText</Delegate>

        <Delegate>AssignPostText</Delegate>

        <Delegate>RefreshDocVariable</Delegate>

    </Argument>

</Method>

```

GetPreviousNode

```
public delegate XmlNode GetPreviousNode(XmlNode node)
```

Returns the previous visible node in the document as an XML node. If provided node is null, the last visible node is returned.

Configuration Example

```
<!-- GetPreviousNode delegate test method definition -->

<Method id="GetPreviousNodeEI"
assembly="Doc_Navigation_Selection"
class="Doc_Navigation_Selection.Selection_Navigation"
method="GetPreviousNodeTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Delegates">

        <Delegate>GetPreviousNode</Delegate>

        <Delegate>SelectNode</Delegate>

    </Argument>

</Method>
```

GetSectionMetadata

```
public delegate XmlElement GetSectionMetadata(XmlNode
visibleNode, bool getWordML)
```

Returns the metadata for the Word Section to which the provided visible node belongs.

The visibleNode has to be a visible XOM node. The getWordML boolean flag indicates whether WordML for the header/footer node must also be returned.

The format of the node that is returned is shown below:

```
<WordSection orientation="portrait"
breakType="sectionBreakNextPage">

    <!-- orientation = portrait|landscape
breakType=sectionBreakNextPage| sectionBreakContinuous -->

    <Headers>

        <!-- max of 3 Header nodes one each of type
primaryPageHeader, firstPageHeader and evenPageHeader are
possible-->

        <Header type="primaryPageHeader">

            <Fields>

                <!-- Each field corresponds to Word's field - type is the
field type, name is the field name, value is the value held by
this feild -->

                <Field type="wdFieldPage" name="PAGE \* MERGEFORMAT"
value="2"/>

                <Field type="wdFieldNumPages" name="NUMPAGES \*
MERGEFORMAT" value="1"/>
```

```

        <Field type="wdFieldDocVariable" name="DOCVARIABLE pf
\* MERGEFORMAT" value="pf Number"/>

    </Fields>

    <!-- This is the text in the header/footer range-->

    <Text>Range text content</Text>

    <!-- This is a WordML representation of header/footer
content-->

    <WordML>WordML content</WordML>

</Header>

</Headers>

<Footers>

    <!-- max of 3 Footer nodes one each of type
primaryPageFooter, firstPageFooter and evenPageFooter are
possible-->

    <Footer type="primaryPageFooter">

        <Fields>

            <Field type="wdFieldDocVariable" name="xxx" value="yyy"/>

        </Fields>

        <Text> Range text content</Text>

        <WordML>WordML Content</WordML>

    </Footer>

</Footers>

</WordSection>

```

If the `getWordML` flag is **false**, the WordML nodes in bold above are absent. This is done because retrieving WordML from headers and footers is a potentially expensive operation and should only be done when it is required to parse any header/footer content that is not in the `<Text>` node or the `<Fields>` node.

GetToggleButtonState

```
public delegate bool GetToggleButtonState(string
externalMethodId)
```

Returns the state (true or false) of the toggle button specified by the given external method ID.

Configuration Example

```

<Method id="GetToggleButtonState" assembly="GetToggleButtonState"
class="ExtensibilityInterface.Delegates.ToggleButtonStateTest"
method="GetToggleButtonStateTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Delegates">

```

```

        <Delegate>SelectNode</Delegate>

        <Delegate>GetToggleButtonState</Delegate>

    </Argument>

</Method>

```

GetXmlResolver

```
public delegate IXmlResolverProvider GetXmlResolver();
```

GetXmlResolver is used to return the current instance of the IXmlResolverProvider class used by Quark XML Author – it could be the default one used by Quark XML Author or an external one provided by the SetXmlResolver delegate.

Information about creating and using custom resolvers can be found in Section 0.

Configuration Example

```

<!-- EI Method tests GetXmlResolver - if provider is successfully
obtained message box will indicate that -->

<Method id="GetXmlResolverTest" assembly="GetXmlResolver"
class="ExtensibilityInterface.Delegates.TestXmlResolver"
method="GetXmlResolverTest">

    <Argument type="Delegates">

        <Delegate>GetXmlResolver</Delegate>

    </Argument>

</Method>

```

GetXomNode

```
public delegate XmlNode GetXomNode(XmlNode exportedNode)
```

Returns the XOM node for the specified exported node.

An external process manipulates a document by modifying the XOM. This delegate is used to create the XOM that represents the exportedNode.

Do not use the document's CurrentExportedNode as the input parameter. Instead, you must use a node within the document's ExportedRoot.

A use case of GetXOMNode is to calculate the CRC for the exportedNode , GetXOMNode of the exportedNode and then store that CRC in the XOM.

Configuration Example

```

<!-- EI Method tests GetXomNode - should display xml of xom node
that corresponds to the first 'p' element of the exported root
node in a MessageBox -->

<Method id="GetXomNodeTest" assembly="GetXomNode"
class="ExtensibilityInterface.Delegates.XomNodeTest"
method="GetXomNodeTest">

    <Argument type="ExportedRoot" />

```

```

    <Argument type="Delegates">

        <Delegate>GetXomNode</Delegate>

    </Argument>

</Method>

```

InsertEmphasis

```

public delegate void InsertEmphasis(XmlNode emphasisFragmentNode,
bool overwriteSelectedContent = false);

```

Inserts an emphasized text node into the specified XML node at the cursor position. The selected content may or may not be replaced based on the value of `overwriteSelectedContent`. An optional parameter with the default value *false* has been introduced in the `InsertEmphasis` delegate. If the value of this parameter is set to *true*, it will replace the selected content on canvas.

For more information, see `InternalClass"Insert Emphasis"`.

The delegate is available for use in XML Author documents and embedded Word documents.

InsertEmphasis Parameters

Parameter	Required	Definition
<code>emphasisFragmentNode</code>	yes	<code>XmlNode</code> . The fragment node of emphasis that needs to be inserted at current selection.
<code>overwriteSelectedContent</code>	no	Boolean. If <i>true</i> , it will insert an emphasis over the selection and will overwrite the selected content. Defaults to false.

Configuration Example

```

<!-- InsertEmphasis delegate test method definition -->

<!-- This method tries to insert the emphasis XML fragment at the
current selection -->

<Method id="InsertEmphasisEI" assembly="XomManipulation"
class="XomManipulation.XomDelegates" method="InsertEmphasisTest">

    <Argument type="Tokens">

        <!--location of the XML emphasis chunk which is being
inserted-->

        <Token>c:\temp\AssignTextNodeTest.txt</Token>

    </Argument>

    <Argument type="Delegates">

        <Delegate>InsertEmphasis</Delegate>

    </Argument>

```

```
</Method>
```

Configuration Example

```
<Method id="TestInsertEmphasis" assembly="XOMCurrentTest"
class="XOMCurrentTest.XOMTest" method="TestInsertEmphasis">

  <Argument type="XomCurrentNode"/>

  <Argument type="Delegates">

    <Delegate>InsertEmphasis</Delegate>

  </Argument>

  <Argument type="Tokens">

    <Token>emphasisName=cite</Token>

    <!-- provide the name of an EmphasisDef that exists as child
to the OLEWordDocument/ParaType/Emphasis -->

  </Argument>

</Method>
```

InsertNode

```
public delegate bool InsertNode(XmlNode visibleXomNode, XmlNode
fragmentNode, string elementDisplayName, bool preserveID = false,
bool showErrorMsg = true);
```

Insert a new node immediately underneath the visibleXomNode specified. The new node and the entire subtree underneath must be schema compliant and insertable immediately under the visibleXomNode.

The third parameter, the element display name, needs to be populated only when there are multiple contexts for the node being inserted. The elementDisplayName helps to identify the element which the EI method intends to insert from a list of possible insertables. In all other cases it can be null. If this property is null and there are multiple insertables possible at the insertable point, the first one in the list will be inserted.

InsertNode Parameters

Parameter	Required	Definition
visibleXomNode	yes	XmlNode. The node under which the fragmentNode will be inserted.
fragmentNode	Yes	XmlNode. The fragment node to be inserted.
elementDisplayName	Yes	String. The friendly name of the element to be inserted.
preserveID	Yes	Boolean. Preserves the ID of the inserted fragment node.

APPLICATION CONFIGURATION: EXTENSIBILITY INTERFACE

Parameter	Required	Definition
		Defaults to false.
showErrorMsg	no	Boolean. Executes InsertNode in silent mode so that no message or warning is displayed in case insertion fails. Defaults to true.

Configuration Example

```
<!-- InsertNode delegate test method definition -->

<!-- This method tries to insert an XML fragment pointed to by
the Token argument just below the location specified by the
XomCurrentNode-->

<Method id="InsertEI" assembly="XomManipulation"
class="XomManipulation.XomDelegates" method="InsertNodeTest">

  <Argument type="XomCurrentNode"/>

  <Argument type="Tokens">

    <!-- location of the xml chunk which is being inserted -->

    <Token>c:\temp\InsertNodeTest.txt</Token>

  </Argument>

  <Argument type="Delegates">

    <Delegate>InsertNode</Delegate>

  </Argument>

</Method>
```

InvokeInternalClass

```
public delegate void InvokeInternalClass(string
internalClassName, Hashtable parameters)
```

Invokes an InternalClass from an extensibility method. This delegate is similar to the InvokeMethodId delegate except that an InternalClass method is being invoked instead of an extensibility method. Parameters required by the InternalClass must be specified in the parameters argument as a Hash table of name-value pairs.

Configuration Example

```
<!-- EI Method tests InvokeInternalClass - the EI will use the
delegate to invoke "DocumentClose"-->

<Method id="InvokeInternalClassTest"
assembly="InvokeInternalClass"
class="ExtensibilityInterface.Delegates.TestInvokeInternalClass"
method="InvokeInternalClassTest">

  <Argument type="Delegates">
```

```

        <Delegate>InvokeInternalClass</Delegate>

    </Argument>

</Method>

```

InvokeMethodId

```

public delegate object InvokeMethodId(string eiMethodName,
XmlNode node = null);

```

Invokes an Extensibility Interfaces method in the DocConfig file that matches the Extensibility Interface method name passed on specified XOM Node else if unspecified, on `XomCurrentNode`. Hence by default this passes the same `XomCurrentNode` and `ExportedCurrentNode` values that were passed to its parent method (the one calling `InvokeMethodId`) and if not, then the corresponding values of specified XOM node. This passes the return value of the invoked method (if any) back to the caller.

ToggleVisibleElement Parameters

Parameter	Required	Definition
node	no	Node. This is an optional Xom Node parameter that can be specified by calling the EI, if the <code>eiMethodName</code> needs to be invoked on a specific XOM node (other than current xom node). If unspecified, it will be invoked on <code>CurrentXomNode</code> .
eiMethodName	yes	String.

Configuration Example

```

<!-- InvokeMethodId delegate test definition -->

<!-- This method invokes another EI method called JustAnotherEI -->

<Method id="InvokeAnotherEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc" method="InvokeMethodIdTest">

    <Argument type="Tokens">

        <Token>JustAnotherEI</Token>

    </Argument>

    <Argument type="Delegates">

        <Delegate>InvokeMethodId</Delegate>

    </Argument>

</Method>

```

IsEditableElement


```
public delegate bool IsEditableElement(XmlNode node)
```

Returns **true** if the specified node is editable, otherwise it returns **false**. The node is not editable and IsEditableElement returns false if:

- inv:access is not present
- or inv:access = read-only
- or inv:access = external
- or qxa:read-only = true

Determination is based on values associated to a proprietary attribute called inv:access where xmlns:inv="urn:xpressauthor:xpressdocument". The inv:access attribute has two possible values: "read-only" and "external". The external is set (by Quark XML Author) for externally referenced elements such as referenced images or DITA conrefs. The read-only is not set by Quark XML Author but Quark XML Author will look at any read-only value set (say by an external method) to disable any UI changes to it.

Similar to inv:access, is the proprietary attribute qxa:read-only where xmlns:qxa="urn:schemas-quark-com:xmlauthor". The difference between the two is that qxa:read-only is persisted with the document when it is saved. The qxa:read-only attribute has two possible values: "true" and "false".

At the element-level, these read-only attributes are used to prevent the editing of an element.

Referenced or embedded images are still resizable, regardless of the reference implementation being used unless qxa:read-only is set to true.

Configuration Example

```
<!-- IsEditableElement delegate test method definition -->

<!-- This method returns whether the incoming node is editable or
not. In this example the EI method is returning an boolean back
to Xpress.

A boolean false returned by any EI method implies a failure which
is interpreted by Xpress to signify halting of any further
activities scheduled to

be executed under the same command(menu item/toolbar click or a
document event trigger). For ex: for the DocumentSave event we
might have a

EI method which checks for certain problems with the document. If
this EI method returns a false, the save operation is halted and
the user is notified of the failure.-->

<Method id="IsEditableEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc" method="IsEditableElement">

  <Argument type="XomCurrentNode"/>

  <Argument type="Delegates">

    <Delegate>IsEditableElement</Delegate>
```

```

    </Argument>

</Method>

```

RefreshDocVariable

```
public delegate void RefreshDocVariable(XmlNode nodeToBeUpdated);
```

Method which will update the portion mark for a given node.

Configuration Example

```

<!-- PreText delegates test method definition -->

<!-- This method assigns a randomly generated string to the
PreText area of a visible node-->

<Method id="PreTextEI" assembly="UIManipulation"
class="UIManipulation.UIDelegates" method="PreTextPostTextTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Delegates">

        <Delegate>GetNodeProperties</Delegate>

        <Delegate>GetPreText</Delegate>

        <Delegate>AssignPreText</Delegate>

        <Delegate>GetPostText</Delegate>

        <Delegate>AssignPostText</Delegate>

        <Delegate>RefreshDocVariable</Delegate>

    </Argument>

</Method>

```

RefreshDocVariables

```
public delegate void RefreshDocVariables();
```

Method which will update the portion marks on screen.

Configuration Example

```

<!-- This method looks for doc variables called "header" and
"footer" in the Word document and sets them to a specific value--
>

<Method id="AssignDocVariableEI" assembly="UIManipulation"
class="UIManipulation.UIDelegates"
method="AssignDocVariableTest">

    <Argument type="Tokens">

        <!-- the header and footer are doc variables defined in the
dot file of the document. -->

        <!-- In this simple example, the docVariables are set to a
hardcoded value. -->

```

```
<Token>header=This is the Header</Token>

<Token>footer=This is the footer</Token>

</Argument>

<Argument type="Delegates">

    <Delegate>AssignDocVariable</Delegate>

    <Delegate>RefreshDocVariables</Delegate>

</Argument>

</Method>
```

RemoveContentItem

```
public delegate void RemoveContentItem(XmlNode node)
```

This delegate applies to embedded Word documents only. Used to delete the specified XML node in the Pseudo XOM and its associated Word Content Control in an embedded Word document

Configuration Example

```
<Method id="TestRemoveContentItem" assembly="XOMCurrentTest"
class="XOMCurrentTest.XOMTest" method="TestRemoveContentItem">

    <Argument type="XomCurrentNode"/>

    <Argument type="Delegates">

        <Delegate>RemoveContentItem</Delegate>

    </Argument>

</Method>
```

AllowToggleTrackChange

```
public delegate void AllowToggleTrackChange (bool
allowToggleTrackChange)
```

This delegate sets whether the Track Changes Toggle is enabled or disabled.

Configuration Example

```
<Method id="SetReviewOptionsEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc" method="SetReviewOptions">

    <Argument type="Delegates">

        <Delegate> AllowToggleTrackChange </Delegate>

    </Argument>

</Method>
```

AllowAcceptRejectChange

```
public delegate void AllowAcceptRejectChange (XmlNode node)
```

This delegate sets whether the Accept/Reject Changes option is enabled or disabled according to configuration.

Configuration Example

```
<Method id="SetReviewOptionsEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc" method="SetReviewOptions">

  <Argument type="Delegates">

    <Delegate> AllowAcceptRejectChange </Delegate>

  </Argument>

</Method>
```

SetWordUserName

```
public delegate void SetWordUserName (string userName, string
userInitials)
```

This delegate is used to set the Username and Initials for the current Word session.

LockActiveDocument

```
public delegate LockOperationResult LockActiveDocument()
```

This delegate specifies locking the current XML Author document as the Active document of Word. The locked document will be considered as the active document for XA processing even if other documents are in focus. This delegate returns the LockOperationResult back to caller

Configuration Example

```
<Method id="LockingTest" assembly="LockUnlock"
class="ExtensibilityInterface.Delegates.LockUnlock "
method="LockingTest">

  <Argument type="Delegates">

    <Delegate>LockActiveDocument </Delegate>

  </Argument>

</Method>
```

UnlockActiveDocument

```
public delegate UnlockOperationResult UnlockActiveDocument()
```

This delegate specifies the unlocking of the locked XML Author document as the active word document and this unlocked document will no longer be considered as an active document when not in focus. This delegate returns the UnlockOperationResult back to caller .

Configuration Example

```
<Method id="Unlocking rest" assembly="LockUnlock"
class="ExtensibilityInterface.Delegates.LockUnlock "
method="UnlockingTest">
```

```
<Argument type="Delegates">
    <Delegate>UnlockActiveDocument </Delegate>
</Argument>
</Method>
```

GetComponentReferenceLink

```
public delegate XmlNode GetComponentReferenceLink (XmlNode
xmlNode, bool AllowComponentTargets = false)
```

This delegate returns the link emphasis node for a component.

ResolveReference

```
public delegate ITarget ResolveReference (XmlNode xmlNode)
```

This delegate returns the component being referred to by the link emphasis node

GetCurrentNode

```
public delegate XmlNode GetCurrentNode()
```

This delegate returns the current visible node in the document as an XML node.

Configuration Example

```
<Method id="TestEI" assembly="XOMManipulation"
class="ExtensibilityInterface.Delegates.XOM"
method="CurrentNode">
    <Argument type="Delegates">
        <Delegate>GetCurrentNode</Delegate>
    </Argument>
</Method>
```

ReplaceNode

```
public delegate bool ReplaceNode(ReplaceMap[] replacePairs)
```

Used to replace one document node with another. (Does not support replacing emphasis nodes.) ReplaceMap is a class in the Invision.XpressExtensibilityDelegates namespace containing three properties. These properties indicate the nodes to be replaced, the node to use as the replacement, and the replacement strategy to use. Each property is explained below.

```
public System.Xml.XmlNode OriginalXomNode;
```

The XomNode to be replaced.

```
public System.Xml.XmlNode ReplaceNode;
```

The new node to replace OriginalXomNode with.

```
public ReplaceStrategy StrategyToUse =
ReplaceStrategy.Dynamic|ReplaceStrategy.Redraw;
```

Specifies the replace strategy to use. **ReplaceStrategy.Dynamic** allows Quark XML Author to make the decision whether to restyle or use delete-insert.

ReplaceStrategy.Redraw specifically uses the delete and insert strategy.

This delegate was created to support changing list styles. Each list type/style has a different XAS element definition, therefore, changing the list style requires altering the document's element structure. In this scenario, there is no change in the number of visible elements. When the number of visible elements being replaced is the same as those replacing them AND the corresponding textnodes and emphasis are unchanged, the ReplaceStrategy flag allows the XOM fragment to be replaced while on the Word canvas, the narrative content is merely re-styled. This avoids the unpleasant, and unnecessary deleting and re-inserting of content on-screen.

Configuration Example

```
<!-- EI Method tests ReplaceNode - the EI will use the current
xom node to replace the previous xom node-->

<Method id="ReplaceNodeTest" assembly="ReplaceNode"
class="ExtensibilityInterface.Delegates.ReplaceNodeTest"
method="TestReplaceNode">

  <Argument type="XomCurrentNode" />

  <Argument type="Delegates">

    <Delegate>ReplaceNode</Delegate>

    <Delegate>GetPreviousNode</Delegate>

  </Argument>

</Method>
```

SelectNode

```
public delegate void SelectNode(XmlNode visibleNode)
```

Passing null value as argument to SelectNode will collapse selection to end of currently selected node.

Configuration Example

```
<!-- This EI method is called by the InvokeAnotherEI EI method
through the InvokeMethodId delegate-->

<Method id="JustAnotherEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc" method="InvokeTest">

  <Argument type="XomCurrentNode"/>

  <Argument type="Delegates">

    <Delegate>SelectNode</Delegate>

  </Argument>

</Method>
```

SelectNodes

```
public delegate void SelectNodes(XmlNode startVisibleNode,
XmlNode endVisibleNode);
```

Selects all content between and including the two endpoint nodes.

Configuration Example

```
<!-- EI Method tests SelectNode should select current XML Author
node and previous XML Author node -->

<Method id="SelectNodesTest" assembly="SelectNodes"
class="ExtensibilityInterface.Delegates.TestSelectNodes"
method="SelectNodesTest">

  <Argument type="XomCurrentNode" />

  <Argument type="Delegates">

    <Delegate>GetPreviousNode</Delegate>

    <Delegate>SelectNodes</Delegate>

  </Argument>

</Method>
```

SetSavedState

```
public delegate void SetSavedState(bool bSaved);
```

Called by an EI method to set the saved state of the document. Quark adapters use EI methods to manage document persistence. This delegate is used to ensure that the state of the document is accurate. This state is evaluated at various times such as when the user attempts to close the document. In this case, if the document state is “not saved” then the user should be prompted to save.

Configuration Example

```
<Method id="SetDocumentSaved" assembly=" assemblyName"
class="Class" method="SetDocumentSaved">

  <Argument type="Delegates">

    <Delegate>SetSavedState</Delegate>

  </Argument>

</Method>
```

SetXmlResolver

```
public delegate void SetXmlResolver(IXmlResolverProvider
provider);
```

Called by an EI method to set a custom XmlResolver instance. Typically, this EI method is called from within the Connect event in AppConfig.xml.

By default, if SetXmlResolver is not used and the document contains a URL declaration (DTD or Schema), Quark XML Author attempts to connect to the resource. If the resource is reachable, the XmlResolver property is set to

XmlUrlResolver, the native XmlResolver for Quark XML Author. If the resource cannot be reached, the XmlResolver property is set to **null**.

SetXmlResolver is used when a custom resolver is needed. The custom resolver must implement the IXmlResolverProvider interface. Information about creating and using custom resolvers can be found in Section 0.

Configuration Example

```
<!-- EI Method tests SetXmlResolver - if resolver is successfully
set message box will indicate that, and you can open DITA files
that contain doctype definitions -->
```

```
<Method id="SetXmlResolverTest" assembly="SetXmlResolver"
class="ExtensibilityInterface.Delegates.TestXmlResolver"
method="SetXmlResolverTest">
```

```
  <Argument type="Delegates">
```

```
    <Delegate>SetXmlResolver</Delegate>
```

```
  </Argument>
```

```
</Method>
```

ShadeRegion

```
public delegate void ShadeRegion(XmlNode[] nodes,
System.Drawing.Color backColor, bool recurse)
```

Shades an array of nodes with the supplied background shading. The shading will be performed for all visible nodes in the collection. This includes block elements, inline images and equations, but does not include emphasis. The recurse parameter specifies whether the procedure should recursively perform this operation on all descendants of the node collection.

Configuration Example

```
<!-- EI Method tests ShadeRegion - should select current XML
Author node and children depending on token -->
```

```
<Method id="ShadeRegionTest" assembly="ShadeRegion"
class="ExtensibilityInterface.Delegates.TestShadeRegion"
method="ShadeRegionTest">
```

```
  <Argument type="XomCurrentNode" />
```

```
  <Argument type="Tokens">
```

```
    <Token>true</Token>
```

```
  </Argument>
```

```
  <Argument type="Delegates">
```

```
    <Delegate>ShadeRegion</Delegate>
```

```
  </Argument>
```

```
</Method>
```

ToggleEditableRegion


```
public delegate void ToggleEditableRegion(XmlNode[] nodes, bool
booleanValueToBeSet, bool recurse)
```

Toggles the editable status of the specified XML node or nodes.

Configuration Example

```
<!-- ToggleEditableRegion test delegate definition -->

<!-- This particular method will take the current selection and
make it non-editable. -->

<Method id="NonEditableRegionEI" assembly="UIManipulation"
class="UIManipulation.UIDelegates"
method="ToggleRegionEditablePropertyTest">

    <!-- The XomSelectedNodes provides an array of VISIBLE nodes
selected by user -->

    <Argument type="XomSelectedNodes"/>

    <Argument type="Delegates">

        <Delegate>ToggleEditableRegion</Delegate>

    </Argument>

</Method>
```

UpdateAttributes

```
public delegate void UpdateAttributes()
```

Calls the Update Attributes event handler for an external attribute panel.

Configuration Example

```
<!-- UpdateAttributes delegate test method definition -->

<Method id="AttributePanelEI" assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc"
method="ShowElementAttributes">

    <Argument type="XomCurrentNode"/>

    <Argument type="Delegates">

        <Delegate>UpdateAttributes</Delegate>

    </Argument>

</Method>
```

AssignRefVariable

```
public delegate void AssignRefVariable(string docVariableName,
string docVariableValue, string fileName = null);
```

Assigns the named Reference Variable. Reference Variables can help resolve indirect references. *See [Variable Reference Support](#).*

AssignRefVariable Parameters

Parameter	Required	Definition
docVariableName	yes	String. Name of the document variable.
docVariableValue	yes	String. Value of the document variable. This value should be direct reference value.
fileName	no	String. The document's full name, including path. Defaults to null for the current document.

GetRefVariable

```
public delegate string GetRefVariable(string docVariableName,
string fileName = null);
```

Returns the named Reference Variable for the Quark XML Author document. [See Variable Reference Support.](#)

GetRefVariable Parameters

Parameter	Required	Definition
docVariableName	yes	String. Name of the document variable.
fileName	no	String. The document's full name, including path. Defaults to null for the current document.

GetAllRefVariables

```
public delegate int GetAllRefVariables(string[,] refVars, string
fileName = null);
```

Gets all the References Variables or their count for the Quark XML Author document. [See Variable Reference Support.](#)

GetAllRefVariables Parameters

Parameter	Required	Definition
refVars	yes	Array. An array to get all the reference variables and their values. Can be null to determine the number of variables.
fileName	no	String. The document's full name, including path. Defaults

APPLICATION CONFIGURATION: EXTENSIBILITY INTERFACE

Parameter	Required	Definition
		to null for the current document.

GetExportedXMLForRefVariables

```
public delegate string GetExportedXMLForRefVariables(string  
fileName = null);
```

Returns the serialized xml for the reference variables. The serialized xml can be persisted by a custom component in BeforeSave Event.

Reference Variables can help resolve indirect references. See Variable Reference Support.

GetExportedXMLForRefVariables Parameters

Parameter	Required	Definition
filename	No	String. The document's full name, including path. Defaults to null for the current document.

ResolveReference

```
public delegate ITarget ResolveReference(XmlNode xmlNode);
```

Resolves references for the LinkEmphasis node. Returns the resolved target.

ResolveReference Parameters

Parameter	Required	Definition
xmlNode	yes	String. Returns the XmlNode of type LinkEmphasis for example "Xref".

Configuration Example

```
<!-- ResolveReference delegate test method definition -->  
  
<Method id="ResolveReferenceTest" assembly=" ResolveReferenceTest  
" class=" ResolveReferenceTest.Example"  
  
method="TestResolveNode">  
  
<Argument type="Delegates">  
  
<Argument type="XomCurrentNode"/>  
  
<Delegate> ResolveReference </Delegate>
```

```
</Argument>
```

```
</Method>
```

EI method Example

This method will resolve link emphasis and the messageBox will display both the URL path and the element xpath if the current selected node is of type link emphasis.

```
Public void TestResolveNode(XmlNode currentXomNode, Delegates[]
delegates)

{

    ResolveReference resolveRefdelegate = delegates[0] as
ResolveReference;

    ITarget target = resolveRefdelegate(currentXomNode);

    MessageBox.Show( target.TargetLocation + " " + target.
TargetElementXPath );

}
```

GetInsertableElements

```
ArrayList GetInsertableElements(XmlNode visibleXomNode);
```

A Delegate that returns a list of Insertables (captionName) for the provided visibleXomNode.

GetInsertableElements Parameters

Parameter	Required	Definition
visibleXomNode	yes	XmlNode. Returns the visible XomNode whose Insertables are required.

Configuration Example

```
<!-- GetInsertableElements delegate test method definition -->

<Method id=" GetInsertableElementsTest" assembly="
GetInsertableElementsTest " class="
GetInsertableElementsTest.Example"

method="TestGetInsertableElements">

<Argument type="XomCurrentNode"/>

<Argument type="Delegates">

    <Delegate> GetInsertableElements</Delegate>

</Argument>
```

```
</Method>
```

EI method Example

This method will return a list of insertables and the messageBox will display all insertables allowed at the current selection location.

```
Public void TestGetInsertableElements (XmlNode currentXomNode,
Delegates[] delegates)

{

    GetInsertableElements getInsertableElementsDelegate =
delegates[0] as GetInsertableElements;

    ArrayList insertables = getInsertableElementsDelegate
(currentXomNode);

    stringBuilder sb = new StringBuilder();

    foreach(string insertable in insertables)

    {

        Sb.Append(insertables + " ");

    }

    MessageBox.Show( sb.ToString() );

}
```

GetNodeDefProperties

```
public delegate void GetNodeDefProperties(string XomName, string
parentXomName, Hashtable ht);
```

This method is used to get all the properties defined for an element in the schema. Currently the following properties are supported:

- XmlName
- Friendly
- Style
- defaultHyperlink
- displayAttribute
- insertAfter
- increaseIndent
- decreaseIndent
- externalMethodId

- default
- numbering
- visible
- readonly

For getting the properties of the EmphasisNode, the parentXomName is not required.

GetNodeDefProperties Parameters

Parameter	Required	Definition
XomName	yes	String. The XomName whose properties are required.
parentXomName	Yes	String. The possible parent Xom Name for which the node properties are required.
ht	Yes	Hashtable. A hashtable to fill the required properties.

Configuration Example

```
<!-- GetNodeDefProperties delegate test method definition -->

<Method id=" GetNodeDefPropertiesTest" assembly="
GetNodeDefPropertiesTest " class="
GetNodeDefPropertiesTest.Example"

method="TestGetNodeDefProperties">

<Argument type="XomCurrentNode"/>

<Argument type="Delegates">

  <Delegate>GetNodeDefProperties </Delegate>

</Argument>

</Method>
```

EI method Example

This method will populate hashtable with node def properties of current selected Node, the message box will show the total number of properties retrieved.

```
Public void TestGetNodeDefProperties (XmlNode currentXomNode,
Delegates[] delegates)

{

  GetNodeDefProperties getNodeDefPropertiesDelegate =
delegates[0] as GetNodeDefProperties;

  Hashtable ht = new Hashtable();
```

```

        getNodeDefPropertiesDelegate (currentXomNode.LocalName ,
currentXomNode.ParentNode.LocalName,ht);

MessageBox.Show("Total Number of properties retrieve is : "
ht.Count);

}

```

GetNodeDirtyType

```

Public delegate DirtyType GetNodeDirtyType (XmlNode xomNode,
List<string> xmlAttribList)

```

This delegate returns the dirty type of xomNode. This delegate can be used to identify which of the nodes in the document are actually dirty and the level of dirtiness. If a node is dirty due to kAttribute, then xmlAttribList will contain the list of xml attribute names due to which nodes becomes dirty.

Level of dirtiness (highest to lowest):

1. kNew, kRemove
2. kModified
3. kAttribute

```

Public enum DirtyType
{
    kNone,
    kModified,
    kNew,
    kRemoved,
    kAttribute
}

```

Configuration example:

```

<!-- GetNodeDirtyType delegate test method definition -->

<Method id="GetNodeDirtyTypeEI" assembly="XXXXX " class="XXXXX"

    method="GetNodeDirtyTypeTest">

        <Argument type="XomCurrentNode"/>

        <Argument type="Delegates">

            <Delegate> GetNodeDirtyType </Delegate>

        </Argument>

    </Method>

```

SetReferenceAttributes

```

Public delegate bool SetReferenceAttributes (XmlNode xomNode,
Hashtable referenceAttributes);

```

This delegate is used to add passed direct and indirect reference attribute value in XOMNode, where allowed. It also handles shading for the same.

Returns true if added, false otherwise.

SetReferenceAttributes Parameters

Parameter	Required	Definition
xomNode	yes	XmlNode. The XOMNode to set the reference attribute to.
referenceAttributes	yes	Hashtable. A Hashtable with reference keys and corresponding values. The only valid keys are "DirectReference" and "IndirectReference".

Configuration example:

```
<Method id="setReferenceAttributesTest"
  assembly="MiscellaneousDelegates"
  class="MiscellaneousDelegates.Misc"
  method="SetReferenceAttributesTest">
  <Argument type="XomCurrentNode"/>
  <Argument type="Delegates">
    <Delegate>SetReferenceAttributes</Delegate>
  </Argument>
</Method>
```

El method example:

```
public bool SetReferenceAttributesTest(XmlNode node, Delegate[]
delegates)
{
    try
    {
        SetReferenceAttributes
        setReferenceAttributesDelegate = delegates[0] as
        SetReferenceAttributes;

        if (setReferenceAttributesDelegate != null &&
node != null && node is XmlElement)
        {
            // Using SetReferenceAttributes Delegate to
            set the conkeyref and conref attr
        }
    }
}
```



```
        Hashtable ht = new Hashtable();

        ht.Add("DirectReference", "c:/Dummy.xml");

        ht.Add("IndirectReference", "c:/Dummy1.xml");

        return setReferenceAttributesDelegate(node,
ht);

    }

}

catch (Exception e)

{

    System.Diagnostics.Debug.WriteLine("SetReferenceAttributes
Exception :" + e.Message + "\nStackTrace:" + e.StackTrace);

}

return false;

}
```

RemoveReferenceAttributes

```
Public delegate bool RemoveReferenceAttributes (XmlNode xomNode)
```

This delegate is used to remove all the direct and indirect reference attribute values from the passed XOMNode. It also handles shading for the same.

Returns true if removed, false otherwise.

RemoveReferenceAttributes Parameters

Parameter	Required	Definition
xomNode	yes	XmlNode. The XOMNode to set the reference attribute to.

Configuration example:

```
<Method id="RemoveReferenceAttributesTest"
assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc"
method="RemoveReferenceAttributesTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Delegates">

<Delegate>RemoveReferenceAttributes</Delegate>

    </Argument>

</Method>
```

El method example:

```

        public bool RemoveReferenceAttributesTest(XmlNode node,
        Delegate[] delegates)
        {
            try
            {
                RemoveReferenceAttributes
                removeReferenceAttributesDelegate = delegates[0] as
                RemoveReferenceAttributes;

                if (removeReferenceAttributesDelegate != null &&
                node != null && node is XmlElement)
                {
                    return
                    removeReferenceAttributesDelegate(node);
                }
            }
            catch (Exception e)
            {
                System.Diagnostics.Debug.WriteLine(RemoveReferenceAttributesExcep
                tion : "+ e.Message + "\nStackTrace:" + e.StackTrace);
            }
            return false;
        }
    
```

SetListRestartContinueNumbering

```

Public delegate void SetListstartRestartContinueNumbering (XmlNode
xomNode, bool restart)
    
```

This delegate is used to restart/continue the list item.returns the dirty type of xomNode.

SetListRestartContinueNumbering Parameters

Parameter	Required	Definition
xomNode	yes	XmlNode. The XOMNode to set the reference attribute to.
restart	yes	If set to true , restart the list item, else continue.

Configuration example:

```

<Method id="SetListRestartNumberingTest"
assembly="MiscellaneousDelegates"
    
```

```

class="MiscellaneousDelegates.Misc"
method="SetListRestartContinueNumberingTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Tokens">

        <Token>True</Token>

    </Argument>

    <Argument type="Delegates">

        <Delegate>SetListRestartContinueNumbering</Delegate>

    </Argument>

</Method>

<Method id="SetListContinueNumberingTest"
assembly="MiscellaneousDelegates"
class="MiscellaneousDelegates.Misc"
method="SetListRestartContinueNumberingTest">

    <Argument type="XomCurrentNode"/>

    <Argument type="Tokens">

        <Token>False</Token>

    </Argument>

    <Argument type="Delegates">

        <Delegate>SetListRestartContinueNumbering</Delegate>

    </Argument>

</Method>

```

EI method example:

```

public void SetListRestartContinueNumberingTest(XmlNode
curXomNode, string []restart, Delegate[] delegates)
{
    try
    {
        SetListRestartContinueNumbering
setListRestartContinueNumbering = delegates[0] as
SetListRestartContinueNumbering;

        if (setListRestartContinueNumbering != null &&
curXomNode != null && curXomNode is XmlElement)

```

```

    {
        // Set restart numbering for New Numbered
        list inserted and not new List item added

        XmlNode listItemNode =
        curXomNode.SelectSingleNode("self::*[starts-with(local-
        name(),'oli')]");

        if (listItemNode != null)
        {

            setListRestartContinueNumbering(listItemNode, restart[0] ==
            "True");

        }
    }
}

catch (Exception e)
{

    System.Diagnostics.Debug.WriteLine("SetListRestartContinueNumberi
    ng Exception :" + e.Message + "\nStackTrace:" + e.StackTrace);

}
}

```

RemoveRefVariable

```
Public delegate void RemoveRefVariable (string docVariableName,
string filename = null)
```

This delegate is used to remove the named reference variable from the Quark XML Author document. See Variable Reference Support.

RemoveRefVariable Parameters

Parameter	Required	Definition
docVariableName	yes	String. Name of the document variable to be removed.
filename	yes	The document;s full name, including the path. Defaults to null for the current document.

List of Available Document Events

The available document events are defined below.

For information on using events, see section 6.2.2.1, “Calling the EI Method from a Document Event”.

Activated

The Activated event fires when a document is created, opened or when the document window receives focus.

AfterSave

The AfterSave event fires after a document is saved.

BeforeComponentDelete

The BeforeComponentDelete is fired by a user action or the DeleteNode EI delegate when the action will result in one or more element being deleted. Specifically, the event is fired before all specified elements have been deleted.

BeforeComponentDelete supports the DeleteNodes argument type. A return value of false will cancel the delete operation.

BeforeOpen

The BeforeOpen event fires just prior to the opening of a document. Typically used in conjunction with conversions, or any other preprocessing that cannot be accomplished with a transform. For example, can be used to in conjunction with the conversion of XML entities (HTML and other) to Unicode characters.

Definition:

BeforeOpen is a child of Namespace.

Any ExtensibilityMethod registering for this event has access to all content in the document, two document arguments and one token argument type. The three document arguments are ExportedRoot which is the document content that can be manipulated and returned, the Filename argument which is the full name of the file being opened and the standard Tokens argument.

The following AppConfig fragment sample shows this event being used to invoke an EI method.

```
<BeforeOpen assembly="EIHarness" method="ExecuteBeforeOpenStuff"
class="EIHarness.EITester" >

  <Argument type="ExportedRoot"></Argument>

  <Argument type="Filename"></Argument>"

  <Argument type="Tokens">

    <Token>xform1= dita/transform1.xsl</Token>

    <Token>xform2= dita/transform2.xsl </Token>

  </Argument>
</BeforeOpen>
```

Responsibilities:

If the BeforeOpen EI throws an exception or returns *false* the Open process is cancelled.

Dependencies / Constraints:

A Namespace node may contain both the BeforeOpen node and the transform attribute. See section 8.7.1, Namespace. The Namespace reevaluate attribute is associated with the BeforeOpen event and the transform attribute.

BeforePaste

Whenever the user attempts to insert an XML component, the BeforePaste event fires, the component is attached to the XOM, the content is inserted into the document, and then the InsertComponent event fires. An EI method associated with this event can receive the argument type **PasteNode**. This is the XML node fragment about to be inserted. A method associated with BeforePaste can modify the PasteNode XML—it can modify attributes, modify element structure, and so forth, to conform to schema or business requirements.

BuildStyleList

Whenever the selection within a document changes, the BuildStyleList event fires in order to rebuild the list of elements that may be added to the document in the new cursor location.

ComponentDelete

The ComponentDelete event fires after one or more elements have been deleted as a result of a user action or is raised by the DeleteNode EI delegate. Specifically, the event is fired after all specified elements have been deleted.

Similar to BeforeComponentDelete, ComponentDelete supports the DeleteNodes argument type. However, in the case of ComponentDelete, the nodes have been detached from the tree. To provide the EI developer context, we also supply references to the deleted component's former parent node. The DeletedNodes argument is now a 2-dimensional array that associates each deleted node with its parent. The deleted node and its parent are essentially stored as a pair.

(0,0) accesses deleted node 1(0,1) accesses deleted node's parent 1

(1,0) accesses deleted node 2(1,1) accesses deleted node's parent 2

Return values from any of the invoked EI methods will not affect the delete operation.

The ComponentDelete event is not fired for the following:

- deletion of emphasis nodes
- DeleteRow(), DeleteColumn() and cell merges

Connect

The Connect event, used only in AppConfig.xml, fires immediately after Quark XML Author is loaded. It is used to connect Quark XML Author to other applications. Connect is the only event that is "global".

To use Connect, define an EI Method for it in the standard manner. Connect can only take delegates or tokens as arguments.

In the ExtensibilityInterface node, you must also define a <Connect> child node that calls the EI Method. In the example below, the EI Method has been defined with the **id** value **DSConnection**.

```
<Connect>  
  
<ExtensibilityMethod id="DSConnection"/>  
  
</Connect>
```

DocClosed

The DocClosed event fires just prior to the closing of a Quark XML Author document. A separate DocClosed event is raised for each Quark XML Author document. All EI methods associated with this event are invoked sequentially. If an associated EI method returns false, the Close operation is cancelled. If the EI method raises an exception, a confirmation dialog prompts the user whether to continue or cancel the closing of the document.

DocOpened

The DocOpened event fires when a document is opened. Called before the Initialize event.

DoubleClickHeaderFooter

The DoubleClickHeaderFooter event fires when the user double clicks the header or footer region of the document if AllowHeaderFooterEdit (Section 9.8) is set to **true**.

Initialize

The Initialize event fires after DocOpened when a document is opened, but only for documents with no Binary data.

InsertElementXom

The InsertElementXom event fires for each element that is added to the XOM, regardless of whether that element is visible in the user interface. Not called for components attached via import, paste or InsertNode delegate.

InsertElementUI

The InsertElementUI event is similar to InsertElementXom, but fires only when adding elements that are visible in the user interface. For example, if a Note element was added, and it was itself invisible to the UI, but contained visible Title and Paragraph child elements, InsertElementXom would fire three times, but InsertElementUI would fire only twice.

InsertComponent

The InsertComponent event fires for drag and drop import, component paste, user-inserted elements, and the InsertNode delegate. The event is raised once for every top-level element that is inserted. For example, if the user inserts an element named Section that contains required Title and Para child elements, the firing order of events is as follows:

Section - InsertElementXom, InsertElementUI (if visible)

Title - InsertElementXom, InsertElementUI (if visible)

Para - InsertElementXom, InsertElementUI (if visible)

Section - InsertComponent

An EI method associated with this event can receive the argument type **PasteNode**. This is the XML node fragment about to be inserted.

InsertComponent will fire when splitting an element if the ElementDef in the XAS file uses the **insertAfter** attribute.

Drag and Drop

There is not an OnDrop event. Use the **linkDropHandler** attribute to specify the ID of the EI method to be called when the user drops a hyperlink onto the element. See “Table 14-1: ElementDef Attributes”.

This feature overrides any internal XML Author behavior. For example, in the sample below, if “tocnumtitle” supports inline media elements, and the hyperlink dropped on this element happens to be a hyperlink to a media that the user wants to insert, the inline media behavior will never be invoked. Instead the EI method specified in the **linkDropHandler** attribute is invoked.

Example configuration

```
<ElementDef name="tocnumtitle" xmlns="title" friendly="Numbered
Title" style="Heading 8" linkDropHandler="TestLinkDropHandler">

  <Section>

    <Para>

      <Emphasis>

        &emphasis-ref;

      </Emphasis>

    </Para>

  </Section>

  ...

</ElementDef>
```

Example method

```
<Method id="TestLinkDropHandler"
assembly="TestEIs\DragDropTester" class="DragDropTester.Test"
method="TestLinkHandlerEI">

  <Argument type="Tag"/>

</Method>
```


Tag: A special argument that contains the hyperlink that was dropped.

RightClickPreText

The RightClickPreText event fires when the user clicks on the portionmark pre-text area of a visible node. Used only in the DocConfig file.

RightClickPostText

The RightClickPostText event fires when the user clicks on the portionmark post-text area of a visible node. Used only in the DocConfig file.

Save

The Save event is fired just before the document is saved locally. Any EI methods specified in the Save node are executed and if any of them throws an exception or returns false, the save is cancelled.

UpdateAttributes

The UpdateAttributes event fires when an attribute is updated via the internal attribute panel. Any external methods defined under the <UpdateAttributes> tag in the EI are executed. If an external attribute panel is used to change attributes, there is a corresponding UpdateAttributes delegate which fires the UpdateAttributes event separately

StyleListParents

This is a special argument type used exclusively by the BuildStyleList EI methods. It contains a list of parents of currently insertable elements provided by the styleList argument with a one – on – one linkage with stylelist items. This argument can help the EI developer to distinguish between insertables under different parents with same name.

Application Configuration: ResolveReferences

Quark XML Author provides two methods for inserting referenced content into a document: a drag-and-drop operation and pasting a reference from the clipboard—if a user copies a component reference to the clipboard, Assign Reference becomes available on the Quark XML Author component context menu. In both cases, Quark XML Author must be able to resolve the reference hyperlink. The AppConfig.xml file's ResolveReferences node provides the means to accomplish this task.

The Reference Node

ResolveReferences contains a collection of Reference nodes. Each Reference node identifies the Content Management System via its pattern attribute, and calls the external DLL that performs the download operation. ResolveReferences and the Reference node take the following format:

```
<ResolveReferences>
  <Reference pattern="pattern" assembly="assembly name"
    class="class name">
    <Token>Token 1</Token>
    <Token>Token 2</Token>
    <Token>Token #</Token>
  </Reference>
  <!-- Additional Reference nodes as required -->
</ResolveReferences>
```

Pattern Attribute

The pattern attribute's value must be a regular expression that contains a matching pattern for the URI that refers to the CMS. When an object is dragged into Quark XML Author, Quark XML Author searches for a pattern that can be matched against the URI for the object being dragged into it. For example:

```
pattern="http://yellowstone/*"
```

This would match everything following http://yellowstone in a URI.

To allow all URIs to match a single Reference node, the wildcard character phrase (.*) can be used as the value of the pattern attribute. The period must be present; an asterisk alone will match only one wild card character.

Assembly and Class Attributes

The assembly and class attribute values specify the assembly and class in which the download method for the CMS can be found. The download method that must be implemented in the DLL is discussed in section 17, “Integration with Content Management Systems”.

Specify the assembly within the class as the value of the assembly attribute. Specify the fully qualified class name, without the .dll extension, as the value of the class attribute.

Tokens

Parameter values that are required (if any) by the download method in the external DLL are specified as Tokens within the Reference node. Each parameter value is supplied as the content of a separate Token element.

Application Configuration: Miscellaneous items

There are additional elements that must be set to configure the application. The first set of elements affect the application environment. The last specifies namespaces for your organization's document classes, and is considered separately.

CultureName

Defines the language and language culture. Defaults to **en-us**, as shown in the example.

```
<CultureName>en-us</CultureName>
```

Language and Numerical Content

Quark XML Author saves numerical content, including date and time values, in English format. When opening documents, Quark XML Author expects numerical content to be in English format.

EnableProgressDialogs

Used to enable the display of the various progress bar dialogs in Quark XML Author. For example, opening a document containing a table or splitting table cells.

Defaults to **true**.

```
<EnableProgressDialogs>false</EnableProgressDialogs>
```

ErrorLogging

Determines if an error log should be defined when Quark XML Author is run. The name of the log file is **XML Author Log.txt** and can be found in Quark XML Author folder with the user's Application Data folder.

```
<ErrorLogging>true</ErrorLogging>
```

ExtendedLoggingInfo

If enabled, ExtendedLoggingInfo will log additional details about exceptions being logged in XML Author Log.txt. This flag tries to collect specific information in case

of any error or data loss happening in XML Author and logs that information in the log file. This flag should be set to true on the machine where data loss or corruption, if any, is occurring and the logs should be sent to Quark XML Author Customer support team for analysis.

```
<ExtendedLoggingInfo>false</ExtendedLoggingInfo>
```

LogFilePath

```
<LogFilePath>{path}</LogFilePath>
```

The optional LogFilePath element can be used to specify a file path (drive or UNC) for the file created in conjunction with ErrorLogging. It supports variable placeholders for special folders such as APPDATA and MYDOCUMENTS. If this element is omitted, Quark XML Author continues to use the default location.

Defaults to:

{the path of the current system's temporary folder}\Quark\XML Author.

UndoHistory

Defines the number of sequential actions that may be reversed when using the Undo command. In the example, up to 50 actions may be reversed.

```
<UndoHistory>50</UndoHistory>
```

There is no maximum, but with some features such as table features the number of undo operations can go up exponentially so best practice is to set the value to 500.

Namespaces

The Namespaces node contains two types of child elements: Namespace and NotSupported.

Namespace

Each supported document class will have its own Namespace element within the Namespaces node.

Table 8-1: Namespace Attributes

Attribute	Required	Definition
config	yes	Specifies the location and filename of the document configuration file for the document class. The location must be specified as a relative path.
internalName	no	For multiple definitions with the same xmlname property at different levels in the XAS,

Attribute	Required	Definition
		identifies a single element definition that maps to the root node of the incoming document.
mapperAssembly	no	The attribute, along with mapperClass, points to an external method that allows for an external condition to be injected in the evaluation of a particular namespace node.
mapperClass	no	This attribute, along with mapperAssembly, points to an external class that implements the Invision.Xpress.Interfaces.IConditionMapper containing the condition to be evaluated.
productLine	no	Specifies a name for a document class which can be matched by the productLine processing instruction. May be defined in conjunction with uri and root.
reevaluate	no	If set to true , specifies that namespace selection logic is run again after the transform and any associated EI (i.e. BeforeOpen) returns. Reevaluation (and optional transform / BeforeOpen) continues until a namespace is selected without a transform / BeforeOpen or reevaluate equals false . Used in conjunction with transform attribute and BeforeOpen document event.
root	no	Specifies the root element for a document class which may be matched by the document root element. Maybe defined in conjunction with productLine and uri.
schema	yes	Specifies the location and filename of the Quark XML Author Structure (.xas file) for the document class. The

Attribute	Required	Definition
		location must be specified as a relative path.
template	yes	Specifies the location and filename of the Word template (.dot file) for the document class. The location must be specified as a relative path.
transform	no	Specifies the location and filename of the default stylesheet for the document class. The location must be specified as a relative path. A Namespace node may contain both the BeforeOpen node and the transform attribute. The Open transform and any ExtensibilityMethod has access to all content in the document.
uri	no	Specifies a URI for the document class which can be matched by the uri attribute of the document root element. May be defined in conjunction with productLine and root. May use the * wildcard character. For example, uri="urn:invision:doc*" would match both "urn:invision:doc" and "urn:invision:doc:trm".

NotSupported

The NotSupported element allows a relevant “not supported” user message when the user attempts to open an XML document that does not match one of the supported document classes.

Table 8-2: NotSupported Attributes

Attribute	Required	Definition
message	yes	Specifies the message to be displayed when the user attempts to open an unsupported
productLine	no	Specifies a name for a document class which can be matched by the productLine processing instruction. May be

Attribute	Required	Definition
		defined in conjunction with uri and root.
root	no	Specifies the root element for a document class which may be matched by the document root element. Maybe defined in conjunction with productLine and uri.
uri	no	Specifies a URI for the document class which can be matched by the uri attribute of the document root element. May be defined in conjunction with productLine and root. May use the * wildcard character. For example, uri="urn:invision:doc*" would match both "urn:invision:doc" and "urn:invision:doc:trm".

TempFilePath

The optional TempFilePath element can be used to specify a file path (drive or UNC) for temp file storage. It supports variable placeholders for special folders such as APPDATA and MYDOCUMENTS. If this element is omitted, Quark XML Author continues to use the default location.

Blocking Shortcut Keys

Microsoft Word provides a variety of built-in shortcut keys to invoke dialogs, change text formatting, and perform other actions that can “break” a Quark XML Author document. In addition, some features in Word are not supported and should be blocked. The ShortcutKey section of the DocConfig can be used to block these built-in shortcut keys.



Quark recommends that you do not block shortcut keys in the AppConfig file. If you do, the shortcut keys will be unavailable even when the user is working with a standard Word document.

To block a shortcut key, simply create a ShortcutKey node for it in the Shortcuts node of the DocConfig file. For example, to disable the Ctrl + R shortcut (which right-justifies the selected paragraph), you would add the following node to the ShortcutKeys element:


```
<ShortcutKey key="R" shift="false" ctrl="true"/>
```

When the keystroke combination is pressed, Quark XML Author intercepts it. Because the node contains no internal class or extensibility interface call, Quark XML Author then discards the keystroke combination and nothing happens.

Note that if you define a keystroke combination, as described in section 3, that is identical to an existing combination, the redefined version will be used by Quark XML Author.

The example below provides some commonly-disabled shortcut keys. Note that this list is not intended to be comprehensive or definitive; your organization may need to disable other keystroke combinations, or may wish to use some of them.

```
<ShortcutKey key="C" shift="true" ctrl="true"/>
<ShortcutKey key="D" shift="false" ctrl="true"/>
<ShortcutKey key="E" shift="false" ctrl="true"/>
<ShortcutKey key="J" shift="false" ctrl="true"/>
<ShortcutKey key="K" shift="false" ctrl="true"/>
<ShortcutKey key="L" shift="false" ctrl="true"/>
<ShortcutKey key="M" shift="false" ctrl="true"/>
<ShortcutKey key="R" shift="false" ctrl="true"/>
```

DisableTemplateAddins

Used to disable all third-party template add-ins. See “Hiding/Disabling Ribbon Tabs for Third-Party Software”. This feature is specified in the AppConfig.

The Quark XML Author add-in cannot influence the position or visibility of the ribbon tabs of a third-party add-in. Therefore, disabling the add-in is the only way to remove its ribbon tab when Quark XML Author is in focus/active.

Two types of add-ins are used: Template and COM. For example, Adobe Acrobat is a COM add-in. XML Author can disable template add-ins and some COM add-ins. COM add-ins installed as available “to all users” cannot be disabled by Quark XML Author. When the user views a standard Word document, COM add-ins that have a user interface components will be re-displayed. In addition, add-ins that are needed by Quark XML Author may not be disabled.

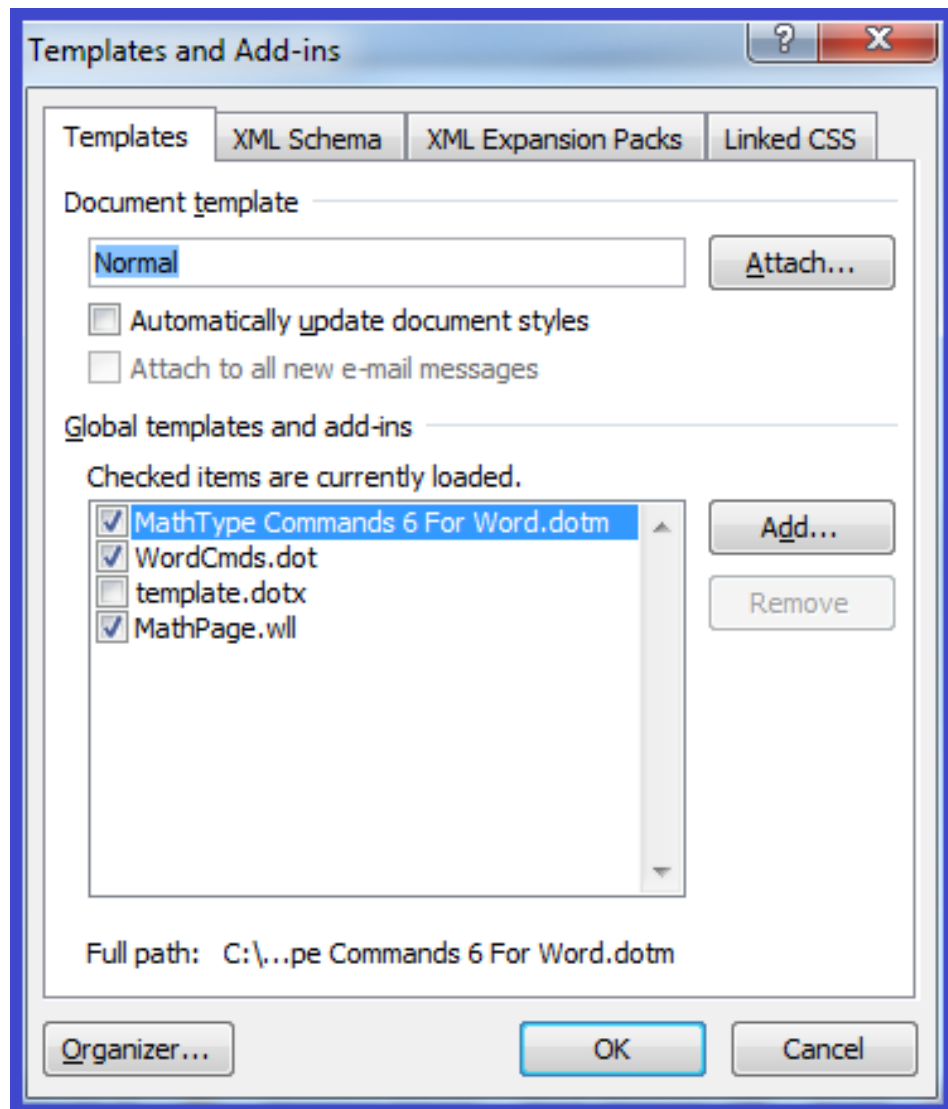


Figure 8-1: Third-party Template Add-ins

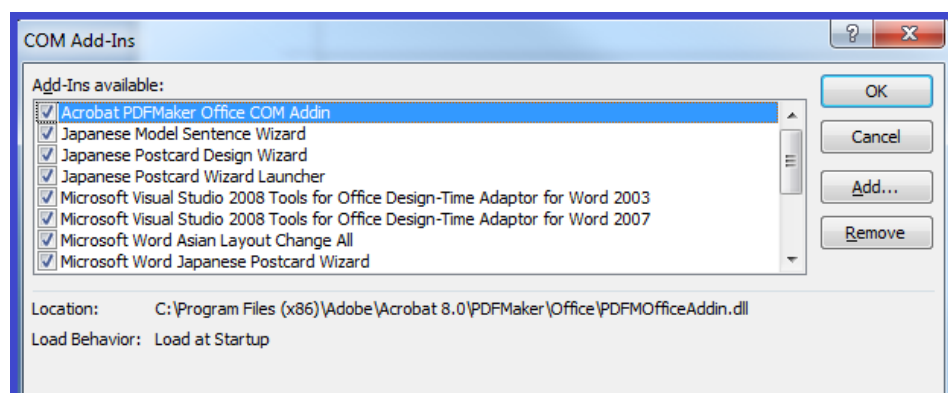


Figure 8-2: Third-party COM Add-ins

This feature can be used in the following ways:

- Disable all add-ins

- Disable all add-ins except the specified list
- Disable the specified list of add-ins

Table 8-3: DisableTemplateAddins Attributes

Attribute	Required	Definition
AddIn	no	Specifies the name of the add-in that should be disabled. This is a case-sensitive wildcard. If the wildcard is contained in any part of an add-in's name then this is considered to be a match and the system attempts to disable that add-in.
All	no	Specifies that the system should attempt to disable all add-ins. The child node <Except> is used to specify add-ins that should be omitted from the disable attempt.

Disable all add-ins

Disables all add-ins possible.

Example

```
<DisableTemplateAddIns>
  <All/>
</DisableTemplateAddIns>
```

Disable all add-ins except the specified list

Disables all add-ins possible except those whose name contains "MathType".

Example

```
<DisableTemplateAddins>
  <All>
    <Except>MathType</Except>
  </All>
</DisableTemplateAddins>
```

Disable the specified list of add-ins

Disables all add-ins whose name contains "MathType" or "Our Custom Addin".

Example

```
<DisableTemplateAddins>
  <AddIn>MathType</AddIn>
  <AddIn>Our Custom Addin</AddIn>
</DisableTemplateAddins>
```

ImageFilePath

```
<ImageFilePath>{path}</ImageFilePath>
```

The optional ImageFilePath element can be used to specify a file path (drive or UNC) that specifies the “export” location where images in embedded Word documents will be stored. Hrefs in serialized Smart Content will include these paths. See “OLE Word Document Integration”. It supports variable placeholders for special folders such as APPDATA and MYDOCUMENTS. If this element is omitted, Quark XML Author continues to use the default location.

Defaults to:

{the path of the current system’s temporary folder}\Quark\XML Author.

Document Level Configuration

DocConfig files can have any name, but will typically be named to indicate the document schema it supports. For example, a schema named chapter.xas might have a DocConfig file named chapter-config.xml.

In addition to the same Menu, CommandBar, and Shortcut configuration options that the AppConfig file has, the DocConfig file includes the elements defined in the rest of this section.

Portionmarks

Every visible element in a Quark XML Author document has a pre-text and post-text area that can be accessed and changed by four extensibility delegates. See section 6.4 for definitions of these delegates, which are AssignPostText, AssignPreText, GetPostText, and GetPreText. In addition, two document events (RightClickPreText and RightClickPostText, discussed in section 6.5) provide a means to launch Extensibility Interface methods whenever they are invoked. Which Portionmarks are available, their values, and the elements to which they belong are specified in the DLL that these delegates invoke.

Portionmarks can be specifically configured at the document, cell, element and cell element level for any visible element, empowering administrators to control the impact that enabling Portionmarks may have on the performance of the document.

- Configuring Portionmarks at the table level:

The configurability of Portionmarks is controlled hierarchically. For example, if Portionmarks are enabled at the **Document** level, but disabled at the **Cell** level, then subsequent element level and cell element configurations become meaningless. The configurability follows the following hierarchical order:

Document Portionmarks > Cell Portionmarks > Element Portionmarks > Cell Element Portionmarks

- Configuring Portionmarks for elements outside a table:

Portionmarks for elements outside a table are governed by the Element Portionmarks configuration. The configurability follows the following hierarchical order:

Document Portionmarks > Element Portionmarks

An example is shown below.

```
<ElementDef name="table-sectionx1" xmlns="section"
friendly="Single Section" visible="true" style="SectionMarker"
allowPortionmark="true">
```

```

<Section>
  <Sequence minOccurs="1" maxOccurs="1">
    <SectionType name="p-start" readonly="true"
default="Section Start"/>
    .....
    ..... </Section>
  <Attributes>
    .....
  </Attributes>
</ElementDef>

```

Making Portionmarks available requires that the Portionmarks node in the document configuration file be populated with Portionmark child elements, which are defined below.

1. Portionmark Attributes

Table 9-1: Portionmark Attributes

Attribute	Required	Definition
defaultDisplay	no	If set to true , Word will display the portionmark when the document is initially opened.
styleName	no	The name of the Word style in the Word template that will be used to render the portionmark data on the screen. The Portion mark style has to be a character based style and not a para based style.
type	yes	Must have a value of either PreText or PostText . Only one node of each is allowed.
addCellPortionmark	no	If set to false , the table cells portionmarks will be disabled. If set to true , the table cells portionsmarks will be enabled. If it is not defined, then the default value is false .

An example is shown below.

```

<Portionmarks addCellPortionmark="false"/>
<Portionmark type="PreText" defaultDisplay="true"
styleName="Portionmark"/>
<Portionmark type="PostText" defaultDisplay="true"
styleName="Portionmark"/>
</Portionmarks>

```

NOTE: Setting the value of the **addCellPortionmark** parameter to **true** might degrade the performance of some table operations and the document open/save operation.

ReferenceShadingColor

Defines the color that will be used as shading for elements that are referenced from external sources. The color is defined by the values of the **red**, **green**, and **blue** attributes, with each value being an integer between 0 and 255.

An optional child element, FailedShadingColor, can be used to specify the color to be used for a failed or broken reference. The color is defined by the values of the **red**, **green**, and **blue** attributes, with each value being an integer between 0 and 255.

AllowRestartContinueNumbering

If set to **Yes**, allows the user to restart numbering when a second numbered list is inserted within a document. If set to **No**, all numbered lists within the document will be numbered sequentially, with no restarting allowed.

DefaultSaveOptions

Attributes of this element specify options for all save operations:

Table 9-2: DefaultSaveOptions Attributes

Attribute	Definition
allowLocalSave	Boolean. Defaults to true . If set to false , the user will not be prompted to save changes when the document is closed, even if there are unsaved changes. The user will instead be prompted, "Do you wish to close XML Author and lose unsaved changes?"
defaultFilepath	When specified, this path is used as the Save In folder in the Save As dialog. The dialog uses this path as the current folder of the dialog. The user is not restricted to this path.
filter	Specifies the default filter options for the Save dialog box.
nativeFormat	Boolean. Set to true if document is to be saved in Word format, false if it is to be saved only as XML. Defaults to true .
transform	XSL code for transforming the document being saved.

ComponentCopy

If present, this optional node either invokes a transform method specified by its transform attribute, or invokes an Extensibility Interface method, via its getReferenceId attribute, that returns the conref or cross reference target string.

Table 9-3: ComponentCopy Attributes

Attribute	Required	Definition
getReferenceId	no	The ID of the Extensibility Interface method that returns the conref or cross reference target string.
transform	no	Path to the stylesheet used to transform a copied component.

Commenting

The Commenting node allows you to specify additional custom attributes, and enumerated values for those attributes, for comments that the user adds to the document in the authoring environment. The author can access these attributes by right-clicking in the Word Comment balloon.

The Commenting node contains one or more Attribute nodes.

Attribute

Each Attribute child element of Commenting describes one attribute that can be applied to document comments.

Table 9-4: Commenting > Attribute Element Attributes

Attribute	Definition
author-att	The name of the attribute that will contain the author's ID value for each comment.
dateTime-att	The name of the attribute that will contain date/time stamp value for each comment.
value-att	The name of the attribute that will contain the value supplied by the user for each comment.

QuarkSection-Level4

Each Attribute child element of Commenting contains one or more Enumeration nodes that describe the possible values that the author can assign.

Table 9-5: Commenting > Attribute > Enumeration Element Attributes

Attribute	Definition
autotext	Any autotext that is to be appended to the comment when the value is selected. You may use the keywords {author} and {timestamp} to insert the author ID and a timestamp along with the selected value.
menuitem	The value as it will appear to the user in the context menu.

Attribute	Definition
value	The value that will be assigned to the XML attribute.

Break

This optional node supports section, page, and column breaks, in conjunction with the sectionBreak, pageBreak, and columnBreak datatypes. See Table 11-3, in section 11.6.2, for more information on these datatypes and how they are used.

The **Break** node has three optional child elements: **SectionBreak**, **PageBreak**, and **ColumnBreak**. These elements can have one of two values: **restricted** and **unrestricted**. These elements default to **unrestricted**, and if the Break node is absent from the DocConfig file, the defaults are applied. In the example below, these types of breaks are restricted:

```
<Break>
  <SectionBreak>restricted</SectionBreak>
  <PageBreak>restricted</PageBreak>
  <ColumnBreak>restricted</ColumnBreak>
</Break>
```

SectionBreak

When SectionBreak is set to **restricted**, the Insert Section Break command can only be invoked when the cursor position in the document is on an Empty (Insertion Point) and the next visible element has a sectionBreak attributeType defined (see section 12.3).

When SectionBreak is set (or defaults) to **unrestricted**, the InsertSectionBreak command can be invoked when the cursor position in the document is on an Empty (Insertion Point) preceding any visible document element. Every visible element in the schema will inherit this capability; there is no need to explicitly define a sectionBreak attributeType for each visible element.

PageBreak

When PageBreak is set to **restricted**, the InsertPageBreak command can only be invoked when the cursor position in the document is on an Empty (Insertion Point) and the next visible element has a pageBreak attributeType defined (see section 12.3).

When PageBreak is set (or defaults) to **unrestricted**, the InsertPageBreak command can be invoked when the cursor position in the document is on an Empty (Insertion Point) preceding any visible document element. Every visible element in the schema will inherit this capability; there is no need to explicitly define a pageBreak attributeType for each visible element.

ColumnBreak

When ColumnBreak is set to **restricted**, the InsertColumnBreak command can only be invoked when the cursor position in the document is on an Empty (Insertion

Point) and the next visible element has a columnBreak attributeType defined (see section 12.3).

When ColumnBreak is set (or defaults) to **unrestricted**, the InsertColumnBreak command can be invoked when the cursor position in the document is on an Empty (Insertion Point) preceding any visible document element. Every visible element in the schema will inherit this capability; there is no need to explicitly define a columnBreak attributeType for each visible element.

AllowHeaderFooterEdit

This optional node can contain either **true** or **false**, and defaults to **false**. An example is shown below:

```
<AllowHeaderFooterEdit>true</AllowHeaderFooterEdit>
```

If set to **true**, document authors can double-click the header/footer regions and modify content in those regions. Quark XML Author will not impose any restrictions on the content in these regions; the WordML content is stored in the document using the CDATA WordSection node.

If set to **false** (or not specified), Quark XML Author will detect a user's attempt to edit the header and footer document region. Quark XML Author will intercept a double-click event and invoke any EI methods associated with EI event, "DoubleClickHeaderFooter", if any. If this event contains no EI method, Quark XML Author will consume the keystroke.

AllowSoftLinebreaks

This optional node can contain either **true** or **false**. Defaults to **false**. An example is shown below:

```
<AllowSoftLinebreaks>true</AllowSoftLinebreaks>
```

If false, document authors cannot insert soft line breaks (using Shift + Enter) into any element in the document. If set to **true**, document authors can insert soft line breaks in any text element.

Smart Paste

Also when soft line breaks are allowed, clipboard content containing a line break will introduce a PasteSpecial context menu item within a paragraph element. This menu option will allow users to preserve line breaks while pasting text in the document.

Reviewing

The <Reviewing> node is a leaf node with one attribute, modelType, with two possible values: **OneNode** or **TwoNode**. Default value is TwoNode.

When set to **OneNode**, the two node model of representing Revision and Comment nodes is converted in the serialized document to a single node model.

HiddenInsertables

This optional node specifies which empty should be used to calculate insertables when there are hidden elements (made hidden by the use of ToggleVisibleElement Internal class) between the visible nodes surrounding the currently selected empty.

Table 9-6: HiddenInsertables Attributes

Attribute	Required	Definition
selectionStrategy	yes	<p>Possible values: FirstHiddenElement, LastHiddenElement and AllHiddenElements.</p> <p>FirstHiddenElement considers the empty between current top visible node and the first invisible node.</p> <p>LastHiddenElement considers the empty between the last invisible empty and current bottom visible node.</p> <p>AllHiddenElements considers all empties between every invisible element pairs and combines them into one insertable list. Be careful using the AllHiddenElements setting. If a large number of items are hidden, then the list of items added to the insertables list may result in a large list whose usability is low.</p>

Change To Menu

The <ShowChangeToMenu> node allows you to specify whether or not “change to” menus are displayed for any elements.

Optional. Boolean. Defaults to **true**.

For information on controlling the display for a specific element type, see excludeFromChangeToMenu in Table 14-1.

ShowInsertBeforeMenu

The <ShowInsertBeforeMenu> node allows you to specify whether or not “Insert Before” menus are displayed in the Context Menu for the current element.

Optional. Boolean. Defaults to **true**.

Insert Before menus contain the list of insertable elements that the user is allowed to insert before a given element. The list is the same as if the user right-clicked on the Insertion Point above the given element.

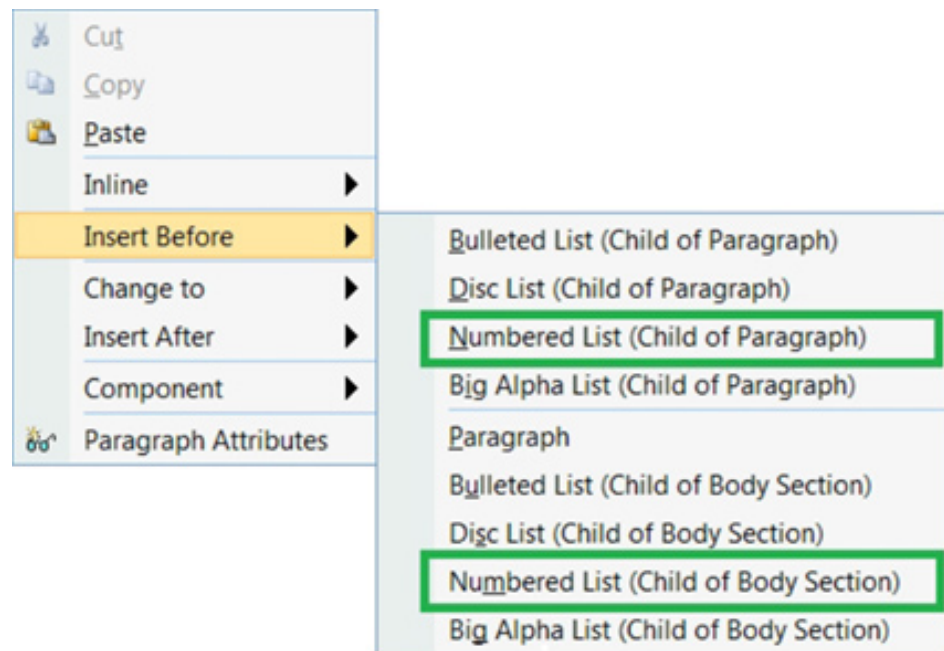


Figure 9-1: Example of Insert Before submenu

PasteTrackDeletedText

If set to true, this will enable pasting track deleted text.

Defaults to **false**

```
<PasteTrackDeletedText>false</PasteTrackDeletedText>
```

ShowInsertAfterMenu

The <ShowInsertAfterMenu> node allows you to specify whether or not “Insert After” menus are displayed in the Context Menu for the current element.

Optional. Boolean. Defaults to **true**.

Insert After menus contain the list of insertable elements that the user is allowed to insert after a given element. The list is the same as if the user right-clicked on the Insertion Point below the given element.

DOCUMENT LEVEL CONFIGURATION

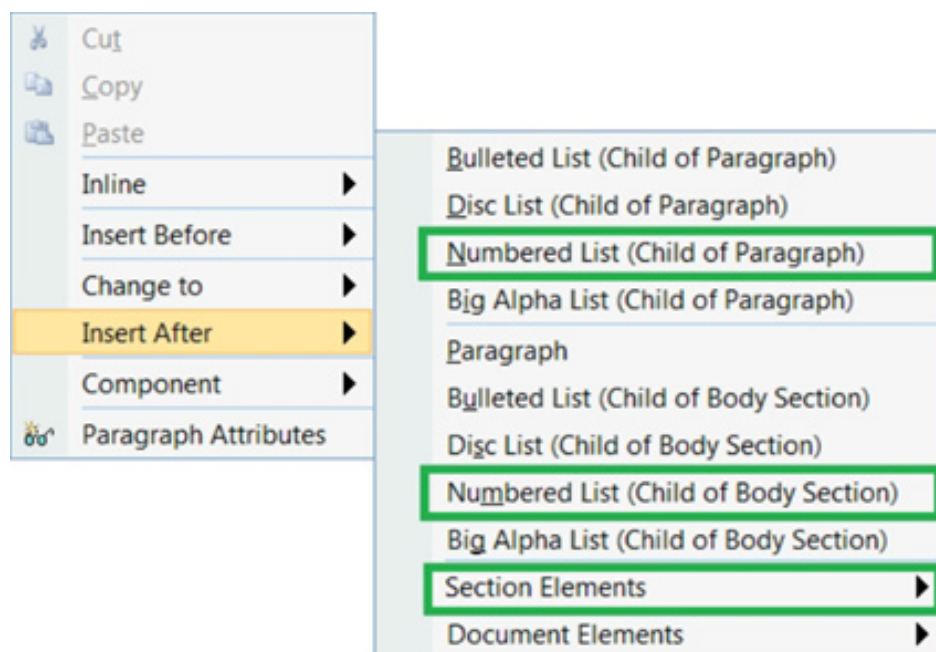


Figure 9-2: Example of Insert After submenu

EnableFastSave

If set to true, this will enhance save and preview operations by optimizing serialization, resulting in lowering the amount of time taken in save and preview operations.

Defaults to **False**.

```
<EnableFastSave>true</EnableFastSave>
```

EnableOpenDocxSupport

To Enable OpenDocx support in Quark XML Author, add the tag `<EnableOpenDocxSupport>true</EnableOpenDocxSupport>` in the document configuration. This is enabled by default in the Quark XML Author adapter for Platform (BUSDOCS) configuration. This feature is not available in the standalone version of Quark XML Author as it requires Quark Publishing Platform.

dir

Bidirectional text flow support in Quark XML Author at the document level is achieved by setting the base text direction for an entire document. This is accomplished using the **dir** attribute on the root element. Valid values for the **dir** attribute are **ltr** and **rtl** (left-to-right and right-to-left). Defaults to **ltr**.<topic id ="" xmlns:inv="urn:xpressauthor:xpressdocument" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xlink="http://www.w3.org/1999/xlink" xsi:noNamespaceSchemaLocation="BUSDOCS.xsd" xml:lang="ar" dir="rtl">

Application Configuration: XpressUpdates

Overview

Xpress Updates is a mechanism for pushing Quark XML Author software and configuration updates to existing Quark XML Author clients. Xpress Updates may update any Quark XML Author file except for the XpressAddIn.dll, including configuration files, starter documents, and XAS files.

Xpress Updates is loaded when Word is started, but before the Quark XML Author Add-in is loaded. Xpress Updates checks a manifest document located on a server. This manifest document contains information about files that are available for download. If the local files have an earlier timestamp than the available files, the user is notified that updates are available and prompted to download them. If the user selects Yes, the files are downloaded and Quark XML Author is loaded. If the user selects No, Quark XML Author is loaded without the new files being downloaded.



Note that the user must have write permission to the XML Author directory. If the user does not have write permission to the XML Author directory, the update operation will fail.

Xpress Updates does not provide end user control over the update mechanism. While the user may ignore or cancel an update through the dialog box, the user may not disable the update mechanism once the administrator has configured it. Nor may the user enable the update mechanism if the administrator disables it.

XpressUpdate.xml

XpressUpdate.xml is the configuration file for Xpress Updates, and is located in the XML Author application folder. It should be configured before building the client installation. Otherwise, the file would need to be copied manually to each client.

The file has the following structure:

```
<XpressUpdate xmlns="urn:invision:XpressUpdate">
```

```
<ManifestLocation></ManifestLocation>
<Credentials username="" password=""
domain="">Windows</Credentials>
<LogFile logging="true">XpressUpdateLog.txt</LogFile>
</XpressUpdate>
```

Each node is defined below.

ManifestLocation: The URL pointing to the manifest file that contains the file update information.

Credentials: Used during authentication on the update server. If the node contents specify **Windows**, the user's Windows credentials are used. If the node contents specify **Supplied**, the values of the username, password, and domain attributes are used.

LogFile: If set to **true**, logging is turned on, and the node content specifies the name of the log file. If set to **false**, logging is turned off. The log file is located in the XML Author application folder.

Manifest.xml

Manifest.xml is the document returned by the server, and contains information about the files that are available to be updated. The file has the following structure:

```
<XpressUpdateManifest xmlns="urn:invision:XpressUpdateManifest">
  <FileRepository>http://localhost/XpressUpdates</FileRepository>
  <Files>
    <File name="file1" location="" dateTime="1/27/2005 10:33
PM"/>
    <File name="file2" location="" dateTime="1/27/2005 10:33
PM"/>
  </Files>
</XpressUpdateManifest>
```

The **FileRepository** node content specifies the location of the files available for download. Each **File** node in the **Files** element specifies the name, location, and timestamp for one available file. The **File** node attributes are defined below.

name: The name of the file that is available to be downloaded.

location: The path relative to the client application folder where the file will be downloaded to. For example, if the file will be downloaded to the \XML Author\quark\stylesheets directory, this value would be **quark\stylesheets**. If this attribute is omitted or left empty, the file is copied to the XML Author application folder.

dateTime: The timestamp to which current files are compared. If the existing file is older than the timestamp, it is replaced.

Dialogs

Xpress Updates uses two dialogs to allow the user to monitor update status as well as cancel updates if necessary. The first dialog informs the user that updates are being checked. This dialog will only be displayed if more than 3 seconds elapse

before the manifest document is returned from the server. The dialog will automatically close when the manifest is finally retrieved. The user may click the **Cancel** button if the retrieval fails or takes too long.

The second dialog displays the files that are being downloaded and their status (Downloading, Complete, or Error). The dialog remains open until the user clicks the **OK** button. The user may click the **Cancel** button at any time to abort file downloading.

Quark XML Author Structure Overview

The Quark XML Author Structure (XAS) is the Quark XML Author equivalent of an XML schema file, in that it defines the allowed components of a document, and their relationships to each other. Quark XML Author uses the XAS to provide more flexibility than would be present with a standard XML Schema Definition (XSD) or Document Type Definition (DTD). This chapter presents an overview of the XAS structure. Subsequent chapters will discuss how each component in an XAS is constructed.

XML Declaration and Processing Instructions

Because the XAS is an XML document, it must begin with an XML declaration, as shown below:

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

Defining Multiple Namespaces

The root element defines the default namespace so that you don’t need to specify it for each element you define. However, each attribute, emphasis, and element definition can specify additional namespaces that will override the default namespace within the documents that are based on the XAS.

Root Element

The root element for the XAS is XpressSchema. Its required attributes are defined in Table 11-1.

Table 11-1: Schema Root Element Attributes and Values

Attribute	Value
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation	urn:xpressauthor:XpressSchema ../Schemas/XpressSchema.xsd

External Entities

The DTD ENTITY declaration and entity references can be used to link separate documents into a single Quark XML Author Schema. The schema validation is still supported but first all the different entities are combined into one file before validating it. An example is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE link [
  <!ENTITY secattributes SYSTEM "SecurityAttributes.xas">
  <!ENTITY emphattributes SYSTEM "EmphasisAttributes.xas">
  <!ENTITY mediaattributes SYSTEM "MediaAttributes.xas">
  <!ENTITY emailattributes SYSTEM
"EmailNotificationAttributes.xas">
  <!ENTITY miscattributes SYSTEM "MiscellaneousAttributes.xas">
  <!ENTITY tableattributes SYSTEM "tableAttributes.xas">
  <!ENTITY topLevelElements SYSTEM "topLevelElements.xas">
  <!ENTITY localMediaElements SYSTEM "LocalMediaElements.xas">
  <!ENTITY slideElements SYSTEM "PowerPointSlideElements.xas">
]>
<XpressSchema xmlns="urn:xpressauthor:xpressschema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
defaultNamespace="urn:us:gov:ic:msp"
xsi:schemaLocation="urn:xpressauthor:xpressschema
../schemas/XpressSchema.xsd">
  &secattributes;
  &emphattributes;
  &mediaattributes;
  &emailattributes;
  &miscattributes;
  &tableattributes;
  <Emphasis>
    .....
  </Emphasis>
  <Elements>
    ....
  </Elements>
  &localMediaElements;
  &slideElements;
  <Elements>
    ....
  </Elements>
  &topLevelElements;
</XpressSchema>
```

Definition Order

Document components are declared in reverse hierarchical order. That is, the document's root node is the last item declared in the document. The root element's immediate children are declared just above it, and their children just above them, and so on, with elements that can contain no other elements declared before any others. Emphasis styles are referenced by elements, so they are defined above the element definitions. Both emphasis definitions and element definitions reference attributes, so attributes comprise the first section of the XAS.

Attributes

Of course, an XML document is composed of more than just elements, and a Quark XML Author document is no different. Elements have attributes which contain metadata describing the document and its contents. In the XAS, attributes are

declared separately from elements, and are then referenced in each element definition.

Like child elements, attributes must be declared before the element to which they will be applied. For this reason, Attributes are defined at the beginning of the XAS

Emphasis Styles

The next section of the XAS defines Emphasis styles. Each emphasis style can also reference attributes, but they, in turn, will be referenced by elements. This section is therefore located below the Attributes section, but above the Elements sections.

The structure of the file looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
  <XpressSchema xmlns="urn:invision:XpressSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:invision:XpressSchema
../Schemas/XpressSchema.xsd"
defaultNamespace="urn:invision:enduserdoc">
  <Attributes>
    <!-- Individual attribute definition elements and child
elements -->
  </Attributes>
  <Emphasis>
    <!-- Individual emphasis definition elements and child
elements -->
  </Emphasis>
  <Elements>
    <!-- Individual element definition elements and child
elements -->
  </Elements>
</XpressSchema>
```

Elements

Figure 11-1 shows a simple element structure.



Figure 11-1: Simple Element Structure

In defining the RevisionHistory element and the elements that make up its structure, the first item defined would be RevisionRemark. Next, Revision would be defined, and would reference RevisionRemark as a required child element, and specify how many are permitted. Finally, RevisionHistory would conclude the definition, and would reference Revision as a required child element, and specify how many are permitted.

Comment Lines

It can be helpful to isolate the various elements and attributes that will be used according to category. This makes it easier to read the XAS later, and can help prevent difficulties such as declaring components in the wrong order. Doing this is not required, nor is there a “right” way to do it. However, the following categorization method has been shown to be very useful:

- **Attributes.** Metadata attributes for elements and emphasis styles are defined first. Elements and emphasis styles may share attributes.
- **Emphasis.** Each type of emphasis is declared and defined. If you have a complex set of Emphasis requirements, you may want to break this section down further.
- **Miscellaneous Elements.** These are all of the elements other than list, table, and media elements that can be contained by top-level elements.
- **List Elements.** This section will contain any element that can be a part of a list but cannot be a child of any other element type.
- **Table Elements.** This section will contain any element that a Table element comprises.
- **Media Elements.** This section will contain any element that relates specifically to media elements.
- **Container Elements.** Elements that can contain almost every other type of document component, but can themselves also be contained by top-level elements, are placed in this section. For example, sections and subsections of a chapter might be defined here.
- **Top-Level Elements.** The elements which are container elements for the rest of the elements in the document class, but are themselves children to the root element, are declared here, followed by the root element itself.

A file using this method of categorization might look like the one shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<XpressSchema xmlns="urn:invision:XpressSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"xsi:schemaLocation="urn:invision:XpressSchema
../Schemas/XpressSchema.xsd"
defaultNamespace="urn:invision:enduserdoc">
  <Attributes>
    <!-- Element Attributes -->
  </Attributes>
  <Emphasis>
    <!-- Emphasis Styles -->
  </Emphasis>
  <Elements>
    <!-- Miscellaneous Elements -->
    <!-- List Elements -->
    <!-- Table Elements -->
    <!-- Media Elements -->
    <!-- Container Elements -->
    <!-- Top-Level Elements -->
  </Elements>
</XpressSchema>
```

Attribute Groups

XpressSchema.xsd identifies two attribute groups which can be associated with an element by reference, obviating the need to define each attribute separately. These groups are common attributes and field attributes. Attributes belonging to each of these groups are defined below.

Note that these are not the only attributes that may be used, and at times, an attribute from a group might be applied separately to an element. The particular

attributes that may be applied to each element are identified in the Attribute Definition Elements section below.

Common Attributes

Common attributes serve to identify the elements in the schema.

Table 11-2: Common Attributes

Attribute	Description	Example
name	Specifies the name of the attribute definition.	<code><AttributeDef name="section001"></code>
namespace	Specifies the namespace to use for the element, and overrides the default namespace. This allows the use of multiple schemas in a single document.	<code><AttributeDef name="section01" namespace="sidebar"></code>
prefix	If the namespace attribute is used, it must be accompanied by this attribute, which will append the specified prefix to the element name in the output XML, associating the element with the overriding namespace.	<code><AttributeDef name="MediaReference" namespace="urn:invision: msp-extensions" prefix="inv"></code>
xmlname	Specifies the name that will be used as the element name in the XML file. Used to allow an attribute to have distinct enumerated values depending on which element it is associated with. Create a separate AttributeDef for each attribute instance and assign the same xmlname value to each one.	<code><AttributeDef name="section01" xmlname="section"></code>

Field Attributes

Table 11-3: Field Attributes

Attribute	Description	Example
attrGroupName	Specifies the name of the tab in the attribute panel on which the attribute being defined will appear.	<code><AttributeDef name="pubdate" datatype="datetime" format="MMDDYYYY"></code>

Attribute	Description	Example
		<pre>friendly="Publication Date" attrGroupName="Publicati on Information"/></pre>
datatype	<p>Specifies the type of data to be supplied. Possible values are combobox, datetime, droplist, imageheight, imagewidth, imageheightdpi, imagewidthdpi, multiline, string, time, uniqueidentifier, columnBreak, pageBreak, and sectionBreak.</p> <p>See Table 11-4 for descriptions of these datatypes.</p> <p>See Table 11-5 for descriptions of table specific datatypes.</p> <p>Default is string.</p>	<pre><AttributeDef name="id" datatype="uniqueidentifi er"/></pre>
default	Specifies a default value for the attribute being defined.	<pre><AttributeDef name="securClassif" friendly="Security Classification" attrGroupName="Releasabi lity" default="TS"/></pre>
displayPattern	Used in the error message displayed to inform the user that the data entered does not match the pattern.	
externalMethodId	Specifies an externalMethodId that provides a string that XML Author will use to populate an attribute's combo box list in the Attributes Panel.	<pre><AttributeDef name="author" friendly="Author Name" externalMethodId="TechPu bsStaff"/></pre>
fixed	Assign the fixed attribute to a AttributeType for an element definition to uniquely identify the element among sibling elements that have identical xmlname values. The value of fixed is a string that is unique within the collection of sibling	<p>In this example, both ElementDefs define elements that are siblings. Both will be present in the XML document as <chapter> nodes, but the chapterType attribute for those nodes will be set according to the fixed</p>

Attribute	Description	Example
	element definitions.	<p>attribute. This allows sibling elements with the same XML name to have entirely different structure definitions.</p> <pre> <ElementDef name="instruction" friendly="Instructional Chapter" xmlname="chapter"> <Sequence> <!-- Sequence definition information - -> </Sequence> <Attributes> <AttributeType name="chapterType" visible="true" fixed="instruct"/> <!-- Other AttributeTypes --> </Attributes> </ElementDef> <ElementDef name="concept" friendly="Conceptual Chapter" xmlname="chapter"> <Sequence> <!-- Sequence definition information - -> </Sequence> <Attributes> <AttributeType name="chapterType" visible="true" fixed="concept"/> <!-- Other </pre>

Attribute	Description	Example
		<pre>AttributeTypes --> </Attributes> </ElementDef></pre> <p>Also see Field Attributes – fixed example.</p>
format	<p>Specifies a format for the data to be supplied. For example, MMDDYYYY represents a date field that is specified as Month, Day, and Year.</p> <p>See “Valid date formats” more a list of valid date formats.</p>	<pre><AttributeDef name="pubdate" datatype="datetime" format="MMDDYYYY"/></pre>
friendly	<p>Specifies a user-friendly name for the attribute that will be displayed to the user in the attribute panel.</p>	<pre><AttributeDef name="pubdate" datatype="datetime" format="MMDDYYYY" friendly="Publication Date"/></pre>
keepWithNext	<p>Boolean. Defaults to false. When set to true, creates an attribute that, when present on an ElementDef, indicates that the element may be toggled so that it will break across a new page with the element that follows it. For example, consider this AttributeDef:</p> <pre><AttributeDef name="keep- with-next" friendly="Keep with Next" keepWithNext="true" visible="false"/></pre> <p>Any ElementDef that includes the keep-with-next attribute defined above could be toggled to mimic Word’s “Keep with Next” paragraph behavior.</p> <p>This feature does not support images. For tables, this feature supports table rows only.</p>	

Attribute	Description	Example
pattern	When specified, selecting OK in the Attribute Panel validates the field against the pattern. If the attribute is required, the validation error will also be triggered if the field is null.	
readonly	Specifies whether the value of the attribute will be read-only. Defaults to false .	<pre><AttributeDef name="documentType" default="Memo" readonly="true"/></pre>
required	Specifies whether the attribute value is mandatory. Defaults to false .	<pre><AttributeDef name="author" required="true"/></pre>
visible	Specifies whether the attribute will be visible in the Attribute Panel. Defaults to true .	<pre><AttributeDef name="documentType" default="Memo" visible="false"/></pre>
widowControl	<p>Boolean. Defaults to false. When set to true, creates an attribute that, when present on an ElementDef, indicates that the element may be toggled so that it will allow widowed and orphaned text to be left at the bottom and tops of pages, respectively. For example, consider this AttributeDef:</p> <pre><AttributeDef name="widow-control" friendly="Widow Orphan control" default="true" widowControl="true" readonly="true" visible="false"/></pre> <p>Any ElementDef that includes the widow-control attribute defined above could be toggled to mimic Word's "Widow/Orphan Control" paragraph behavior.</p>	
dir	Specifies the text direction in an element or document.	

Attribute	Description	Example
	Defaults to ltr (left to right).	

Valid date formats

The following link provides a list of [valid date formats](#)

Field Attributes – fixed example

This is an additional example to accompany the definition of the fixed attribute. See Table 11-3: Field Attribute

If the schema has `<p/>` defined three different ways, one is text-only, one is children-only (no text), and one is mixed (allows text and children). Quark XML Author must be able to differentiate within the configuration between the elements in order to work with documents containing structures like the following.

```
<p> I'm a text only para</p>
<p>
  <child>
  <child>
  <child>
</p>
<p> I can have text and children
  <child>
</p>
```

The following shows how those elements would be configured:

```
<AttributeDef name="para_type" visible="false"
readonly="true"/>

<ElementDef xmlns="p" name="p_text_only" friendly="Text
Paragraph" />
  <Attributes>
    <AttributeType name="para_type" fixed="textOnly"/>
  </Attributes>
</ElementDef>
<ElementDef xmlns="p" name="p_children_only"
friendly="Paragraph Section" />
  <Attributes>
    <AttributeType name="para_type" fixed="childrenOnly"/>
  </Attributes>
</ElementDef>
<ElementDef xmlns="p" name="p_mixed" friendly="Paragraph" />
  <Attributes>
    <AttributeType name="para_type" fixed="mixed"/>
  </Attributes>
</ElementDef>
```

The following content example shows the three para elements in use:

```
<p para_type="textOnly"> I'm a text only para</p>
<p para_type="childrenOnly">
  <child>
```

```

    <child>
    <child>
  </p>
  <p para_type="mixed"> I can have text and children
    <child>
  </p>

```

Table 11-4: Field Attributes – datatype values

datatype	Description	Example
columnBreak	<p>Set an AttributeDef's datatype to columnBreak to indicate a Column Break attribute.</p> <p>See section 9.7 for information on DocConfig settings. See section 12.3 for information about creating and using break attributes.</p>	
datetime	This attribute stores the date and time.	
droplist	droplist lets users select from a list of values populated from values specified in the schema.	
imageheight, imagewidth, imageheightdpi, imagewidthdpi	<p>In general, these attributes are set when an image is inserted, refreshed, or resized. When a document is opened, if these attributes do not exist then they are added and they are set using the values retrieved from the image file. When an image is inserted, these attributes are added and they are set using the values retrieved from the image file. In those two scenarios, if the system determines that the image size is larger than the area available on the Word canvas, then the system will reduce the size of image to fit while maintaining the aspect ratio. These attributes are set when the user manually alters the size of the image on the Word canvas. The user may set the size to be larger than the area available on the Word canvas.</p> <p>imageheight and imagewidth values are in pixels.</p>	
multiline	multiline indicates that the	

datatype	Description	Example
	textbox for the attribute will have three lines, will support carriage returns, and display a tooltip showing the entire value of the attribute when the mouse hovers over it.	
pageBreak	Set an AttributeDef's datatype to pageBreak to indicate a Page Break attribute. See section 9.7 for information on DocConfig settings. See section 12.3 for information about creating and using break attributes.	
sectionBreak	Set an AttributeDef's datatype to sectionBreak to indicate a Section Break attribute. See section 9.7 for information on DocConfig settings. See section 12.3 for information about creating and using break attributes.	
string	A simple string. This is the default datatype.	
time	This attribute stores the time.	
uniqueidentifier	If the value is uniqueidentifier , a GUID will be generated as the value of the attribute.	
directionAttribute	Allows support for direction attributes for elements.	<code><AttributeDef name="dir" datatype="directionAttribute" visible="false"/></code>

Table 11-5: Field Attributes – datatype values for table elements

The following datatypes can be applied to all table elements and be inherited in the order of Table > Group > Section > Row | Column > Cell (meaning that the cell inherits from the Column only if it is explicitly defined on the Column and not on the Row).

Datatype	Description	Example
tableBorderBottomStyle	Set to 1 to style the bottom	<code><AttributeDef</code>

Datatype	Description	Example
	border using the style specified in the Table Style. Set to 0 to use no styling. Note that if rowBorderStyle is set to None , then setting tableBorderBottomStyle to 1 will have no effect. Styling is not applied. This applies to the bottom border of the table element, not including the table frame.	<pre>name="rowsep" datatype="tableBorderBottomStyle" friendly="Border Style - bottom" visible="false"/></pre>
tableBorderRightStyle	Specifies the right border of the table element, not including the table frame.	<pre><AttributeDef name="colsep" datatype="tableBorderRightStyle" friendly="Border Style - right" visible="false"/></pre>
tableAlignmentHorizontal	Specifies the horizontal alignment of the content.	<pre><AttributeDef name="align" datatype="tableAlignmentHorizontal" friendly="" visible="false" default="left"> <Restriction> <Enumeration value="left"/> <Enumeration value="right"/> <Enumeration value="center"/> <Enumeration value="just"/> </Restriction> </AttributeDef></pre>
tableAlignmentVertical	Specifies the vertical alignment of the content.	<pre><AttributeDef name="valign" datatype="tableAlignmentVertical" friendly="" visible="false" default="top"> <Restriction> <Enumeration value="top"/> </Restriction> </AttributeDef></pre>

Datatype	Description	Example
		<pre> <Enumeration value="middle"/> <Enumeration value="bottom"/> </Restriction> </AttributeDef> </pre>

Table 11-6: Field Attributes – datatype values for tables

Table-level datatypes.

Datatype	Description	Example
tableFrame	Specifies the border of the table, not including cells.	<pre> <AttributeDef name="frame" datatype="tableFrame" friendly="Table Frame Borders" visible="false" default="all"> <Restriction> <Enumeration value="all"/> <Enumeration value="top"/> <Enumeration value="bottom"/> <Enumeration value="topbot"/> <Enumeration value="sides"/> <Enumeration value="none"/> </Restriction> </AttributeDef> </pre>

Table 11-7: Field Attributes –datatype values for table rows/columns

tableRowColumnIndex	Specifies the index of the row/column that contains the cell. Required for merged cells in CALS tables.	<pre> <AttributeDef name="colnum" datatype="tableRowColumn Index" friendly="" visible="false"/> </pre>
---------------------	---	--

tableRowColumnIdentifier	The name or identifier of the row/column that contains the cell. Required for merged cells in CALS tables.	<code><AttributeDef name="colname" datatype="tableRowColumn Identifier" friendly="" visible="false"/></code>
--------------------------	--	--

Table 11-8: Field Attributes – datatype values for table cells

Datatype	Description	Example
tableCellColumnStart	For a merged cell, specifies the column in which the cell starts. Used to facilitate CALS namestart.	<code><AttributeDef name="namestart" datatype="tableCellColumn Start" friendly="" visible="false"/></code>
tableCellColumnEnd	For a merged cell, specifies the column in which the cell ends. Used to facilitate CALS nameend.	<code><AttributeDef name="nameend" datatype="tableCellColumn End" friendly="" visible="false"/></code>
tableCellColumnCount	For a merged cell, specifies the number of columns the cell spans. Used to facilitate HTML colspan and horizontal merge.	<code><AttributeDef name="namecount" datatype="tableCellColumn Count" friendly="" visible="false"/></code>
tableCellRowCount	Specifies the number of additional rows in the merge (equals total rows minus one). Required for vertically merged cells in CALS tables.	<code><AttributeDef name="cellrowcount" datatype="tableCellRowCo unt" friendly="" visible="false"/></code>
tableCellContentRotate	Specifies that the content of the cell should be rotated.	<code><AttributeDef name="rotate" datatype="tableCellConte ntRotate" friendly="" visible="false"/></code>
tableCellBackgroundColor	Specifies the background color (hex RGB) of the cell. To display a specific set of color	<code><AttributeDef name="bkgndcolor" datatype="tableCellBackg</code>

Datatype	Description	Example
	choices to the user, use Restriction Enumerations. Otherwise, the user interface displays Word's stock colors for cell shading.	<pre> roundColor" friendly="" visible="false" > <Restriction> <Enumeration value="F0F8FF "/> <Enumeration value="FAEBD7"/> </Restriction> </AttributeDef> </pre>
tableCellRowSpan	Specifies the total number of rows in the merge. Required for vertically merged cells in HTML tables. Used to facilitate HTML rowspan.	<pre> <AttributeDef name="rowspan" datatype="tableCellRowSp an" visible="false"/> </pre>
tableCellMoreRows	Specifies the number of additional rows in the merge (equals total rows minus one). Required for vertically merged cells in CALS tables. Used to facilitate CALS morerows. Behaves identically to the deprecated tableCellRowCount. It is added for clarity for the engineer implementing CALS. The current value of tableCellRowCount is preserved for backward compatibility despite the misleading name.	<pre> <AttributeDef name="morerows" datatype="tableCellMoreR ows" visible="false"/> </pre>

Table 11-9: Field Attributes – datatype values for table, group, section

Datatype	Description	Example
tableColumnCount	For a table or table group, specifies the number of columns it contains.	<pre> <AttributeDef name="cols" datatype="tableColumnCou nt" friendly="" visible="false"/> </pre>
tableColumnWidth	Specifies the width of the	<pre> <AttributeDef </pre>

Datatype	Description	Example
	column. Required for merged cells in CALS tables. Required to allow the width of columns to be modified. See “Column Widths” for details.	<code>name="colwidth" datatype="tableColumnWidth" friendly="" visible="false"/></code>
tableRowCount	For a table, table group or table section, specifies the number of rows it contains. Specifies the height of the row. Required to allow the height of the rows to be modified.	<code><AttributeDef name="rows" datatype="tableRowCount" friendly="" visible="false"/></code>
tableRowHeight	Specifies the height of the row. Required to allow the height of rows to be modified.	<code><AttributeDef name="rowHeight" xmlname="height" datatype="tableRowHeight" visible="false"/></code>
tableStyle	Specifies the name of the style of the table. For tables or groups only.	<code><AttributeDef name="tgroupstyle" datatype="tableStyle" friendly="" visible="false"/></code>

Quark XML Author Structure Attribute Definition

Each element that makes up the attribute definition section of the schema is defined below. An example is provided with each to demonstrate proper syntax for the element's use.

The Attributes element contains the collection of attribute definition elements for the schema, and is a direct child of the XpressSchema root element. It uses no attributes.

```
<Attributes>
  <AttributeDef name="id" visible="false"/>
  <!-- Additional AttributeDef elements here -->
</Attributes>
```

AttributeDef

AttributeDef defines a single attribute for the schema. It may use the Restriction child element to specify a restricted list of values from which the user must choose when populating the attribute.

AttributeDef may use any of the attributes defined in the Common Attributes and Field Attributes groups, as well as the optional attributes defined below.

Table 12-1: AttributeDef Attributes

Attribute	Required	Definition
contextId	no	A dash-delimited list of rule IDs in a validation configuration file.

Restriction

Restriction is used as a child of the AttributeDef element to restrict the user to a pre-determined list of values that may be used to populate the defined attribute. Each possible value is defined in an Enumeration child element. Restriction has no attributes, but must have one or more Enumeration children.

```
<AttributeDef name="filter" friendly="Filter" default="publish">
  <Restriction>
    <Enumeration value="publish"/>
    <Enumeration value="supress"/>
  </Restriction>
</AttributeDef>
```

Enumeration

Enumeration defines a single value that may be used to populate the defined attribute. Enumeration has one required attribute: `value`. This attribute is a string representing the value that may be selected by the user to populate the defined attribute

Using uniqueidentifier

Quark XML Author supports attribute definitions with `datatype="uniqueidentifier"` for automatically assigning a GUID value. When used without the `format` attribute, the attribute is populated in the format: `id=" 982x0d0c-1446-4b39-a704-646d549b50b3"`. The `xsd:ID` datatype requires the value to begin with an alpha character. To comply with this requirement, use `datatype="uniqueidentifier"` in conjunction with the `format` attribute. The value of the `format` attribute will be prepended to the GUID. For example:

```
<AttributeDef name="id" datatype="uniqueidentifier" format="id-"
required="true"/>
```

The preceding example would produce a value in the format `id=" id-982x0d0c-1446-4b39-a704-646d549b50b3"`.

Associating Attribute Definitions with Elements and Emphasis

Both element definitions (section 14) and emphasis definitions (section 13) can have attributes associated with them. The `ElementDef` and `EmphasisDef` elements each have an optional child element called `Attributes`. The `Attributes` element contains an `AttributeType` element for each attribute that is associated with the element or emphasis. The `AttributeType` element's **name** attribute contains the name of an `AttributeDef` defined earlier in the schema. The `AttributeType` element can have any of the Field Attributes defined in section 11.6.2. An example is shown below.

```
<Attributes>
  <AttributeType name="author"/>
  <AttributeType name="copyright"/>
  <AttributeType name="name"/>
  <AttributeType name="editor"/>
  <AttributeType name="keywords"/>
  <AttributeType name="status"/>
  <AttributeType name="id"/>
  <AttributeType name="revisionflag"/>
</Attributes>
```

SectionBreak, PageBreak, and ColumnBreak Attributes

To specify whether or not a given element may be preceded by a section, page break, or column break, you must define an attribute with the `datatype` value `sectionBreak`, `pageBreak`, or `columnBreak`. For example:

```
<AttributeDef name="sectionBreak" datatype="sectionBreak"
readonly="true" visible="false"/>

<AttributeDef name="pageBreak" datatype="pageBreak"
readonly="true" visible="false"/>
```

```
<AttributeDef name="columnBreak" datatype="columnBreak "
readonly="true" visible="false"/>
```

You would then associate one or more of these attributes with that specific element. Information about associating AttributeDefs with elements is found in section 12.2.

Finally, you would set the value of the SectionBreak, PageBreak, ColumnBreak node in the DocConfig to **restricted**. See section 9.7 for information about these DocConfig nodes.

Quark XML Author Structure Emphasis Definition

Each element that makes up the emphasis definition section of the schema is defined below.

The Emphasis element contains the collection of emphasis definition elements for the schema, and is a direct child of the XpressSchema root element. It uses no attributes.

EmphasisDef

EmphasisDef defines a single emphasis style that may be associated with text elements in the schema. Its Style child element defines the actual appearance of the emphasis within the document. EmphasisDef may use any of the attributes defined in the commonAttributes group. It may also use the attributes defined below.

Table 13-1: EmphasisDef Attributes

Attribute Name	Required	Definition
contextId	no	A dash-delimited list of rule IDs in a validation configuration file.
excludeFromEmphasisCombo	no	Defaults to false . When set to true , the emphasis is not visible in the drop-down box.
friendly	no	Specifies a name for the emphasis style to be presented to the user within the interface, instead of the emphasis element name
preserveSpace	no	Defaults to false . When set to true , the emphasis will include any trailing space/non-breaking-spaces that are part of the selection. For example, the trailing space/non-breaking-space that is selected when a user double-clicks on a word.
readonly	no	Defaults to false . If set to true ,

Attribute Name	Required	Definition
		the textnode of a readonly emphasis may not be changed. The emphasis can be deleted if its parent container is editable. Emphasis may not be nested within a readonly emphasis.
selectOnFocus	no	Defaults to false . When set to true , Quark XML Author will auto-select the entire emphasis when the user clicks or selects within the emphasis region. If the user selects a range that includes but extends outside the emphasis, Quark XML Author does not perform the auto-selection. This allows the user to select and over-type the emphasized text (assuming that inv:required is not true).
tooltipTransform	no	Defines the path to an XSLT file used to define the text displayed when the user hovers over the emphasized text.

Specialized

Many features of Quark XML Author, such as footnotes, end notes, and cross reference links, are implemented as emphasis. The <Specialized> node is used to define enhanced emphasis functionality.

The specialized node's child elements define the emphasis behavior. These are <Footnote>, <Endnote>, <Link>, <Entity>, and <ProcessingInstruction>.

Footnote

Use the <Footnote> element to establish presentation options for document footnotes.

Table 13-2: Footnote Attributes

Attribute Name	Required	Definition
allowedNumberingSchemes		Used for footnote emphasis. Specifies the allowed numbering formats that can be used for note items. If more than one is allowed, they are separated by spaces. Possible values are:

Attribute Name	Required	Definition
		<ul style="list-style-type: none"> • arabic • bigAlpha • smallAlpha • bigRoman • smallRoman
captionAttribute	no	Specifies the name of the EmphasisDef attribute that will contain the value of a custom mark assigned by the author at runtime.
disableCustomMarkTextbox	no	Boolean. Set to true if the Custom Mark option should be disabled in the Footnote Options dialog. Defaults to false .
footnoteLocation	yes	Determines where footnotes will be placed in the rendered document. Valid values are document , page , section , and text .

The sample below shows one possible Footnote configuration.

```
<EmphasisDef name="fn" friendly="Footnote">
  <Specialized>
    <Footnote footnoteLocation="page"
allowedNumberingSchemes="smallRoman" captionAttribute="callout"/>
  </Specialized>
  <Style superscript="true"/>
  <Attributes>
    <AttributeType name="callout"/>
  </Attributes>
</Emphasis/>
</EmphasisDef>
```

Endnote

Use the <Endnote> element to establish presentation options for document endnotes and tablenotes. Endnotes are generalized to facilitate notes that are placed at the end of a content element. Therefore, endnote is used to facilitate tablenote.

Table 13-3: Endnote Attributes

Attribute Name	Required	Definition
allowedNumberingSchemes		Used for endnote and tablenote emphasis. Specifies the allowed numbering formats that can be used


```

        </Emphasis>
    </Para>
</Section>
</ElementDef>
<!-- - - - - -
- - - - -
- - - -->
<ElementDef name="table" friendly="Table" visible="false">
    <Section>
        <Sequence minOccurs="0">
            <SectionType name="title" friendly="Table Title"/>
        </Sequence>
        <Sequence>
            <SectionType name="tgroup"/>
        </Sequence>
        <Sequence>
            <!-- minOccurs = 0 and maxOccurs=1 always for a EndnoteType
            element -->
            <!-- Furthermore there can only be one reference of a
            particular EndNoteType element in the entire schema-->
            <EndnoteType name="Tablenote"/>
        </Sequence>
        <Sequence minOccurs="0">
            <SectionType name="desc" friendly="Table Caption"/>
        </Sequence>
    </Section>
    <Attributes>
        <AttributeType name="id"/>
        <AttributeType name="conref"/>
        <AttributeType name="frame"/>
        <!--
                                <AttributeType name="rowsep"/>
                                <AttributeType name="colsep"/>
    -->
    </Attributes>
</ElementDef>
<!-- - - - - -
- - - - -
- - - -->
<EmphasisDef name="tn" friendly="Tablenote" >
    <Specialized>
        <Endnote endnoteElementDef="Tablenote"
        allowedNumberingSchemes="arabic" captionAttribute="callout"
        disableCustomMarkTextbox="true"/>
    </Specialized>
    <Style superscript="true"/>
    <Attributes>
        <AttributeType name="callout"/>
        <AttributeType name="class"/>
        <AttributeType name="outputclass"/>
    </Attributes>
    <!-- The Formatting attributes gives the ability to apply
    formatting attributes to a Tablenote

    <Formatting>
        <FormattingType name="b"/>
        <FormattingType name="i"/>
        <FormattingType name="u"/>
        <FormattingType name="strikethrough"/>
        <FormattingType name="sub"/>
        <FormattingType name="sup"/>
        <FormattingType name="color"/>
        <FormattingType name="highlight"/>
    </Formatting>
</EmphasisDef>

```

Link

Use the <Link> node to define the behavior of hyperlink emphasis. Hyperlinks can be defined as <Internal> (target is within the document), <External> (target is another document or a URL), or both, via <Multiple>. The <Link> node may also use the <Custom> node to indicate that the link creation and navigation behavior will be handled by an external application.

External

Use the <External> node to indicate a hyperlink to an external target, such as a different document, a component within a different document, or a URL. The <External> node can be used as a direct child of <Link> or it can be used as a child of <Multiple> when a combination of internal and external links is desired.

When used as a child of <Multiple>, the substyle attribute is required. The value of the substyle attribute specifies the name of a substyle, defined in the <SubStyles> node of the <EmphasisDef>, that defines the appearance of the external link.

When used as a child of <Link>, the appearance of the link is specified by the <Style> node.

Table 13-4: External Attributes

Attribute Name	Definition
substyle	Required when used as a child of <Multiple>. Specifies the name of the substyle that defines the appearance of the external link.

Internal

Use the <Internal> node to indicate a hyperlink to an internal target, such as a component within the document. One or more child <Target> nodes define the elements that may be link targets.

The configuration attribute specifies the file which contains the Link Management configuration. Typically, this is the DocConfig file for the document class.

When used as a child of <Multiple>, the substyle attribute is required. The value of the substyle attribute specifies the name of a substyle, defined in the <SubStyles> node of the <EmphasisDef>, that defines the appearance of the external link.

When used as a child of <Link>, the appearance of the link is specified by the <Style> node.

Table 13-5: Internal Attributes

Attribute Name	Definition
configuration	Points to the file that contains the Link Management configuration. Typically, this is the DocConfig file for the document class.
substyle	Required when used as a child of <Multiple>. Specifies the name of the substyle that defines the appearance of the external link.

The <Internal> node can be used as a direct child of <Link> or it can be used as a child of <Multiple> when a combination of internal and external links is desired.

Target

The <Target> element is used to define an element that may be the target of an Internal hyperlink. Each target type is defined by a separate <Target> element; the <Internal> node contains one or more of them.

Table 13-6: Target Attributes

Attribute Name	Definition
definition	The name of the <ElementDef> of the target element. This value is the value of the name attribute, not the xmlname of the element.
link	The name of the <ElementDef> attribute that supplies the target ID.
xpath	Deprecated. Formerly specified the XPath pointer to a document location.

Custom

Use the <Custom> node to indicate that the link creation and behavior will be handled by an external application.

When used as a child of <Multiple>, the substyle attribute is required. The value of the substyle attribute specifies the name of a substyle, defined in the <SubStyles> node of the <EmphasisDef>, that defines the appearance of the external link.

When used as a child of <Link>, the appearance of the link is specified by the <Style> node.

Table 13-7: Custom Attributes

Attribute Name	Definition
assembly	The name of the external assembly that handles the link.
class	The name of the external class that handles the link. The assembly and class must implement the Invision.Xpress.Interfaces.EmphasisInterfaces.ICustomLink interface.
substyle	Required when used as a child of <Multiple>. Specifies the name of the substyle that defines the appearance of the custom link.

Multiple

Use the <Multiple> node to wrap a combination of External, Internal, or Custom nodes.

Entity

This emphasis type is used to tag a content entity in the document. Any content entity that is tagged by this emphasis will maintain a synchronous mapping with the entity node in the document.

Adding this definition to the schema implies that every text XML entity encountered in the parent para type element under which this emphasis is defined will be wrapped by this emphasis element. Without defining this emphasis, any encountered text entities will be converted to their literal values and rendered on the canvas. They will be serialized as literal text. Use the <Entity> element to establish presentation options for content entities. Its child elements define its behavior.

Internal content entities defined within the document may be added, modified or deleted. External content entities (public and private) defined outside of the document may not be added, modified or deleted.

1. Wrapper

This is used for backward compatibility to allow for metadata-based (attributes) entity support. With this approach, an entity can be associated with metadata which will persist in the serialized document. This metadata among other things can be used to control whether an entity can be edited/deleted (using attributeName/attributeValue pairs) from the Modify/Delete children.

Add

Specifies whether an entity can be added through the UI. Set its allow attribute to **true** to allow the entity to be added, **false** to prevent.

Modify

If allow attribute is set to **true**, or an entity reference wrapper element has attributeName equal attributeValue, the instance value may be modified through the UI.

Table 13-8: Modify Attributes

Attribute Name	Required	Definition
allow	no	Boolean. Set to true if the content entity can be modified. Set to false if the content entity may not be modified. Defaults to true .
attributeName	no	Name of the attribute which specifies whether the entity can be modified.
attributeValue	no	Value of the attribute which specifies whether the entity can be modified.

Delete

If an entity reference wrapper element has `attributeName` equal `attributeValue`, the instance value may be deleted through the UI. If `allow` attribute is set to **true**, the entity definition may be deleted, provided there are no entity references.

Table 13-9: Delete Attributes

Attribute Name	Required	Definition
Allow	no	Boolean. Set to true if the content entity can be deleted. Set to false if the content entity may not be deleted. Defaults to true .
attributeName	no	Name of the attribute which specifies whether the entity can be deleted.
attributeValue	no	Value of the attribute which specifies whether the entity can be deleted

1. UserDefined

This is used to support a more generic entity scenario where metadata and wrapper emphasis are not preserved. During serialization, the emphasis element wrapper will be dropped and only the original entity will be written.

Table 13-10: UserDefined Attributes

Attribute Name	Required	Definition
allowAdd	no	Boolean. Specifies whether a new Internal entity definition can be added to the current document. Defaults to true .
allowDelete	no	Boolean. Specifies whether an entity definition can be deleted. An entity definition can be deleted only if there are no entity references within the document. This attribute is irrelevant for external entities. Quark XML Author will not delete external entity definitions. Defaults to true .
allowModify	no	Boolean. Specifies whether an entity definition's value can be modified. This attribute is not relevant for external entities. Quark XML Author will not modify external entity definitions. Defaults to true .

ProcessingInstruction

This emphasis type is used to tag a processing instruction in the document. This emphasis is required to be defined to preserve any existing processing instructions in the document otherwise they will be dropped from document.

If Quark XML Author encounters an XML processing instruction in a document, it will temporarily apply a special emphasis to that processing instruction. This emphasis is read-only to prevent any changes to the processing instruction. The emphasis provides a visual indicator to differentiate a processing instruction from surrounding narrative text. When the document is saved, the special emphasis is removed. This returns the processing instruction to its original state.

In the following sample, a Specialized emphasis definition has been added to wrap processing instructions in as emphasis to display in the UI. The emphasis will be read-only and will not show up in the emphasis combo box.

```
<EmphasisDef name="PI" friendly="Processing Instruction">
  <Specialized>
    <ProcessingInstruction/>
  </Specialized>
  <Style backColor="blue"/>
</EmphasisDef>
```

With the above structure, any existing PIs in the document will be wrapped in the above emphasis . Given an element that looks like the following:

```
<p> Testing para with PIs in it <?show [AQ]?> - more text
here</p>
```

The XOM structure will look like the following:

```
<p> Testing para with PIs in it <PI><?show [AQ]?> </PI>- more
text here</p>
```

On Save/Normalize, the special emphasis is removed.

Style

Style defines how the Emphasis will appear in the Word document, through its attributes, which are described below. All are optional.

Style Attributes

Table 13-11: Style Attributes

Attribute Name	Definition
backColor	Name of the color to be used as background highlighting for the emphasized text in the Word document. A list of valid color names is provided in Appendix A. Alternately, you may specify the RGB components of the color using numbers from 0 to 255, separated by commas.
bold	Set to true if the emphasized text is to be bolded. Defaults to false .

Attribute Name	Definition
fontName	Name of a font that will be used to render the emphasized text in the Word document. The font named must be installed on the user's computer.
fontSize	Font size, in points, to be used to render the emphasized text in the Word document. The value must be an integer no less than 8 nor greater than 72.
foreColor	Name of the color to be used to render the emphasized text in the Word document. A list of valid color names is provided in Appendix A. Alternately, you may specify the RGB components of the color using numbers from 0 to 255, separated by commas.
italic	Set to true if the emphasized text is to be italicized. Defaults to false .
strikethrough	Set to true if the emphasized text is to be rendered with strikethrough formatting. Defaults to false .
subscript	Set to true if the emphasized text is to be rendered in subscript. Defaults to false .
superscript	Set to true if the emphasized text is to be rendered in superscript. Defaults to false .
underline	Set to true if the emphasized text is to be underlined. Defaults to false .

Substyles

An option for emphasis is to use Substyles instead of a single Style. Substyles allow you to assign multiple appearances to a single emphasis style. The user will see each appearance as a separate emphasis, but in the document XML, the same emphasis element will appear, regardless of what the user chooses, and an attribute of the emphasis element will define its appearance. For example, in the example below, the user may choose to assign Bold, Underline, or Italics as an emphasis.

```
<EmphasisDef name="EmphasizedText" friendly="Emphasized Text">
  <Style/>
  <SubStyles key="emphasisType">
    <SubStyle value="b" friendly="Bold">
      <Style bold="true"/>
    </SubStyle>
    <SubStyle value="u" friendly="Underline">
      <Style underline="true"/>
    </SubStyle>
    <SubStyle value="i" friendly="Italics">
      <Style italic="true"/>
    </SubStyle>
  </SubStyles>
</Attributes>
  <AttributeType name="emphasisType"/>
</Attributes>
```

```
</EmphasisDef>
```

Applied to the Strikethrough definition above, the emphasized text would look like this in the document XML:

```
<Paragraph>
  Optional. Set to <EmphasizedText
emphasisType="b">true</EmphasizedText> if the emphasized text is
to be rendered with strikethrough formatting. Defaults to
<EmphasizedText emphasisType="b">false</EmphasizedText>.
</Paragraph>
```

The <Substyles> element has one required attribute: key. The **key** value is the name of the attribute that defines the emphasis substyle. Thus, in the example above, **emphasisType** becomes the attribute for the EmphasizedText element.

Each <Substyle> child element will have a **value** attribute. The value of this attribute is the value that will be used to identify the type of emphasis to be applied to the selected text. The <Substyle> element in turn has a child <Style> element which defines its appearance, in exactly the same manner as the regular <Style> element does.



If the Substyles node is used, the sibling Style node must be an empty node with no data as shown in the example above.

Attributes and Attributes Type

Emphasis elements can have attributes associated with them, just as regular elements can. They are referenced in the Attributes section of the EmphasisDef element. Each attribute to be associated with the emphasis definition is identified with the AttributeType element. The AttributeType element's **name** attribute contains the name of an AttributeDef defined earlier in the schema. AttributeType can use any of the Field Attributes defined in Section 11.6.2.

```
<EmphasisDef name="Org" friendly="Organization">
  <Style foreColor="Lavender"/>
  <Attributes>
    <AttributeType name="href.jwics"/>
    <AttributeType name="href.siprnet"/>
    <AttributeType name="href.stoneghost"/>
    <AttributeType name="href.niprnet"/>
    <AttributeType name="href.domain5"/>
  </Attributes>
</EmphasisDef>
```

Associating Emphasis with Elements

Emphasis definitions can be associated with the Para element definition. The Para element has an optional child element named **Emphasis**. This element contains an **EmphasisType** child node for each emphasis style that can be applied to the text in a Para element. An example is shown below.


```
<ElementDef name="para" friendly="Paragraph" style="Body Text">
  <Section>
    <Para>
      <Emphasis>
        <EmphasisType name="color"/>
        <EmphasisType name="bold"/>
        <EmphasisType name="italic"/>
        <EmphasisType name="underline"/>
        <EmphasisType name="link"/>
        <EmphasisType name="replaceable"/>
        <EmphasisType name="xref"/>
        <EmphasisType name="systemitem"/>
        <EmphasisType name="userinput"/>
        <EmphasisType name="anchor"/>
        <EmphasisType name="glossterm"/>
        <EmphasisType name="index"/>
      </Emphasis>
    </Para>
  </Section>
</ElementDef>
```

Extensibility Methods

With some emphasis styles, you may want to provide a way for users to access a tool such as a metadata wizard via a command in the context menu. An emphasis style may use an extensibility method to provide this functionality.

The syntax for using Extensibility Methods is shown below:

```
<ExtensibilityMethods>
  <ExtensibilityMethod id="method number" friendly="friendly
name" showInComponentContextMenu="true or false"
showInContextMenu="true or false" faceID="#" />
</ExtensibilityMethods>
```

Table 13-12: ExtensibilityMethod Attributes

Attribute Name	Required	Definition
enableXPath	no	<p>Value is a boolean XPath expression which evaluates to a node-set in which false is returned for an empty node-set and true otherwise. When applied to the runtime element, indicates whether to enable the method on the Context Menu. The XPath expression will be evaluated against the current runtime element and thus must be relative to it.</p> <p>Namespaces are not supported in the xpath syntax.</p> <p>For the Ribbon, enableXPath on evaluates against root nodes. Against current node is not supported.</p>

Attribute Name	Required	Definition
faceID	no	Allows you to specify an icon that will be displayed next to the context menu item. The value of faceID is an integer corresponding to the ID number of an icon in the Word template FaceID.dot.
friendly	no	Specifies a name for the context menu command.
id	yes	Specifies the ID of the EI method to be called.
showInComponentContextMenu	no	Defaults to true . When set to false , the command will not appear in the Component context menu.
showInContextMenu	no	Defaults to true . When set to false , the command will not appear in the main context menu, but may still appear in the Component context menu if showInComponentContextMenu is set to true .
showXPath	no	<p>Value is an XPath expression which evaluates to a node-set, and when applied to the current element, indicates whether to include the method on the context menu: if a node-set is returned, the method is included; if an empty node-set is returned, the method is omitted. When applied to the runtime element, indicates whether to include the method on the Context Menu. The XPath expression is applied to the XOM therefore the names are XOM element names. The XPath expression will be evaluated against the current runtime element and thus must be relative to it.</p> <p>Namespaces are not supported in the xpath syntax. For example,</p>

Attribute Name	Required	Definition
		<p>showXPath="self::node()[local-name()='Section']".</p> <p>XML Author does not support dynamic show/hide of Ribbon items, therefore showXPath is not applicable for Ribbon items.</p>

See Section 6 for more information on how to configure Extensibility Interface methods.

Restricting Emphasis Nesting

By default, emphasis defined for a text element cannot be nested. For example, if an element named Paragraph can have Bold, Italic, and Underline emphasis applied, those three emphases cannot be nested unless the schema designer explicitly allows it.

To allow emphasis nesting, add an Emphasis child to the EmphasisDef element. Within the Emphasis element, add an EmphasisType node for each emphasis that you want to allow to be nested. EmphasisType is a leaf node with one attribute: **name**. The value of **name** refers to the Emphasis name.

In the example below, the bold emphasis allows italic and underline emphasis to be nested within it.

```
<EmphasisDef name="bold" friendly="Bold">
  <Style bold="true"/>
  <Emphasis>
    <EmphasisType name="italic"/>
    <EmphasisType name="underline"/>
  </Emphasis>
</EmphasisDef>
```

In the next example, the italic emphasis allows only underline emphasis to be nested within it.

```
<EmphasisDef name="italic" friendly="Italic">
  <Style italic="true"/>
  <Emphasis>
    <EmphasisType name="underline"/>
  </Emphasis>
</EmphasisDef>
```

Thus, an author could apply bold emphasis to a range of text and then apply either italic or underline to it. However, if the author first applied italic emphasis to text, only the underline emphasis could subsequently be applied to that range, because italic does not permit bold to be nested within it. Likewise, the author could apply bold and then italic, but not bold, then italic, and then underline. To allow all three emphasis styles to be applied in any combination, all three would need to be defined to allow the others. The italic EmphasisDef would receive a second EmphasisType element, as shown below:

```
<EmphasisDef name="italic" friendly="Italic">
  <Style italic="true"/>
  <Emphasis>
    <EmphasisType name="underline"/>
    <EmphasisType name="bold"/>
  </Emphasis>
</EmphasisDef>
```

```
</Emphasis>
</EmphasisDef>
```

The underline emphasis would be defined as shown below:

```
<EmphasisDef name="underline" friendly="Underline">
  <Style underline="true"/>
  <Emphasis>
    <EmphasisType name="italic"/>
    <EmphasisType name="bold"/>
  </Emphasis>
</EmphasisDef>
```

Inline Media

The EmphasisDef element has an optional child element named Media. This element contains a MediaType child node for each media type (both reference and embedded) that can be inserted inline to the text in a EmphasisDef element.

Table 13-13: MediaType Attributes

Attribute	Required	Definition
name	yes	Identifies an ElementDef node in the schema for the media element

See section 14.6.2, "Images



In order for images to be rendered properly on the Word canvas, every image element definition in the solution must be configured with following image data-types: imagewidth, imageheight, imagewidthdpi, imageheightdpi. See "Table 11-4: Field Attributes – datatype values".

Supported Image Formats", See "Table 11-4:Field Attributes – datatype values;, Supported Image Formats", for a list of supported image formats.



An ElementDef cannot have an media element as both a child element and an inline element in the same ElementDef.

Inline OLE

The EmphasisDef element has an optional child element named OLE. This element contains a OLEType child node for each OLE type that can be inserted inline to the text in a EmphasisDef element.

Supported element types include:

- MathTypeEquation
- OLEWordDocument

Table 13-14: OLEType Attributes

Attribute	Required	Definition
name	yes	name of the elementDef node containing the OLE definition.

Quark XML Author Structure Element Definition

Each element that makes up the element definition section of the schema is defined below. An example is provided with each to demonstrate proper syntax for the element's use.

The Elements element contains the collection of element definition elements for the schema, and is a direct child of the XpressSchema root element. Elements uses no attributes.

```
<Elements>
  <ElementDef name="title">
    <!-- Element definition data -->
  </ElementDef>
  <!-- Additional ElementDef elements here -->
</Elements>
```

ElementDef

ElementDef defines a single element of the schema. ElementDef names the element and provides basic information about it. Child nodes, discussed later in this section, further define the element.

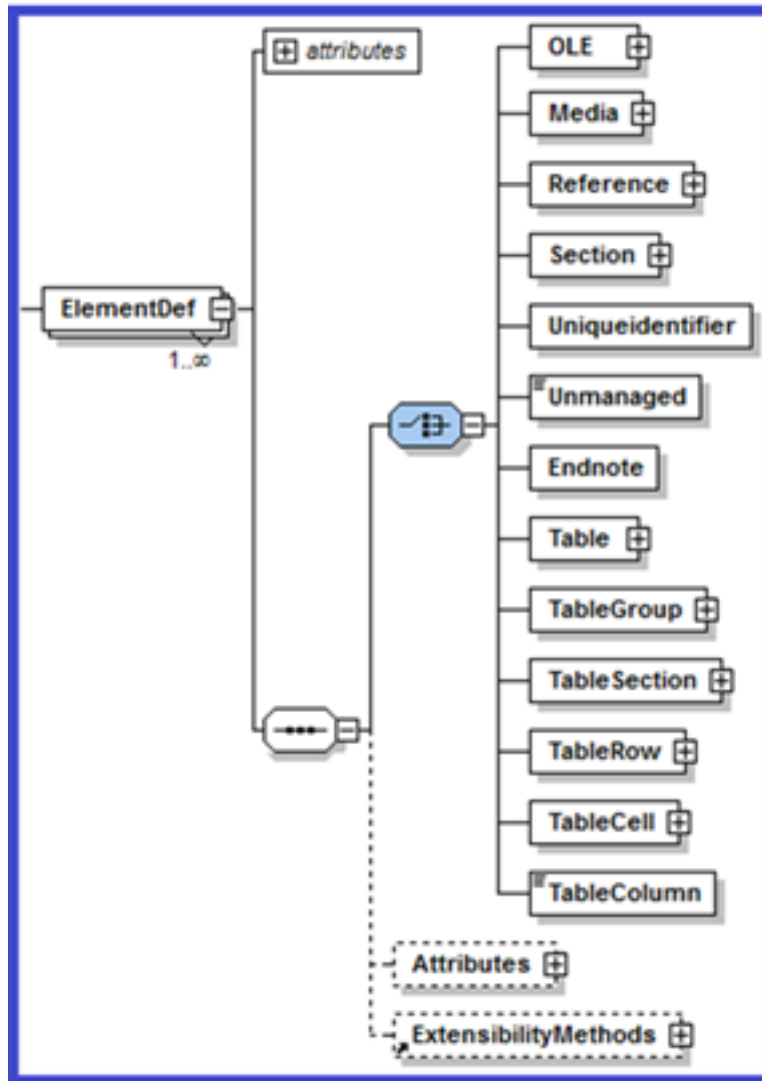


Figure 14-1: Configuration-XSD ElementDef definition

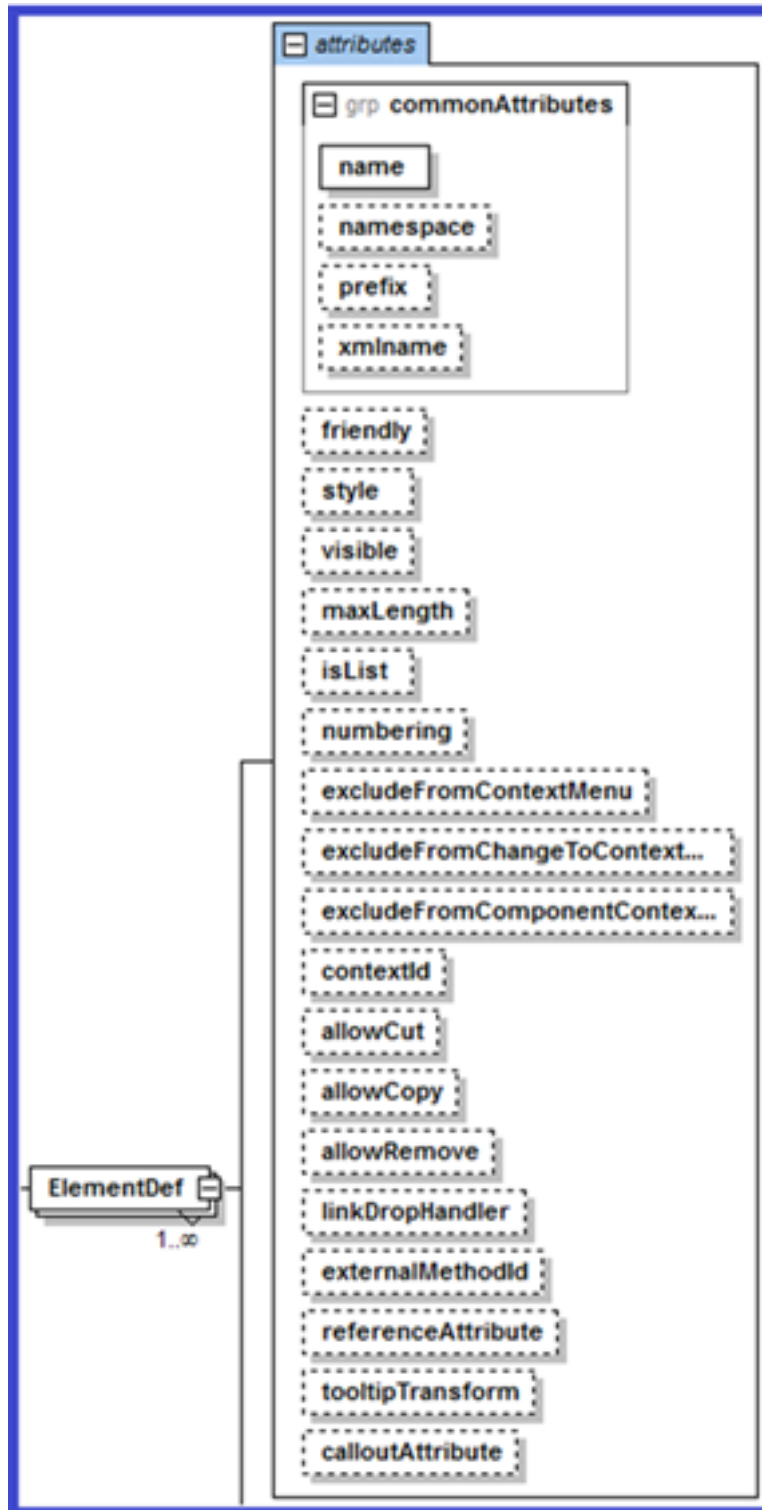


Figure 14-2: Configuration-XSD ElementDef definition with attributes

ElementDef may use any of the attributes defined in the Common Attributes attribute group. It has several additional optional attributes available, each of which is defined below.

Table 14-1: ElementDef Attributes

Attribute	Required	Definition
allowCopy	no	Boolean. Defaults to true . If set to false , the Copy context menu item for the element is disabled.
allowCut	no	Boolean. Defaults to true . If set to false , the Cut context menu item for the element is disabled.
allowRemove	no	Boolean. Defaults to true . If set to false , the Remove context menu item for the element is disabled.
calloutAttribute	no	Used with any numbered style ElementDef. When the document is saved/exported, the calloutAttribute attribute value will contain the list string value of the numbered style element.
contextId	no	Specifies a contextual identifier for the element. If a SectionType within the ElementDef also has a contextId, the SectionType contextId overrides the ElementDef contextId.
excludeFromChangeToContext Menu	no	Boolean. Set to true to remove the Change To menu from this element's context menu and exclude this element from the Change To context menu of other element types. The Change To feature would be unavailable from the user interface. Defaults to false .
excludeFromComponentContextMenu	no	Boolean. Defaults to false . Set to true to exclude the component from the component context menu.
excludeFromContextMenu	no	Boolean. Set to true if the ElementDef defines an element that the user cannot insert directly into the document. This applies to internally generated insertable items, not to items added via externalMethodID.

Attribute	Required	Definition
excludeExternalFriendlyFromContextMenu	no	Boolean. Set to true if the ElementDef defines an element that the user cannot insert directly into the document. This applies to items added via externalMethodID.
externalMethodId	no	<p>The externalMethodId attribute on an ElementDef node lets an EI developer control the actual fragment that gets inserted in Quark XML Author. This is different from the EIs defined under the <ExtensibilityMethods> node which let you perform operations on an existing fragment. If used, this attribute adds an insertable item to the Context Menu. Be sure to set excludeFromContextMenu to true, so that the internally generated insertable item is hidden.</p> <p>The EI method that is called must return an XmlElement. If it returns null or a non-Xml Element the insert will be cancelled as if the user cancelled the Insert operation. If the EI throws an exception, an error dialog will be shown in Quark XML Author notifying the user about Insert failure. The returned XmlElement must match the name of the insertable element which the user selected and must match the structure defined for the element in the XAS. If it does not match, the user will be shown an error dialog and Insert will be cancelled.</p> <p>Note that the method called can use the Tag argument type to pass the actual caption name that the user clicks in</p>

Attribute	Required	Definition
		<p>Quark XML Author. See Tag in section 6.2.1.2 for more information about the Tag argument. This provides a way of providing the caption to the EI, but the EI may not modify the caption.</p> <p>See the case study presented in Chapter 21 for examples of using externalMethodId in this context.</p> <p>This attribute can only be specified for an optional element in the schema.</p>
externalMethodFriendly	no	<p>Specifies the friendly name associated with the external method that will be displayed in the user interface, Content Menu. <ElementDef name="section1" friendly="Section 1" xmlns="topic" visible="false" referenceAttribute="conref" externalMethodId="GetElement" externalMethodFriendly="Section 1 from server"></p> <p>If externalMethodId is specified and externalMethodFriendly is omitted, the label displayed to the user in the insertables Context Menu and Styles control is @friendly or @name with "(custom)" appended.</p>
Friendly	no	Specifies a friendly name for the element being defined.
isList	no	<p>Boolean. Defaults to false. If set to true, this element will be regarded as a list item by the list manager. If set to false, the list manager will ignore this node.</p>

Attribute	Required	Definition
		<p>isList specifies whether or not a numbered list may continue numbering from a previous sibling numbered list. If a configuration contains the following hierarchy:</p> <p>Section</p> <p> List</p> <p> List Item</p> <p>Then a Quark XML Author document may contain the following content:</p> <p>Section A</p> <p> List A</p> <p> List Item 1</p> <p> List Item 2</p> <p>Section B</p> <p> List B</p> <p> List Item 1</p> <p> List Item 2</p> <p>In the configuration, if isList is set to true for Section, then the numbering of List B may continue from List A. List B > List Item 1 may be numbered 3.</p> <p>In the configuration, if isList is set to false for Section and true for List, List B may not continue from List A. List B > List Item 1 may only be numbered 1.</p>
isTable	no	Deprecated in 4.x tables. For pre 4.0 tables, set to true if the ElementDef defines a table

Attribute	Required	Definition
		element. Defaults to false . See “Table” for more information about creating tables.
linkDropHandler	no	Specifies the ID of the EI method to be called when the user drops a hyperlink onto the element. See “Drag and Drop”. Overrides any other internal XML Author behavior.
maxLength	no	Specifies the maximum amount of visible characters that can be added to the element and its descendant elements. The amount is a cumulative total, not the amount allowed in each element.
numbering	no	Valid values are: continue and restart . If set to continue , numbering will continue from previous numbered element, ignoring internal logic for numbering continuance or restart. Defaults to restart .
referenceAttribute	no	Used when an element can be the target of a cross reference or, in DITA, a conref. The value holds the name of the attribute whose value is required for the resolution of the element. In DITA’s case that attribute will be conref.
style	no	Specifies the name of a Word style that will be used to render the element in the authoring environment.
tooltipTransform	no	Defines the path to an XSLT file used to define the text displayed when the user hovers over the element.
visible	no	Boolean. Set to false if the element will not be visible to the user within the authoring environment. Defaults to false .
allowPortionmark	No	Boolean. If set to true, sets the visibility of Portionmarks for

Attribute	Required	Definition
		an element type. Defaults to False

Section

Section represents a collection of multiple elements that form a coherent subset of the document, such as a chapter in a book or a subtopic in an article. Section may contain a Para element and/or one or more Sequence or Choice elements, which in turn contain the document elements that comprise the document section.

- See Para for more information.
- See Sequence for more information.
- See Choice for more information.

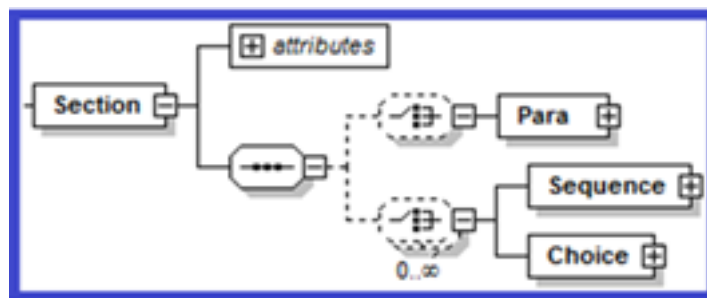


Figure 14-3: Configuration-XSD ElementDef > Section definition

Table 14-2: Section Attributes

Attribute	Required	Definition
displayAttribute	no	String. Allows you to specify that instead of the element name, the value of the attribute named will be displayed to the user.
displayElement	no	Boolean. If set to true , the display name of the element's Component menu will be the name of the element's first child element. This attribute is used to facilitate the following use case. The element C has two child elements, A and B. This attribute allows the user to right-click on A or B and access the Component menu for C, which is the "insertable"

Attribute	Required	Definition
		parent of A and B.

Para

Para represents a text element, such as a paragraph, caption, or title. It may have an Emphasis child element to indicate that emphasis styles may be applied to the text it contains. Para uses these optional attributes: default, readonly, childTextNode and defaultHyperlink.

Table 14-3: Para Attributes

Attribute	Required	Definition
childTextnode	no	Set to true to allow combining multiple child text nodes specified under the para node into the parent node. Does not support the insertion of soft line breaks. Pressing Shift-Enter key acts the same as pressing Enter key. Can be used in conjunction with insertAfter . See “Table 14-133: SectionType Attributes” for more information.
default	no	Default text for the para element.
defaultHyperlink	No	Specifies the target hyperlink emphasis that should be applied when a text type link is dragged onto this para type element. If defaultHyperlink is not specified, text links are ignored. For a given dropped link on a para type element, precedence is resolution of an inline element resolution and then evaluation for hyperlink emphasis. Formats are categorized as definitions with and without SubStyles defined: Without SubStyles defined: {emphasisName} Specify the name of an existing emphasis definition. This format is used

Attribute	Required	Definition
		<p>when the target emphasis definition has Style defined, but no SubStyles defined.</p> <p>With SubStyles defined:</p> <p>{emphasisName}@{attributeName}={attributeValue}.</p> <p>For example, defaultHyperlink="xrefemph@format='html'" or defaultHyperlink="xrefemph@outputclass='inter'"</p> <p>The string following the @ identifies which sub-style to apply on the canvas.</p> <p>For example,</p> <pre> <EmphasisDef name="xrefemph" friendly="Cross Reference" xmlns="xref" excludeFromEmphasisCombo="true"> <Style/> <SubStyles key="format"> <SubStyle value="dita" friendly="Cross- Reference"> <Style foreColor="blue" underline="true"/> </SubStyle> <SubStyle value="html" friendly="Hyperlink"> <Style foreColor="blue" underline="true"/> </SubStyle> </pre>

Attribute	Required	Definition
		<pre> </SubStyles> <Attributes> .. <AttributeType name="outputclass"/> </Attributes> <Emphasis/> </EmphasisDef> or <EmphasisDef name="xrefemph" friendly="Cross Reference" xmlns="xref" excludeFromEmphasisCombo ="true" readonly="true"> .. <Style/> <SubStyles key="outputclass"> <SubStyle value="intra" friendly="Cross Reference"> <Style underline="true" foreColor="Blue"/> </SubStyle> <SubStyle value="inter" friendly="Hyperlink"> <Style underline="true" foreColor="Blue" italic="true"/> </SubStyle> </pre>

Attribute	Required	Definition
		<pre> </SubStyles> <Attributes> .. <AttributeType name="outputclass"/> </Attributes> <Emphasis/> </EmphasisDef> </pre>
readonly	no	Set to true to lock the para element upon insertion so that the user cannot edit the default text.

Qua childTextnode Serialization

This section discusses the serialization of childTextnodes elements. Given the following sample configuration fragment.

```

<ElementDef name="p">
    <Section>
        <Para/>
        <Choice minOccurs="0"
maxOccurs="unbounded">
            <SectionType name="..."/>
            <SectionType name="my-child-
textnode"/>
        </Choice>
    </Section>
</ElementDef>

```

When a user inserts my-child-textnode, an element is added to the XOM just like any other element. However, the element is temporary and exists only while the document is open in Quark XML Author. The childTextnode element only exists to hold a text fragment for its parent para-type (<p>). When serialized, the childTextnodes wrapper is dropped, leaving its text fragment within the parent element.

For example, the following XOM:

```

<p>first text fragment
    <my-child-textnode>second text fragment</my-child-textnode>
    <my-child-textnode>third text fragment</my-child-textnode>
</p>

```

is serialized as:

```

<p>first text fragment
second text fragment
third text fragment
</p>

```

Let's look at a mixed-content example.

The following content on the canvas inside a cell (an element named “entry”):

————(Empty)

<CellChildTextNode> first child text node</CellChildTextNode>

————(Empty)

- One

————(Empty)

- Two

————(Empty)

<CellChildTextNode> second child text node</CellChildTextNode>

———— (Empty)

<CellChildTextNode> third child text node</CellChildTextNode>

is serialized as:

<entry> first child text node

<bulletul><bulletuli>one</bulletuli><bulletuli>two</bulletuli></bulletul> second
child text node\n third child text node</entry>

During Serialization, adjacent childTextnode elements are serialized as a single textnode fragment, with each element delimited in XML with a line break.

Deserialization inserts a childTextnode element for each line break for each textnode fragment. When the document is re-opened, the content is displayed on the canvas the same as it did before the document was closed.

childTextnode

The following discusses the user experience and developer’s perspective.

From a user’s perspective, right-clicking a childTextnode provides access to Attributes and EI methods associated with the childTextnode’s parent element.

From an EI developer’s perspective, with the current selection on a child textnode, the XomCurrentNode argument references an element in the XOM associated with the childTextnode, distinct from its parent element.

For more information see, “**Error! Reference source not found.**”

Emphasis is not ‘inherited’ from the parent. The emphasis defined for the child text node must match the parent. Multiple parent elements could contain the childtextnode Child so this is important.

Emphasis

Emphasis represents an inline element. The Para element has an optional child element named Emphasis. The Emphasis element contains an EmphasisType child node for each emphasis style that can be applied to the text in a Para element.

Table 14-4: Emphasis Attributes

Attribute	Required	Definition
allowMedia	no	Specifies whether media is allowed under Emphasis. If set to yes , allows media elements under any emphasis. If set to no , does not allow media under any emphasis. If set to restricted , the allow/disallow behavior is derived from the individual emphasis definitions under the EmphasisDef/Media node.
allowOLE	no	Specifies whether OLE elements are allowed under Emphasis. If set to yes , allows OLE elements under any emphasis. If set to no , does not allow OLE elements under any emphasis. If set to restricted , the allow/disallow behavior is derived from the individual emphasis definitions under the EmphasisDef/OLE node.

Table 14-5: EmphasisType Attributes

Attribute	Required	Definition
name	yes	Identifies an EmphasisDef node in the schema.

Inline Media

The Para element has an optional child element named Media. This element contains a MediaType child node for each media type that can be inserted inline to the text in a Para element.

Table 14-6: MediaType Attributes

Attribute	Required	Definition
name	yes	Identifies an ElementDef node in the schema.

See “Images” for a list of supported image formats.

Inline OLE

The Para element has an optional child element named OLE. This element contains a OLEType child node for each OLE type that can be inserted inline to the text in a Para element.

Supported element types include:

- MathTypeEquation

Table 14-7: OLEType Attributes

Attribute	Required	Definition
name	yes	The name of the elementDef containing the OLE definition.

Sequence

A Sequence element indicates that the elements it contains can only be inserted into a document in the order in which they are listed. Sequence may contain any of the following child elements: OLEType, MediaType, TableType, ReferenceType, SectionType, UniqueidentifierType, UnmanagedType and EndnoteType.

These elements are defined later in this chapter.

- See “OLEType” for more information.
- See “TableType” for more information.

See “Table 14-10: TableType Attributes

Attribute	Required	Definition
Name	yes	Specifies the name by which the element will be identified.
insertAfter	no	Allows you to add new body text after a table when the <Enter> key is used at the top or at the end of a table.

- See “MediaType” for more information.
- See “ReferenceType” for more information.
- See “SectionType” for more information.
- See “UniqueidentifierType” for more information.
- See “UnmanagedType” for more information.
- See “EndNoteType” for more information.

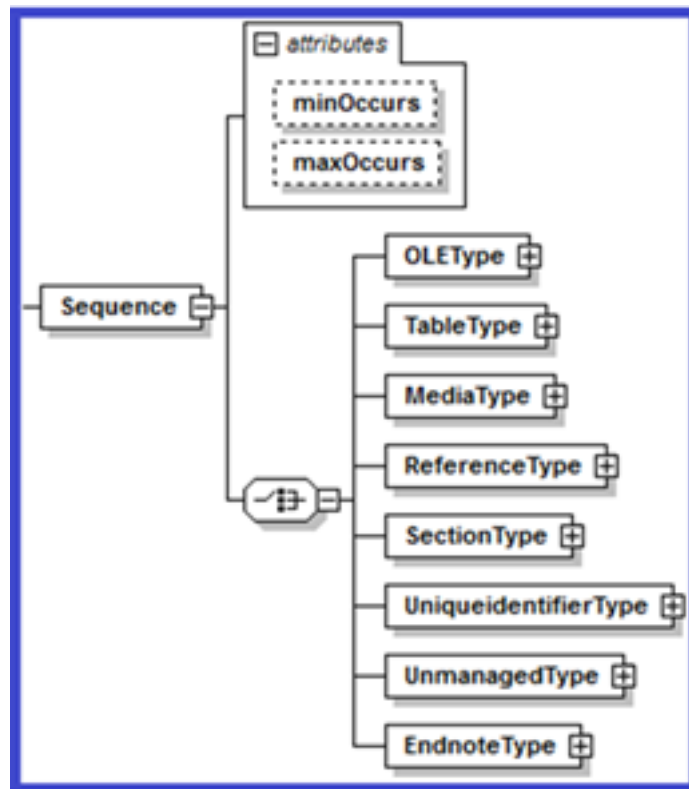


Figure 14-4: Configuration-XSD Section > Sequence definition

Table 14-8: Sequence Attributes

Attribute	Required	Definition
maxOccurs	no	Indicates the maximum number of times the element can be added to the section. Valid values are: 1 and UNBOUNDED. The latter indicates that there is no limit to the number of times the element may be added. Defaults to 1.
minOccurs	no	Indicates the minimum number of times the element can be added to the section. Valid values are 0 and 1. Defaults to 1.

Choice

A choice element indicates that the elements it contains can be inserted into a document in any order. Choice may contain any of the following child elements: OLEType, MediaType, TableType, ReferenceType, SectionType, UniqueidentifierType, UnmanagedType and EndnoteType.

These elements are defined later in this chapter.

QUARK XML AUTHOR STRUCTURE ELEMENT DEFINITION

- See “OLEType” for more information.
- See “TableType” for more information.

See “Table 14-10: TableType Attributes

Attribute	Required	Definition
Name	yes	Specifies the name by which the element will be identified.
insertAfter	no	Allows you to add new body text after a table when the <Enter> key is used at the top or at the end of a table.

- See “MediaType” for more information.
- See “ReferenceType” for more information.
- See “SectionType” for more information.
- See “UniqueidentifierType” for more information.
- See “UnmanagedType” for more information.
- See “EndNoteType” for more information.

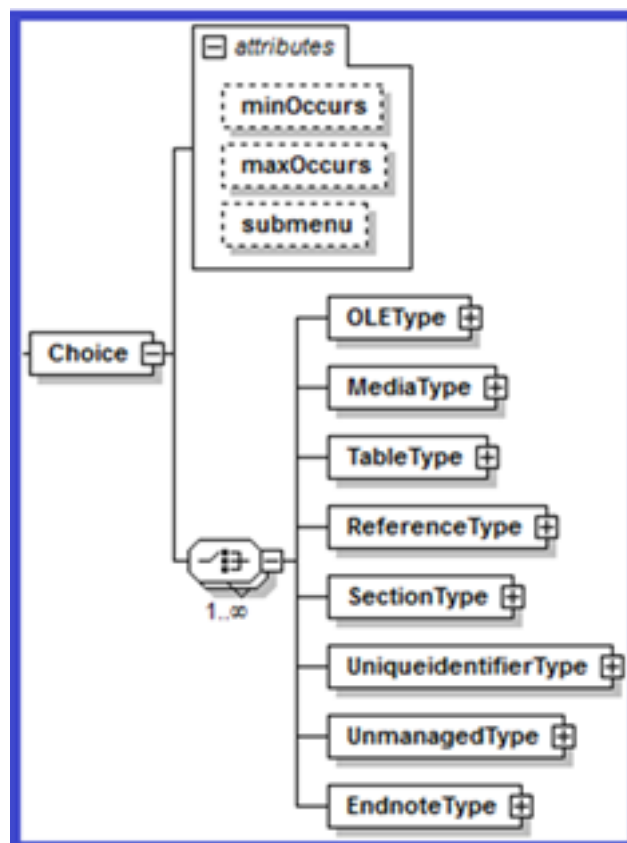


Figure 14-5: Configuration-XSD Section > Choice definition

The number of required Choice elements that can be specified for a given section is unlimited. Required elements can have required Choice elements and this can cascade to an arbitrary depth. Likewise, required Choice elements can

themselves have required Choice elements and this can cascade to an arbitrary depth.

Table 14-9: Choice Attributes

Attribute	Required	Definition
maxOccurs	no	Indicates the maximum number of times the element can be added to the section. Valid values are: 1 and UNBOUNDED. The latter indicates that there is no limit to the number of times the element may be added. Defaults to 1.
minOccurs	no	Indicates the minimum number of times the element can be added to the section. Valid values are 0 and 1. Defaults to 1.
submenu	no	Boolean. Used only with the Choice element. If set to false , required child elements will appear beneath the element in the context menu, instead of in a submenu. Defaults to true .

OLEType

Indicates that a type of OLE element can be inserted into the Sequence or Choice. It uses only one attribute: **name**, which is required, and specifies the name by which the element will be identified.

See “OLE” for more information.

TableType

Indicates that a type of Table can be inserted into the Sequence or Choice.

See “Table” for more information.

Example:

```
<TableType name="CalsTable" insertAfter="p"/>
<TableType name="simpletable" insertAfter="p"/>
<TableType name="HtmlTable" insertAfter="p"/>
```

Table 14-10: TableType Attributes

Attribute	Required	Definition
Name	yes	Specifies the name by which the element will be identified.
insertAfter	no	Allows you to add new body text after a table when the <Enter> key is used at the top or at the end of a table.

MediaType

MediaType indicates that a media element can be inserted into the Sequence or Choice within the Section.

See “Media” for more information.

Table 14-111: MediaType Attributes

Attribute	Required	Definition
friendly	no	Specifies a friendly name for the element.
name	yes	Specifies the name by which the element will be identified.
style	no	Specifies a Word style name that will be used to render the element’s text in the user interface.

ReferenceType

ReferenceType indicates that a reference element can be inserted into the Sequence or Choice within the Section.

See “Reference” for more information.

Table 14-122: ReferenceType Attributes

Attribute	Required	Definition
insertAfter	no	Specifies the element to be inserted after the current element if the user hits the Enter key.
insertAfterBlank	no	Used with list items only. Specifies the element to be inserted after the parent list element if the user presses the Enter key and the current list item has no text.
friendly	no	Specifies a friendly name for the element. This is used to

Attribute	Required	Definition
		override the friendly specified in the definition.
name	yes	Specifies the name by which the element will be identified.
style	no	Specifies a Word style name that will be used to render the element's text in the user interface.

SectionType

SectionType indicates that a Section element can be inserted into the Sequence or Choice within another Section.

See “Section” for more information.

SectionType may use any of the attributes defined within the definitionAttributes attribute group. It also requires the use of the **name** attribute, which specifies the name by which the element will be identified.

Table 14-133: SectionType Attributes

Attribute	Required	Definition
contextId	no	Specifies a contextual identifier for the reference. If the parent element has a contextId, the SectionType contextId overrides the parent.
default	no	Default text for the element. Note that certain characters (forward slash, left and right square braces, left and right curly braces, star, and tilde) are reserved by the system and must be represented by codes. Codes are given in Table 5-18 on page 82.
decreaseIndent	no	Value indicates a different element to which the present element will be changed when the user presses Shift + Tab at the beginning of the element.
friendly	no	Specifies a friendly name for the element being defined.
increaseIndent	no	Value indicates a different element to which the present element will be changed when the user presses the Tab key at

Attribute	Required	Definition
		the beginning of the element
insertAfter	no	Specifies the element to be inserted after the current element if the user presses the Enter key. When the user types in an Empty, the system uses the insertAfter setting of the previous element.
insertAfterBlank	no	Used with list items only. Specifies the element to be inserted after the parent list element if the user presses the Enter key and the current list item has no text.
name	yes	Specifies the name by which the element will be identified.
numbering	no	If set to continue , numbering will continue from previous numbered element, ignoring internal logic for numbering continuance or restart. Defaults to restart . Valid values are continue and restart .
readOnly	no	Boolean. Set to true to lock the para element upon insertion so that the user cannot edit the default text.
style	no	Specifies the name of a Word style that will be used to render the element in the authoring environment.
visible	no	Boolean. Set to false if the element will not be visible to the user within the authoring environment. Defaults to true .
allowPortionmark	No	Boolean. If set to true, sets the visibility of Portionmarks for a section type. Defaults to false .

Multiple Increase/Decrease Indent values

The increaseIndent and decreaseIndent attributes can also specify multiple descendant/ancestor values separated by a forward slash (/). Consider the following example:

```
<ElementDef name="ol">
  <Section>
    <Sequence maxOccurs="unbounded">
      <SectionType name="oli" increaseIndent="suboli/subuli"/>
    </Sequence>
  </Section>
</ElementDef>
```

```

    </Sequence>
  </Section>
</ElementDef>

```

With the following XOM structure:

```

<ol>
  <oli> ordered list item 1
    <subul>
      <subuli>Unordered sub list item</subuli>
    </subul>
  </oli>
  <oli>ordered list item 2 </oli>
</ol>

```

Tabbing on the last oli element (shown highlighted above) will convert it to a subuli item under the previous oli sibling. The resulting structure will become:

```

<ol>
  <oli>
    ordered list item 1
    <subul>
      <subuli>Unordered sub list item</subuli>
      <subuli>ordered list item 2 </subuli>
    </subul>
  </oli>
</ol>

```

This is because in the SectionType for **oli**, we have specified both **suboli** and **subuli** as possible targets. This example assumes, of course, that the ElementDef for **oli** allows the necessary child elements that themselves contain **subuli** and **suboli** children.

If **subuli** was not specified (i.e. increaseIndent="suboli"), the resulting structure would be:

```

<ol>
  <oli>
    ordered list item 1
    <subul>
      <subuli>Unordered sub list item</subuli>
    </subul>
    <subol>
      <subuli>ordered list item 2 </subuli>
    </subol>
  </oli>
</ol>

```

For example, a new **subol** gets added under **subul** if allowed by the schema.

UniqueidentifierType

UniqueidentifierType indicates that a Uniqueidentifier element can be inserted into the Sequence or Choice within a Section. It uses only one attribute: **name**, which is required, and specifies the name by which the element will be identified. See "Uniqueide

UnmanagedType

UnmanagedType indicates that an Unmanaged element can be inserted into the Sequence or Choice within a Section. It uses only one attribute: **name**, which is required, and specifies the name by which the element will be identified.

See “Unmanaged” for more information.

EndNoteType

Indicates a type of EndNote that can be stored in the Sequence or Choice. The value for minOccurs on the parent Sequence or Choice is always 0 and the value for maxOccurs on the parent is always 1 for the EndnoteType element, so these values do not need to be specified.

See “EndNote” for more information.

Uniqueidentifier

A Uniqueidentifier element auto-assigns a GUID to the element’s text node. As with other ElementDefs, a Uniqueidentifier may be visible or hidden. When a document is opened, all null Uniqueidentifier elements are auto-assigned. Likewise, when a Uniqueidentifier element is inserted, it is auto-assigned.

See “Uniqueidentifier” for more information.

Unmanaged

An Unmanaged element is one that is inserted into the XOM, but is not used or represented to the user by Quark XML Author. It allows elements to be inserted into the document via and for the use of external applications.

See “UnmanagedTypeUniqueidentifierType” for more information.

OLE

An OLE element allows the insertion of OLE objects as a child of a Section > Sequence and Section > Choice.

See “OLEType” for more information.

It has one attribute, **readonly**, described in Table 14-14.

Table 14-144: OLE Attributes

Attribute Name	Required	Definition
readonly	no	If set to true , the OLE object may not be edited. Defaults to false .

An OLE element may contain a MathTypeEquation or a OLEWordDocument as a child element. MathTypeEquation attributes are described in Table 14-15. OLEWordDocument attributes are described in Table 23-1.

Read-only scenarios

In regards to the **readonly** attribute, MathTypeEquation and OLEWordDocument are treated differently. In XML Author, if the user double-clicks an

MathTypeEquation, the system does not launch the equation editor. If the user double-clicks an OLEWordDocument, the system launches a new Word window that contains the embedded document. This allows users to view the entire Word document. The embedded document is also treated this way if the XML Author document is read-only. For more information, see “SetAccessMode”, “Starter Document” and “IsEditableElement”.

In these read-only scenarios, the new Word window that the system launches is the opened in protected mode with a password. This prevents the user from editing the Word document. Using the document protection feature of Word in this way allows XML Author to react to changes made to the read-only status of the parent XML Author document while it is open. If the embedded document is already open in separate Word window, and the status of the parent XML Author document becomes read-only, then changes in the embedded document are not synched

MathTypeEquation

For more information on implementing math equation support, see “MathType Integration”.

Table 14-155: MathTypeEquation Attributes

Attribute Name	Required	Definition
exportFormat	no	Rendition file type. Supported image formats include:WMF and GIF. Defaults to GIF.
saveMTEF	no	<p>Deprecated in 4.5.1. XML Author will open documents that contain data in this format and convert the data to OLE Object format. This occurs transparent to the user. The change was required to reliably preserve and render MathType equation dimensions.</p> <p>If set to true, the math equation data is saved as MTEF (lossless), MathML and image formats, otherwise the data is saved as MathML and image formats. Defaults to true.</p>
saveOLEObject	no	<p>If set to true, the math equation data is saved as binary OLE object, MathML and image formats, otherwise the data is saved as MathML and image formats. Defaults to true. Must be used instead of saveMTEF to reliably preserve and render MathType equation</p>

Attribute Name	Required	Definition
		dimensions. This must be set to true , because Mathtype 6.0 doesn't have the ability to import MathML.
includeMathTypeData	no	If set to true , annotation data is saved. Defaults to true .
includeTranslator	no	If set to true , translator comments are saved. Defaults to true .
translator	no	Specifies a .TDL translator file which is used to control the format of exported MathML. Supports MathType and custom translator files. Defaults to "MathML2 (no namespaces).tdl". Be aware that some translators do not export MathML, but instead act as inputs to other translators. Contact Design Science technical support for details.

An example is shown below:

```
<OLE readonly="false">
  <MathTypeEquation exportFormat="GIF" saveOLEObject="true"
    translator="MathML2 (no namespace).tdl" includeTranslator="true"
    includeMathTypeData="true"/>
</OLE>
```

For information on inline OLE see section 13.1.2 and section 14.2.1.5.

OLEWordDocument

Allows the user to embed a fully functional Word document within an XML Author document. The user can edit the document without restrictions, but the intended use of this feature is to support embedded large/complex tables.

For details implementing embedded Word document support, see "OLE Word Document Integration".

Media

There are two types of Media elements available: one which defines a parent reference element to allow a media object from a repository to be referenced in the document, and one which allows image media to be inserted directly into the document.

See “Table 14-10: TableType Attributes

Attribute	Required	Definition
Name	yes	Specifies the name by which the element will be identified.
insertAfter	no	Allows you to add new body text after a table when the <Enter> key is used at the top or at the end of a table.

MediaType



An ElementDef cannot have a media element as both a child element and an “Inline Media” in the same ElementDef.

Media as Local Object

When media are to be inserted as a local object, the Media element is used as the child of an ElementDef element. It is an empty element, and serves only to notify the Quark XML Author add-in that its content is to be Base64 encoded.

Table 14-166: Local Media Attributes

Attribute	Required	Definition
maxFileSize	no	Specifies the maximum file size in megabytes. Defaults to 0, which indicates no maximum file size.

Images



In order for images to be rendered properly on the Word canvas, every image element definition in the solution must be configured with following image data-types: imagewidth, imageheight, imagewidthdpi, imageheightdpi. See “Table 11-4: Field Attributes – datatype values”.

Supported Image Formats

Quark XML Author supports the following image file formats:

BMP, DIB, EMF, EMZ, GIF, JFIF, JPEG, JPG, JPE, PNG, RLE, TIFF, TIF, WMZ, and WMF.

EPS images are not longer supported in Office. For more information from Microsoft see, “ [Support for EPS images has been turned off in Office](#)”.

Image Validation

Image Type

When the user attempts to insert an image in a document, the system performs a check to ensure that the image data format matches the format associated with the file extension. If the binary signature in the image header is the correct value for that image file type, then the system inserts the image. If not, the system displays the following message to the user. “{0}” is replaced with the file extension:

“Image data does not match the format associated with the file extension: {0}.”

XML Author processes images based on the file type indicated by the file extension. Occasionally, Quark has encountered images where the image data format and the file extension do not match. These bad images could not be processed and occasionally resulted in corrupt XML Author documents. This check was added to prevent that situation.

Image Metadata

In order for images to be rendered properly on the Word canvas, the following image metadata must be known: image width, image height, image width dpi, image height dpi. These values are needed in order to needed in order for the system to properly calculate the area or “box” on the Word canvas in which the image is rendered. For this discussion, the values “exist” if the attributes are present for the XML image element. Before an image is rendered these values are analyzed.

If the image height and width exist and the image height dpi and image width dpi exist and match the values in the image binary data, then the system has all necessary information to calculate the area and render the image.

However, it is possible that some of these values may be missing in the image element or different from the values in the image binary data. In this situation, the following logic applies.

If either image width or image height does exist, then retrieve image width, image height, image width dpi and image height dpi from the image binary data and store them in the image element.

If image width and image height exist, but either image width dpi or image height dpi does not exist, then retrieve image width dpi and image height dpi from the image binary data and store them in the image element.

If image width, image height, image width dpi and image height dpi exist, but either image width dpi or image height dpi is different from the same value in the image binary data, then retrieve image width dpi and image height dpi from the image binary data and use those dpi values to recalculate the image height and width and store these values in the image element.

Reference

Reference defines an element that is used to insert external content into a document, such as a media item or reusable text. It contains one or more ComponentType child elements, or one Media child element, and has no attributes.

Table 14-177: Reference Attributes

Attribute	Required	Definition
altAttribute		Specifies any attribute whose value will act as a tooltip for a referenced image.

See “ReferenceType” for more information.

```
<ElementDef name="media" friendly="Media Reference"
style="Reference">
  <Reference>
    <Media xmlns="mediaobjectDef" friendly="Media">
      <Mimetypes>
        <!-- Mimetype elements -->
      </Mimetypes>
      <Attributes>
        <!-- AttributeType elements -->
      </Attributes>
    </Media>
  </Reference>
</Attributes>
  <!-- AttributeType elements -->
</Attributes>
</ElementDef>
```

Media

As a child of the Reference element, Media may have two child elements: Attributes, which identifies attributes that will be applied to the element in the schema, and Mimetypes. The Mimetypes element is further described below.

Table 14-188: Referenced Media Attributes

Attribute	Required	Definition
excludeFromComponentContextMenu	no	Boolean. Defaults to false . Set to true to exclude the component from the component context menu.
friendly	no	Specifies a user-friendly name that will identify the element to the user.
maxFileSize	no	Specifies the maximum file size in megabytes. Defaults to 0, which indicates no maximum file size.
required	no	Allows the media object to be deleted on its own

Attribute	Required	Definition
		(mediaObjectDef in the above definition) without deleting the parent media element (media in the above definition). In other words, setting required=false implies mediaObjectDef minOccurs=0 maxOccurs=1 (i.e optional). If not specified, the required attribute defaults to true. If false, the image can be deleted without deleting the image container. With required=false, the excludeFromContextMenu for the container element (media ElementDef above) must be true.
style	no	Specifies a Word paragraph style that will be used to display the element title in the Word document.
xmlname	yes	Specifies the name by which the element is identified in the XML document.

Mimetypes and MimeType

The MimeType element contains the collection of MimeType elements that define what types of media can be inserted into the media element. It uses no attributes.

Mimetype has one attribute: **type**. Type identifies a media type and file format that may be inserted into the media element, as shown in the Syntax Sample below.

```

<Media xmlname="mediaobjectDef" friendly="Media">
  <Mimetypes>
    <MimeType type="image/bmp"/>
    <MimeType type="image/gif"/>
    <MimeType type="image/jpeg"/>
    <MimeType type="image/jpg"/>
    <MimeType type="image/png"/>
    <MimeType type="image/tif"/>
    <MimeType type="image/tiff"/>
  </Mimetypes>
  <Attributes>
    <!-- AttributeType elements -->
  </Attributes>
</Media>

```

For more information on supporting images, see “Images”.

ComponentType

As a child of the Reference element, ComponentType defines a non-media item that may be inserted into the document as a reference. It uses only one attribute: name, which specifies the type of content that can be referenced.

```
<ElementDef name="AllRef" friendly="Reference" style="Reference">
  <Reference>
    <ComponentType name="para"/>
    <ComponentType name="illust"/>
    <ComponentType name="list"/>
    <ComponentType name="table"/>
    <ComponentType name="Slide"/>
  </Reference>
  <Attributes>
    <AttributeType name="element.number"/>
  </Attributes>
</ElementDef>
```

EndNote

The Endnote element supports both traditional end notes as well as notes tied to specific elements. Typically, this capability is used to create table notes, but it would be possible to have section notes, paragraph notes, and so on. Endnote is a leaf of ElementDef, and serves as a holding area for all endnote emphasis element instances.

See “EndNoteType” for more information.

The structure of the ElementDef is as follows:

```
<ElementDef name="Endnote" visible="false">
  <Endnote/>
</ElementDef>
```

Endnote is used in conjunction with the EndnoteType element (a child of Sequence or Choice) and must also have a supporting emphasisDef element.

The following discussion demonstrates how these various components integrate, and builds upon the ElementDef example given above.

Define Endnote Emphasis

The Endnote element supports both traditional end notes as well as notes tied to specific elements. Typically, this capability is used to create table notes, but it would be possible to have section notes, paragraph notes, and so on. Endnote is a leaf of ElementDef, and serves as a holding area for all endnote emphasis element instances.

See “EndNoteType” for more information.

The structure of the ElementDef is as follows:

```
<ElementDef name="Endnote" visible="false">
  <Endnote/>
</ElementDef>
```

Endnote is used in conjunction with the EndnoteType element (a child of Sequence or Choice) and must also have a supporting emphasisDef element.

The following discussion demonstrates how these various components integrate, and builds upon the ElementDef example given above.

Using the EndnoteType Element

The node that will store the collection of Endnotes is identified by using the EndnoteType element as a child of a Sequence node somewhere in the schema. Typically, this node is located in the document root declaration element, as in the highlighted nodeset in the following example:

```
<ElementDef name="dita" friendly="Document" visible="false"
excludeFromComponentContextMenu="true">
  <Section>
    <Sequence minOccurs="0" maxOccurs="1">
      <!-- This node contains information for processing Entity
emphasis tagging -->
      <UnmanagedType name="formdata"/>
    </Sequence>
    <Sequence minOccurs="0" maxOccurs="unbounded">
      <SectionType name="topic" friendly="Heading 1"/>
    </Sequence>
    <Sequence>
      <!-- minOccurs=0 and maxOccurs=1 always for a EndnoteType
element -->
      <!-- Furthermore there can only be one reference of a
particular EndNoteType element in the entire schema-->
      <EndnoteType name="Endnote"/>
    </Sequence>
  </Section>
</ElementDef>
```

As stated in the comment line of the example above, minOccurs is always 0 and maxOccurs is always 1 for the EndnoteType element, so these values do not need to be specified. Also, a particular Endnote definition can only be referenced once.

Note that if the schema contained a second Endnote definition for use with table notes (so that a document could have endnotes and table notes), a second EndnoteType element would be needed to refer to the second Endnote definition.

InsertEndNote InternalClass

The final component needed is the InsertEndNote internal class, which is discussed in section 5.34. The example given below references the EmphasisDef defined as an example in section 14.8.1, above.

```
<InternalClass name="InsertEndNote" emphasisName="en"/>
```

If table notes were implemented in the same document class, a second InternalClass would need to be defined in the Extensibility Interface to refer to the table note emphasis.

Table

See “TableType” for more information.

Prerequisites: You should be familiar with the following sections before proceeding:

```
<Elements>
  <ElementDef name="title">
    <!-- Element definition data -->
```

```
</ElementDef>
<!-- Additional ElementDef elements here -->
</Elements>
```

- ElementDef
- Section
- Para
- Sequence

Quark XML Author be configured to support tables with and without sections and tables with and without groups.

See “TableType” for more information.

Prerequisites: You should be familiar with the following sections before proceeding:

```
<Elements>
  <ElementDef name="title">
    <!-- Element definition data -->
  </ElementDef>
  <!-- Additional ElementDef elements here -->
</Elements>
```

- ElementDef
- Section
- Para
- Sequence

Quark XML Author be configured to support tables with and without sections and tables with and without groups.

CALS Exchange Table

OASIS identified a subset of the full CALS table model that had a high probability of successful interoperability among the OASIS vendor products. This subset is the CALS Exchange Table Model.

<http://www.oasis-open.org/specs/tr9503.html>

The table viewed using the View XML feature.

```
<table id="id-14ed52d0-b4c8-42c3-99a8-3eb92c7fd4f" frame="all" tabledef="CalsTable">
  <tgroup cols="3" id="id-3b475fa9-c997-4b6d-80bb-684358b3d093" textType="default" align="left" valign="top">
    <colspec colsep="1" colwidth="152.90*" align="left" valign="top"/>
    <colspec colsep="1" colwidth="152.90*" align="left" valign="top"/>
    <colspec colsep="1" colwidth="152.90*" align="left" valign="top"/>
  </tgroup>
  <thead id="id-cac31a99-adc4-4270-a8b8-4ab801ff2d57" valign="top">
    <row rowsep="1" id="id-601155ab-7095-4257-afbd-775fd2961bb6" valign="top" height="9999999">
      <entry id="id-e3a55216-ac72-415d-8168-5ac9d57f73ad" textType="default" align="left" valign="top">CALS</entry>
      <entry id="id-b427e14c-3b1c-4036-ba48-a6919b0d95b5" textType="default" align="left" valign="top"/>
      <entry id="id-45c73c20-4753-436f-914a-8a83f7671dcf" textType="default" align="left" valign="top"/>
    </row>
  </thead>
  <tbody id="id-0dc19f50-f315-4562-baf6-d9967415f7d9" valign="top">
    <row rowsep="1" id="id-03511adc-2e69-49f2-ac68-5b9c7982568d" valign="top" height="9999999">
      <entry id="id-94718508-49e2-4f42-ab65-8e60d52920ef" textType="default" align="left" valign="top">B1</entry>
      <entry id="id-774fb965-b40a-4dca-923c-b848f6225499" textType="default" align="left" valign="top"/>
      <entry id="id-ba794dc1-1146-4067-9871-03f0bfa34d0f" textType="default" align="left" valign="top"/>
    </row>
    <row rowsep="1" id="id-74049ac1-fc37-4b65-93d1-88d8fbb4a2e9" valign="top" height="9999999">
      <entry id="id-974bad6e-a1ad-4f49-bf84-e8abfbbd3a8e" textType="default" align="left" valign="top">C1</entry>
      <entry id="id-1150d921-154a-444f-bba8-7e5354dd159a" textType="default" align="left" valign="top"/>
      <entry id="id-ae336e59-b3df-464f-bbb2-dd8272a32359" textType="default" align="left" valign="top"/>
    </row>
  </tbody>
</table>
```

Figure 14-6: CALS Exchange table stored in Quark XML Author

Table

Many different table definitions have been created to capture a variety of information. This variety of definitions include tables with and without groups, sections and columns. The Quark XML Author configuration schema supports the definition of many standard table variants including CALS Exchange and HTML tables.

This section assumes that you have a working knowledge of the table structure (CALS Exchange, HTML, etc.) that you are represent in Quark XML Author configuration schema. Some structure may be discussed, but for details see their respective specifications.

The CALS Exchange table uses **table** to support table.

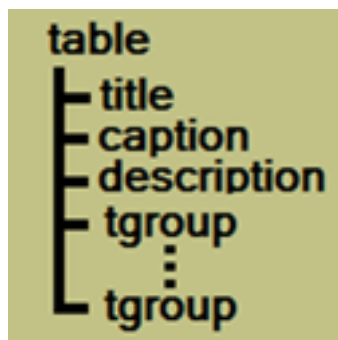


Figure 14-7: CALS Exchange Table

In the sample configuration, the “CalsTable” Table supports the CALS Exchange table.

```

<ElementDef name="CalsTable" xmlns="table" friendly="CALS Table" >
  visible="true" style="Heading Separator">
  <Table>
    <BeforeTable>...</BeforeTable>
    <TableGroupType name="CalsGroup"/>
    <AfterTable>...</AfterTable>
    <TableStyles default="Basic">...</TableStyles>
    <Templates default="General">...</Templates>
  </Table>
  <Attributes>...</Attributes>
</ElementDef>
  
```

Figure 14-8: CalsTable defined

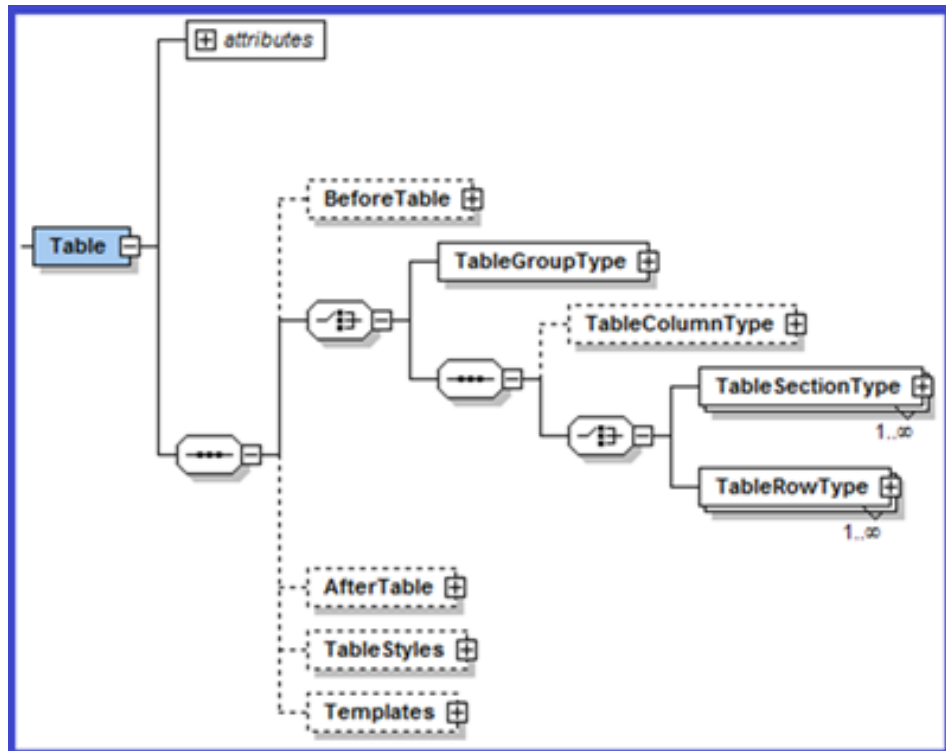


Figure 14-9: Configuration-XSD Table definition

Quark XML Author supports the following table elements:

- "Groups"
- "Sections"
- "Columns"
- "Rows"
- "Cells"
- "BeforeTable"
- "AfterTable"
- "TableStyles"
- "Templates"

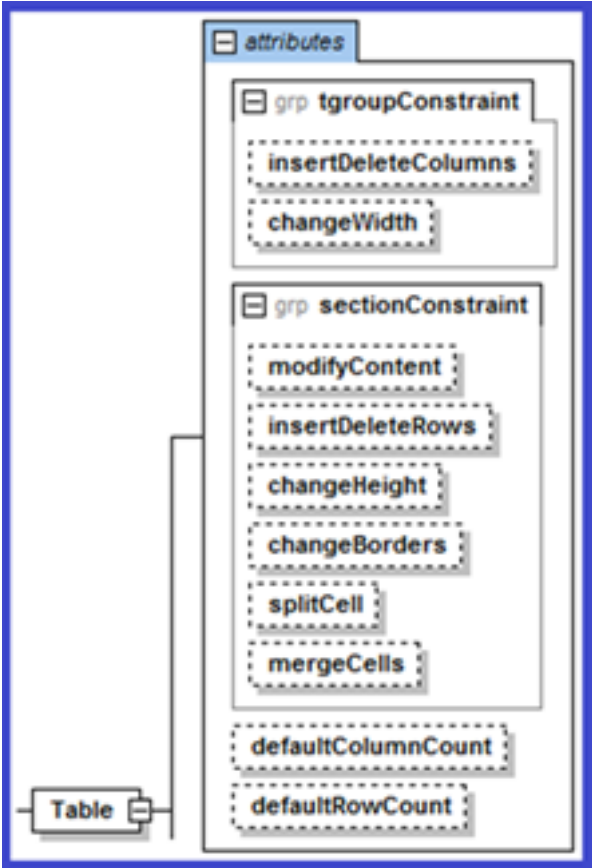


Figure 14-10: Configuration-XSD Table definition – attributes

Table contains the following attributes:

Table 14-199: attributes for Table

Attribute Name	Definition
defaultColumnCount	Default number of columns. Valid values are 1-63.
defaultRowCount	Default number of rows.

Table 14-20: tgroup constraints for Table

Attribute Name	Definition
insertDeleteColumns	Defaults to allow . If set to deny , the user will not be able to insert or delete columns in the table.
changeWidth	Defaults to allow . If set to deny , the user will not be able to change column widths in the table.

section constraints for table

The following constraint attributes may be applied to **Table**, **Tablegroup** and **Tablesection** elements.

Table 14-201: section constraint attributes for table

Attribute Name	Definition
changeBorders	Defaults to allow . If set to deny , the user will not be able to modify the borders in the table.
changeHeight	Defaults to allow . If set to deny , the user will not be able to change row heights in the table.
insertDeleteRows	Defaults to allow . If set to deny , the user will not be able to insert or delete rows in the table.
mergeCells	Defaults to allow . If set to deny , the user will not be able to merge cells in the table.
splitCell	Defaults to allow . If set to deny , the user will not be able to split cells in the table.

Constraints Inheritance Path

Table elements inherit constraint attribute values from their ancestors according to the following inheritance chain ('|' separates siblings):

Table > Group > Column | Section > Row > Cell

In tables without a group defined:

Table > Column | Section > Row > Cell

In tables without sections or a group defined:

Table > Column | Row > Cell

Fixed attribute and tables

The following statement from Table 11-3: Field Attributes also applies to tables and table elements.

“Assign the **fixed** attribute to a AttributeType for an element definition to uniquely identify the element among sibling elements that have identical **xmlname** values. “

To support multiple table elements in the same configuration use the **fixed** attribute.

For example, to support multiple table definitions in the same configuration make the **tabledef** attribute **fixed** and specify a different **tabledef** value for each definition.

This does not have to be applied to configurations with a single table definition.

Documents that contain unconfigured features

In general, documents that contain unconfigured features are still opened, but no changes pertinent to that feature are allowed. For example, if a document contains merged cells, but that feature is not configured then the existing merged cells are rendered, but no future merges are allowed.

Groups

Quark XML Authors supports table groups. Not all table types use groups.

The CALS Exchange table uses **tgroup** to support groups.

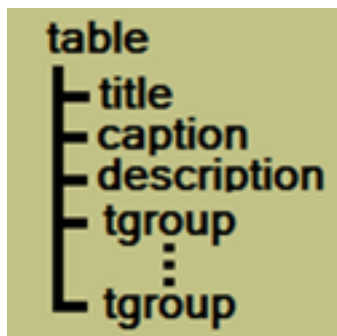


Figure 14-11: CALS Exchange Table

In the sample configuration, the “CalsGroup” **TableGroupType** definition supports the CALS Exchange **tgroup**.

```

<ElementDef name="CalsTable" xmlns="table" friendly="CALS Table">
  <Table>
    <TableGroupType name="CalsGroup"/>
    <TableStyles default="Basic">...</TableStyles>
    <Templates default="General">...</Templates>
  </Table>
  <Attributes>...</Attributes>
</ElementDef>
    
```

Figure 14-12: Reference of CalsGroup

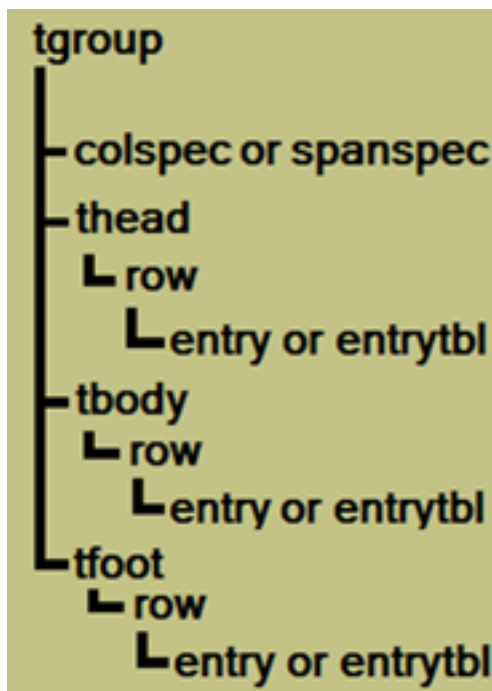


Figure 14-13: CALS Exchange Table > tgroup

```
<ElementDef name="CalsGroup" xmlns="tgroup" friendly="Table Group">
  <TableGroup>
    <TableColumnType name="CalsColumn" minOccurs="1" maxOccurs="unbounded"/>
    <TableSectionType name="CalsHeading" minOccurs="0" maxOccurs="1"/>
    <TableSectionType name="CalsBody"/>
  </TableGroup>
  <Attributes>...</Attributes>
  <ExtensibilityMethods>...</ExtensibilityMethods>
</ElementDef>
```

Figure 14-14: CalsGroup defined

The following configuration schema fragment defines the **TableGroup**.

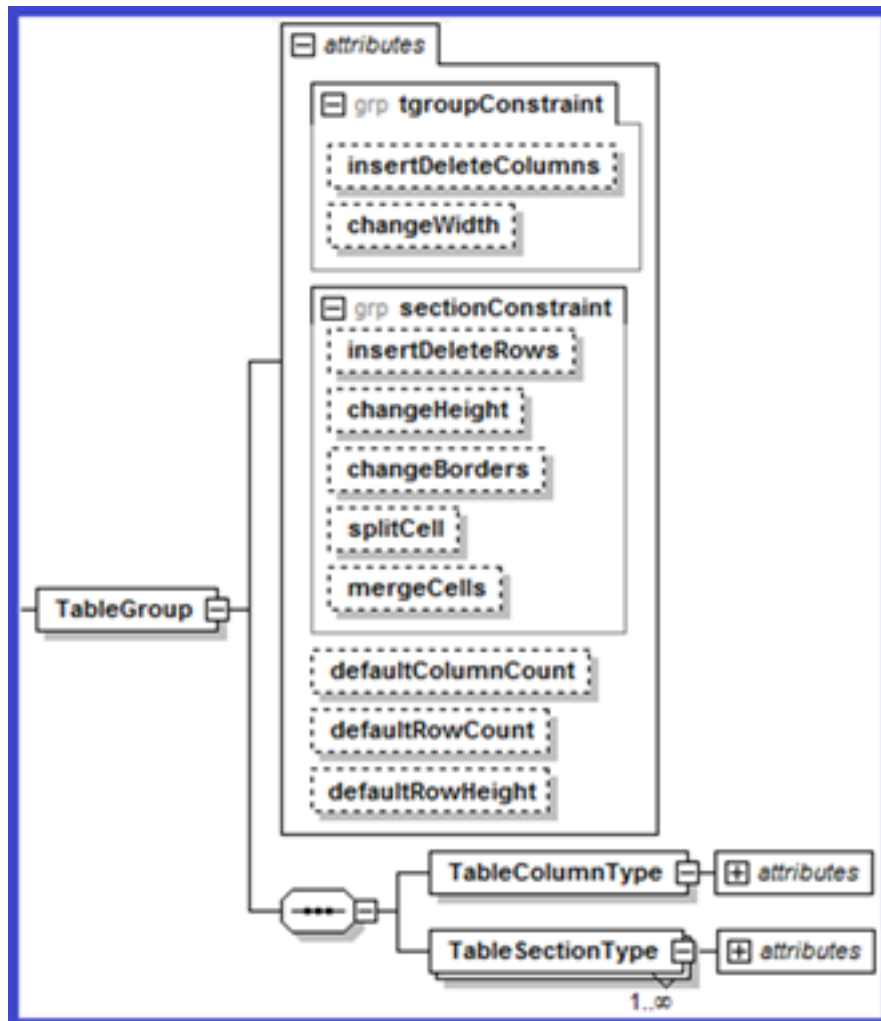


Figure 14-15: Configuration-XSD TableGroup definition

Tablegroup Constraints

TableGroup contains the following attributes:

Table 14-212: attributes for Tablegroup

Attribute Name	Definition
defaultColumnCount	Default number of columns. Valid values are 1-63.

Attribute Name	Definition
defaultRowCount	Defines the default number of rows within this group of a new instance of a table.
defaultRowHeight	Defines the default height of the row within the group.

Table elements inherit constraint attribute values from their ancestors according to the following inheritance chain. See Constraints Inheritance Path for more information.

Table 14-223: tgroup constraints for Tablegroup

Attribute Name	Definition
insertDeleteColumns	Defaults to allow . If set to deny , the user will not be able to insert or delete columns in the tablegroup.
changeWidth	Defaults to allow . If set to deny , the user will not be able to change column widths in the tablegroup.

The following constraint attributes may be applied to **Table**, **Tablegroup** and **Tablesection** elements.

Table 14-234: section constraint attributes for tablegroup

Attribute Name	Definition
changeBorders	Defaults to allow . If set to deny , the user will not be able to modify the borders in the tablegroup.
changeHeight	Defaults to allow . If set to deny , the user will not be able to change row heights in the tablegroup.
insertDeleteRows	Defaults to allow . If set to deny , the user will not be able to insert or delete rows in the tablegroup.
mergeCells	Defaults to allow . If set to deny , the user will not be able to merge cells in the tablegroup.
splitCell	Defaults to allow . If set to deny , the user will not be able to split cells in the tablegroup.

Sections

Quark XML Authors supports table sections. Tables may contain a variety of sections such as header, body and footers sections. Not all table types use sections. CALS Exchange and HTML tables use sections while DITA Simple tables do not.

Header and Body

CALS Exchange Header and Body

CALS Exchange tables use **thead** to support table header, **tbody** to support table body and **tfoot** to support table footer sections.

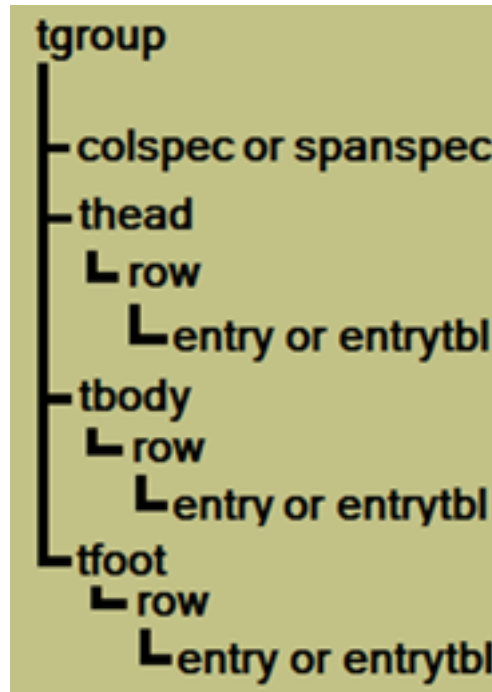


Figure 14-16: CALS Exchange Table > tgroup

In the sample configuration, the “CalsHeading” **TableSectionType** definition supports the CALS Exchange **thead**. The “CalsBody” **TableSectionType** definition supports the CALS Exchange **tbody**.

```

<ElementDef name="CalsGroup" xmlns="tgroup" friendly="Table Group">
  <TableGroup>
    <TableColumnType name="CalsColumn" minOccurs="1" maxOccurs="unbounded"/>
    <TableSectionType name="CalsHeading" minOccurs="0" maxOccurs="1"/>
    <TableSectionType name="CalsBody"/>
  </TableGroup>
  <Attributes>...</Attributes>
  <ExtensibilityMethods>...</ExtensibilityMethods>
</ElementDef>
    
```

Figure 14-17: Reference of CalsHeading and CalsBody sections

```
<ElementDef name="CalsHeading" xmlns="thead" friendly="Table Head" visible="false">
  <TableSection changeBorders="allow" changeHeight="allow" defaultRowCount="1"
    insertDeleteRows="allow" mergeCells="allow" splitCell="allow"
    <TableRowType name="CalsRow"/>
  </TableSection>
  <Attributes>
    &univ-atts;
    &global-atts;
    &table-borders-bottom-atts;
    <AttributeType name="class"/>
    <AttributeType name="outputclass"/>
    <AttributeType name="valign"/>
  </Attributes>
</ElementDef>
```

Figure 14-18: CalsHeading defined

```
<ElementDef name="CalsBody" xmlns="tbody" friendly="Table Body" visible="false">
  <TableSection changeBorders="allow" changeHeight="allow" defaultRowCount="2"
    insertDeleteRows="allow" mergeCells="allow" splitCell="allow"
    <TableRowType name="CalsRow"/>
  </TableSection>
  <Attributes>
    &univ-atts;
    &global-atts;
    &table-borders-bottom-atts;
    <AttributeType name="class"/>
    <AttributeType name="outputclass"/>
    <AttributeType name="valign"/>
  </Attributes>
</ElementDef>
```

TableSection

The following configuration schema fragment defines the TableSection.

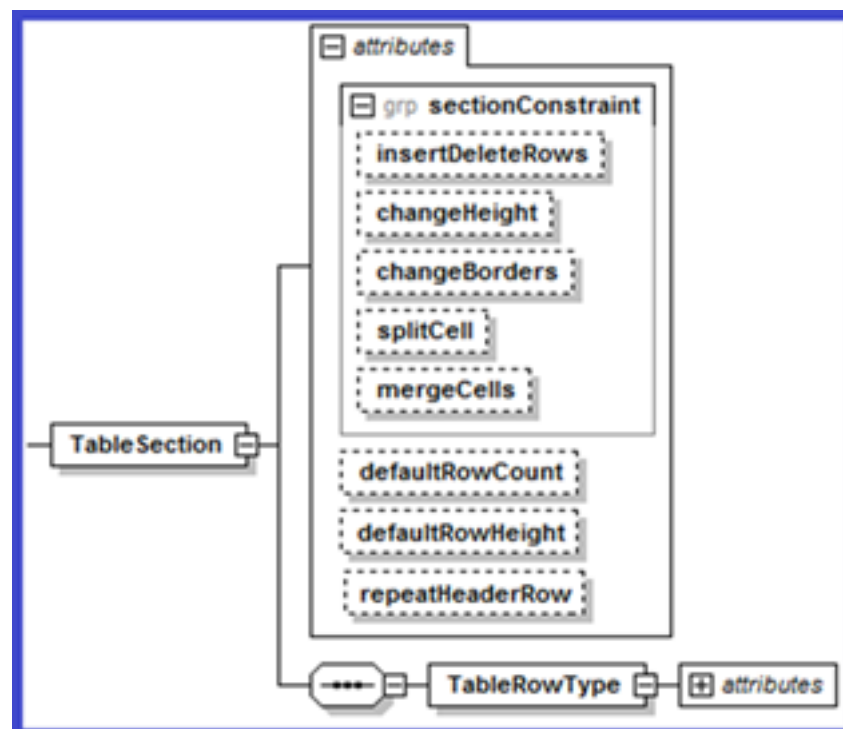


Figure 14-20: Configuration-XSD TableSection definition

Table elements inherit constraint attribute values from their ancestors according to the following inheritance chain. See Constraints Inheritance Path for more information.

TableSection contains the following attributes:

Table 14-245: attributes for Tablesection

Attribute Name	Definition
defaultRowCount	Defines the default number of rows within this section of a new instance of a table.
defaultRowHeight	Defines the default height of the rows within this section.
repeatHeaderRows	Boolean. Specifies that the section is a header section. Specifies whether or not the “repeat header rows” feature is enabled. Replaces the deprecated TableHeading Internal Class.

This needs to be on TableGroup and TableSection attributes.

The following constraint attributes may be applied to **table**, **tablegroup** and **tablesection** elements.

Table 14-256: section constraint attributes for tablesection

Attribute Name	Definition
changeBorders	Defaults to allow . If set to deny , the user will not be able to modify the borders in the tablesection.
changeHeight	Defaults to allow . If set to deny , the user will not be able to change row heights in the tablesection.
insertDeleteColumns	Defaults to allow . If set to deny , the user will not be able to insert or delete columns in the tablesection.
insertDeleteRows	Defaults to allow . If set to deny , the user will not be able to insert or delete rows in the tablesection.
mergeCells	Defaults to allow . If set to deny , the user will not be able to merge cells in the tablesection.
splitCell	Defaults to allow . If set to deny , the user will not be able to split cells in the tablesection.

Columns

Quark XML Author supports columns, but does not support column-centric tables. Therefore, HTML **colgroup** is not supported. Quark XML Author assumes that any

given table must have at least one column and ignores any constraints that contradict that requirement.

CALS Exchange uses **colspec** to support columns.

```
-<tgroup cols="3" id="id" textType="default" align="left" valign="top">
  <colspec colwidth="152.90*" align="left"/>
  <colspec colwidth="152.90*" align="left"/>
  <colspec colwidth="152.90*" align="left"/>
```

Figure 14-21: CALS Exchange table columns stored in Quark XML Author

In the sample configuration, the “CalsColumn” **TableColumnType** definition supports the CALS Exchange **colspec**.

```
<ElementDef name="CalsGroup" xmlns="tgroup" friendly="Table Group">
  <TableGroup>
    <TableColumnType name="CalsColumn" minOccurs="1" maxOccurs="unbounded"/>
    <TableSectionType name="CalsHeading" minOccurs="0" maxOccurs="1"/>
    <TableSectionType name="CalsBody"/>
  </TableGroup>
  <Attributes>...</Attributes>
  <ExtensibilityMethods>...</ExtensibilityMethods>
</ElementDef>
```

Figure 14-22: Reference of CalsColumn

```
<ElementDef name="CalsColumn" xmlns="colspec" visible="false">
  <TableColumn/>
  <Attributes>
    &table-borders-right-atts;
    &table-borders-bottom-atts;
    <AttributeType name="colnum"/>
    <AttributeType name="colname"/>
    <AttributeType name="colwidth"/>
    <AttributeType name="align"/>
    <AttributeType name="valign"/>
  </Attributes>
</ElementDef>
```

Figure 14-23: CalsColumn defined

The following configuration schema fragment defines the **TableColumn**.



Column Resize, Split and Merge Feature Requirements

Quark XML Author only supports the column resize, split and merge features if the following are defined:

- splitCell constraint attribute is set to **allow**. See Table 14-256: section constraint attributes for tablesection.
- columns are defined, such as the CALS Exchange **colspec**

- the column definition includes an attribute with datatype equals **tableColumnWidth**
- at least one of the following type attributes exists:
 - tableCellColumnStart, tableCellColumnEnd and tableRowColumnIdentifier. See “Table 11-7: Field Attributes –datatype values for table rows/columns”. OR
 - tableCellColumnCount. See “Table 11-8: Field Attributes – datatype values for table cells”. OR
 - tableCellRowCount. See “Table 11-9: Field Attributes – datatype values for table, group, section”.
- resulting table do not exceed the maxCount of either row or columns (applicable only for split)

In the sample configuration, merge, split and resize of columns are supported.

```

<!-- C O L U M N S -->
<AttributeDef name="colnum" datatype="tableRowColumnIndex"
              friendly="" visible="false"/>
<AttributeDef name="colname" datatype="tableRowColumnIdentifier"
              friendly="" visible="false"/>
<AttributeDef name="colwidth" datatype="tableRowColumnSize"
              friendly="" visible="false"/>
<!-- R O W -->
<AttributeDef name="rowHeight" xmlname="height"
              datatype="tableRowColumnSize" visible="false"/>
<!-- C E L L S -->
<AttributeDef name="namest" datatype="tableCellColumnStart"
              friendly="" visible="false"/>
<AttributeDef name="nameend" datatype="tableCellColumnEnd"
              friendly="" visible="false"/>
<AttributeDef name="morerows" datatype="tableCellRowCount"
              friendly="" visible="false"/>
<AttributeDef name="rotate" datatype="tableCellContentRotate"
              friendly="" visible="false"/>
    
```

Figure 14-25: configuration for merge, split and resize of columns

The merge feature uses the mergeCells constraint attribute at the table, group and section levels:

See “Table 14-201: section constraint attributes for table”.

See “Table 14-234: section constraint attributes for tablegroup”.

See “Table 14-256: section constraint attributes for tablesection”.

The merge feature uses the following logic which is based on two patterns. If tableCellColumnStart and tableCellColumnEnd are defined and tableRowColumnIdentifier has a unique value, then horizontal merge is permitted using these three attributes. If not, then Quark XML Author determines if tableCellColumnCount is defined, and if so, horizontal merge is permitted using that attribute. The first pattern is considered the CALS pattern while the second is the HTML pattern.

The following summarizes the datatypes needed for the merge feature.

For HTML,

horizontal:

- datatype = tableCellColumnCount facilitates colspan

vertical:

- datatype = tableCellRowSpan facilitates rowspan

```
<!-- HTML table attributes-->
<AttributeDef name="border" friendly="Border"/>
<AttributeDef name="width" datatype="tableColumnWidth"
visible="false"/>
<AttributeDef name="colspan" datatype="tableCellColumnCount"
visible="false"/>
<AttributeDef name="rowspan" datatype="tableCellRowSpan"
visible="false"/>
```

For CALS,

horizontal:

- datatype = tableCellColumnStart facilitates namest
- datatype = tableCellColumnEnd facilitates nameend

vertical:

- datatype = tableCellRowCount(deprecated, but for backward compatibility set to tableCellMoreRows)
- datatype=tableCellMoreRows facilitates morerows

See “Table 11-8: Field Attributes – datatype values for table cells” for details on datatypes used by the merge feature.

Pre 4.0 Table Columns

In Quark XML Author version 4.0, if the user creates a table in a Quark XML Author document and the table configuration contains a column definition then column elements such as the HTML colspec are created for the table in the document. However, if a pre-4.0 document is similarly opened, column elements are not created. Therefore in these pre 4.0 tables, the user will not be able to perform the following features: column resize, split and merge.

Column Widths

To be able to change the column width, an attribute with datatype equals **tableColumnWidth** must be included in the column definition. See Table 11-9: Field Attributes – datatype values for table, group, section. And **changeWidth** must be set to “allow”. See Table 14-20: tgroup constraints for Table.

Fit to Window table

A table upon which the user has invoked the AutoFit > Fit to Window feature or the template contains 'proportional' units. The xml for these tables will contain column widths expressed in 'proportional' units as indicated by the trailing asterisk.

```
<colspec colsep="1" colnum="1" colname="001" colwidth="113.40*"/>
<colspec colsep="1" colnum="2" colname="002" colwidth="148.50*"/>
<colspec colsep="1" colnum="3" colname="003" colwidth="196.80*"/>
```

Figure 14-26: example xml for Fit to Window feature

If a table does not allow the user to change column widths, but the user inserts or deletes columns, then the user will have to use the columns width that the system calculates after the action.

Fixed Column Width table

A table upon which the user has invoked the AutoFit > Fixed Column Width feature or the template contains 'absolute' units. The xml for these tables will contain column widths expressed in 'absolute' units, such as 'points'.

```
<colspec colsep="1" colnum="1" colname="001" colwidth="113.40pt"/>
<colspec colsep="1" colnum="2" colname="002" colwidth="148.50pt"/>
<colspec colsep="1" colnum="3" colname="003" colwidth="196.80pt"/>
```

Figure 14-27: example xml for Fixed Column Width feature.

Rows

Some tables store rows in sections. Others store rows at the root of the table.

Some table types use sections to contain rows. Other table types allow rows to exist without sections at the root of the table. The following applies to both table types.

In the sample configuration, the "CalsRow" **TableRowType** definition supports the CALS Exchange row.

```
<ElementDef name="CalsBody" xmlns="tbody" friendly="Table Body" visible="false">
  <TableSection changeBorders="allow" changeHeight="allow" defaultRowCount="2"
    insertDeleteRows="allow" mergeCells="allow" splitCell="allow" type="Body">
    <TableRowType name="CalsRow"/>
  </TableSection>
  <Attributes>...</Attributes>
</ElementDef>
```

Figure 14-28: Reference of CalsRow in a section

```
<ElementDef name="CalsRow" xmlns="row" friendly="Row" visible="false">
  <TableRow>
    <TableCellType name="CalsEntry"/>
  </TableRow>
  <Attributes>
    &univ-atts;
    &global-atts;
    &table-borders-bottom-atts;
    <AttributeType name="class"/>
    <AttributeType name="outputclass"/>
    <AttributeType name="valign"/>
    <AttributeType name="rowHeight"/>
  </Attributes>
</ElementDef>
```

Figure 14-29: CalsRow defined

The following configuration schema fragment defines the **TableRow**.

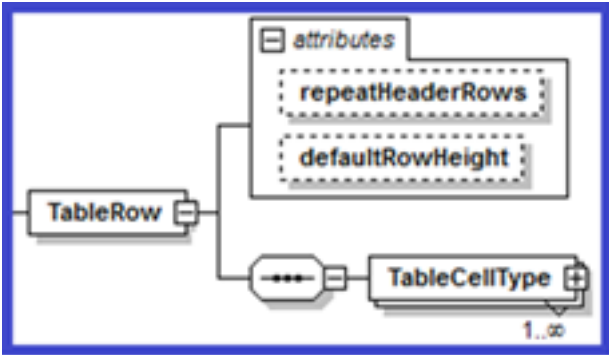


Figure 14-30: Configuration-XSD TableRow definition

TableRow contains the following attributes:

Table 14-26: attributes for Tablerow

Attribute Name	Definition
defaultRowHeight	Defines the default height of the row.
repeatHeaderRows	Boolean. Specifies whether or not the “repeat header rows” feature is enabled.

Row Heights

To be able to change the row height width, an attribute with datatype equals **tableRowHeight** must be included in the row definition. See Table 11-9: Field Attributes – datatype values for table, group, section. And **changeHeight** must be set to “allow”. See “Table 14-201: section constraint attributes for table”.

You may specify the value of **changeHeight** at each level in the table hierarchy or allow the setting to be inherited: Table, TableGroup and TableSection. See “Table 14-20: tgroup constraints for Table” for more information. This is useful if you want to create different user experiences. For example, you can create a table configuration where the user can change the height of body rows, but not header rows.

AutoFit

The **AutoFit** feature is not available when the selection includes any content outside of a table.

Cells

First let's discuss the authoring experience and content structure within a table cell, then we'll discuss defining the cell.

Authoring Experience and Content Structure

Many different user experiences can be designed and implemented in Quark XML Author. This holds true both outside of tables and inside of tables. For example, the display of structural indicators for various elements can be toggled on and off (visible attribute = true or false). The display of the table cell structural indicator is also possible, but in the following example we do not display the indicator. The user is not able to enter text in the table cell structural indicator.

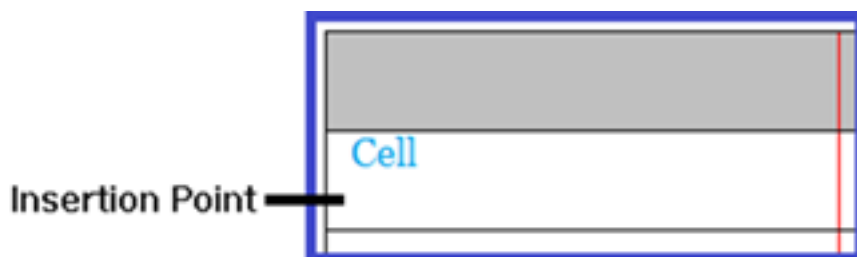


Figure 14-31: table cell with visual indicator.

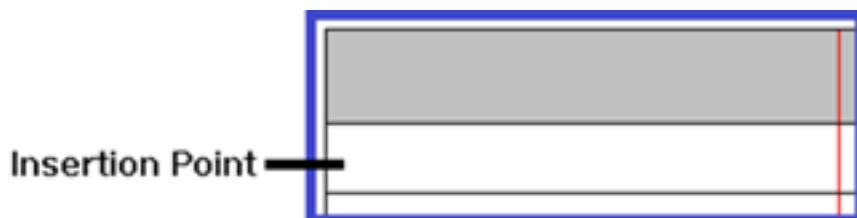


Figure 14-32: table cell without visual indicator.

The following is an excerpt from the Quark XML Author User Guide. This version of the product does not display a structural indicator for the Cell element.

User Guide

Authoring in a table cell is similar to authoring on the Word canvas outside of a table. Add new content in each table cell using Insertion Points. You can think of it as authoring in a miniature Word canvas. The content structure for this canvas is the Cell element. A visual indicator for the Cell element is not provided. Regardless, the Cell element is accessible on the Context Menu > Component Menu.

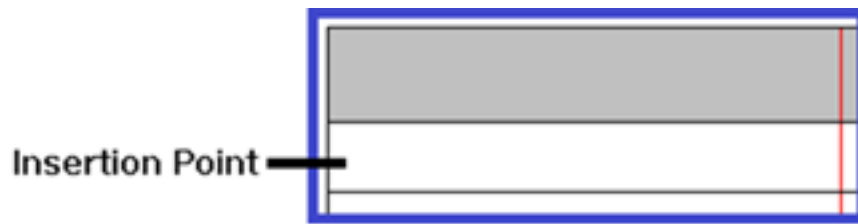


Figure 14-33: An empty cell

If you want a paragraph at the top of the table cell, simply enter the text. In this situation, the system can be configured to automatically insert a paragraph element into the Cell element. The system can insert other types of content elements depending on the configuration. See “Table 14-278: TableCell Attributes” for more information on specifying a **defaultElement** for the TableCell.

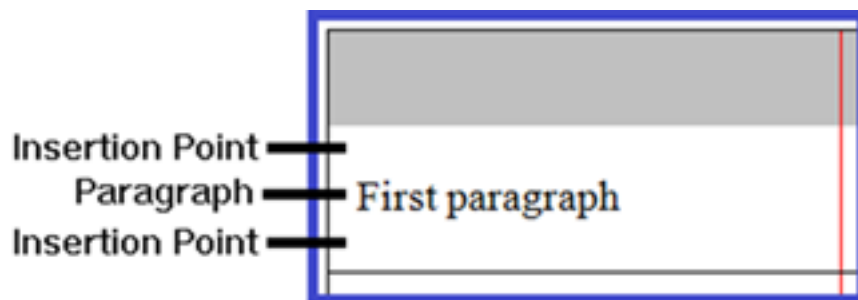


Figure 14-34: A cell with text only

Otherwise, insert the desired element type in the Insertion Point using standard insert techniques. For example, right-click on the Insertion Point and the Context Menu of allowed elements is displayed.

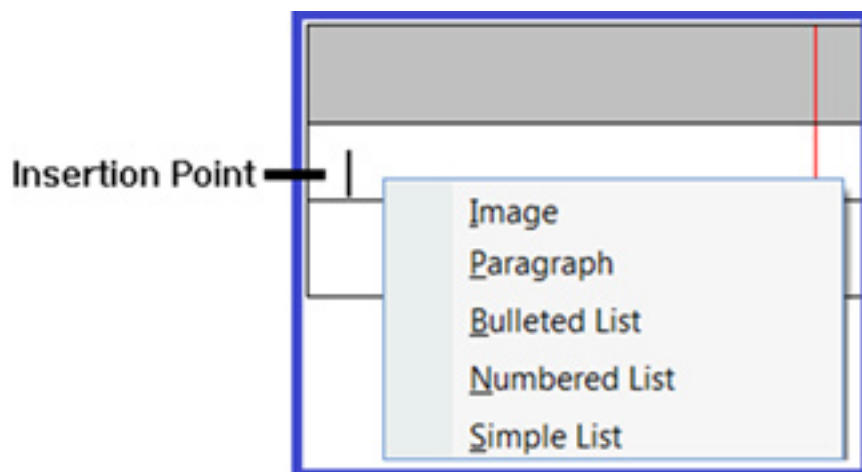


Figure 14-35: Cells support mixed content.

In our example, we select Bulleted List from the Context Menu.

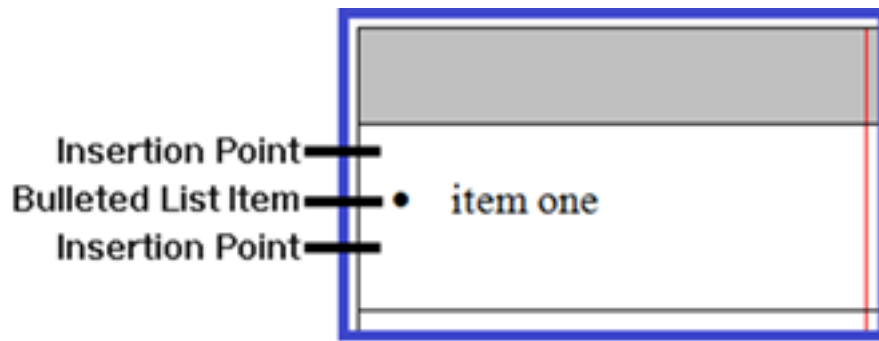


Figure 14-36: Cells with bulleted list only

The bulleted list is inserted and ready to receive text. We enter “item one”.

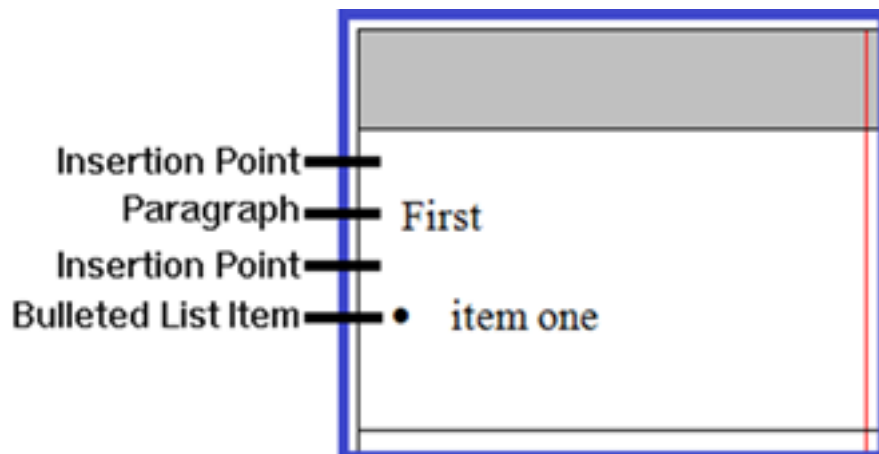
Elements inserted in the table cell are children of the Cell.

The content structure for this table is represented as the cell with a child bulleted list.



Figure 14-37: The content structure

If later you decide that you need text at the top of the cell, simply insert a paragraph element in the topmost Insertion Point.



The content structure for this table is represented as the cell with a child paragraph and a child bulleted list.



Figure 14-39: The content structure


Text (1,1)	Text (1,2)	Text (1,3)
Text (2,1)	Text (2,2)	Text (2,3)
<ul style="list-style-type: none"> bulleted list item 	<ol style="list-style-type: none"> numbered list item 	 <p>Text (3,3)</p>

Figure 14-40: CALS table cells with various content types

childTextNodes

The XML of the CALS table in Figure 14-40 is the following (assumption: nodes are defined as childTextNodes not Paragraphs. See “Paragraph and ChildTextnode Elements”) :

```

<table id="_dd2f9581...">
<tgroup cols="3" id="_18fd38aa...">
  <colspec colsep="1"/>
  <colspec colsep="1"/>
  <colspec colsep="1"/>
<thead id="_98e2fd05...">
<row rowsep="1" id="_b03e9318...">
  <entry id="_3c18b0a7...">Text (1,1)</entry>
  <entry id="_51f6bb62...">Text (1,2)</entry>
  <entry id="_651eal5a...">Text (1,3)</entry>
</row>
</thead>
<tbody id="_c9cff7c1...">
<row rowsep="1" id="_cabff024...">
  <entry id="_a0c9557f...">Text (2,1)</entry>
  <entry id="_13f0110c...">Text (2,2)</entry>
  <entry id="_5cf2483a...">Text (2,3)</entry>
</row>
<row rowsep="1" id="_7d2callc...">
<entry id="_3eff3aac...">
<ul id="_7ab1ac56...">
  <li id="_4f9048b1...">bulleted list item</li>
</ul>
</entry>
<entry id="_2b0c3181...">
<ol id="_7131af6a...">
  <li id="_0edc9e7e...">numbered list item</li>
</ol>
</entry>
<entry id="_0d668949...">
  <image href="C:\Current\camera_back_menu_96dpi.jpg" height="65"
width="122" id="_fab4eff5...">
  <p id="_bd16267b...">Text (3,3)</p>
</entry>
</row>
</tbody>
</tgroup>
</table>

```

Cell Definition Overview

For some tables, cell are defined the same (reusable) no matter whether they are in a header, body or footer. For other tables, cell definitions are separate and different for different parent containers.

- Reusable Cells Definition
- Body Cell Canvas
- Header Cells
- Cell Canvas

Table cells may be configured to contain mixed-content such as paragraphs, lists, and figures. For an example, see the **Choice** (unbounded) definition in “CALS Exchange Cells”.

Reusable Cell Definitions

Some table structures use section-specific cells or row-specific cells.

CALS Exchange Cells

If your table structure has sections and the cells within the different section types are similar, you may be able to create a single cell element definition rather than a section-specific cell definition. This is the case with the following CALS Exchange table example.

In the sample configuration, the “CalsEntry” **TableCellType** definition supports CALS Exchange **entry**. CalsRow is referenced by both CalsHeading and CalsBody effectively reusing “CalsEntry” in both section types. See “CALS Exchange Header and Body.

```
<ElementDef name="CalsRow" xmlns="row" friendly="Row"
  visible="false" allowCut="false" allowRemove="false">
  <TableRow>
    <TableCellType name="CalsEntry"/>
  </TableRow>
  <Attributes>...</Attributes>
</ElementDef>
```

Figure 14-41: Reference of CalsEntry

```
<ElementDef name="CalsEntry" xmlns="entry" friendly="Cell"
  contextId="R-2-4-7" visible="false">
  <TableCell>
    <Sequence minOccurs="0">
      <UnmanagedType name="prolog"/>
    </Sequence>
    <Choice minOccurs="0" maxOccurs="1">
      <SectionType name="toctitle"
        friendly="Unnumbered Title" style="Heading 2"
        default="**** Type Level 2 title ****"
        increaseIndent="tocnumtitle"/>
      <SectionType name="tocnumtitle"
        friendly="Numbered Title" style="Heading 2 Numbered"
        default="**** Type Level 2 title ****"
        increaseIndent="toctitle"/>
    </Choice>
    <Choice minOccurs="0" maxOccurs="unbounded">
      <SectionType name="bulletul"/>
      <SectionType name="discul"/>
      <SectionType name="numol"/>
      <SectionType name="bigaol"/>
      <SectionType name="sFigure"/>
      <SectionType name="foreign"/>
      <SectionType name="CellChildTextNode" insertAfter="CellChildTextNode"/>
    </Choice>
  </TableCell>
  <Attributes>...</Attributes>
</ElementDef>
```

Figure 14-42: CalsEntry defined

In the sample configuration, the “CellChildTextNode” element definition supports text capture within CALS Exchange **entry**. Text-based elements specified as childTextnodes are typically used for text capture in table cells. See “Paragraph and ChildTextnode Elements” for more information.

```
<ElementDef name="CellChildTextNode">
  <Section>
    <Para defaultHyperlink="xrefemph@outputclass='inter'"
      childTextnode="true">
      <Emphasis>
        &emphasis-ref;
        <EmphasisType name="fontSize"/>
        <EmphasisType name="tn"/>
      </Emphasis>
      <Media>
        <!-- Contains all supported media types -->
        <MediaType name="image"/>
        <MediaType name="refImage"/>
      </Media>
      <OLE>
        <!-- Contains all supported OLE types -->
        <OLEType name="content"/>
      </OLE>
    </Para>
  </Section>
</ElementDef>
```

Figure 14-43: CellChildTextNode defined

```
<ElementDef name="CalsEntry" xmlns="entry" friendly="Cell"
  contextId="R-2-4-7" visible="false">
  <TableCell>
    <Sequence minOccurs="0">...</Sequence>
    <Choice minOccurs="0" maxOccurs="1">...</Choice>
    <Choice minOccurs="0" maxOccurs="unbounded">...</Choice>
  </TableCell>
  <Attributes>
    <AttributeType name="namest"/>
    <AttributeType name="nameend"/>
    <AttributeType name="morerows"/>
    <AttributeType name="rotate"/>
    <AttributeType name="rowsep"/>
    <AttributeType name="valign"/>
    <AttributeType name="colsep"/>
    <AttributeType name="align"/>
  </Attributes>
</ElementDef>
```

Figure 14-44: CalsEntry (with attributes) defined

TableCell contains the following attributes:

Table 14-278: TableCell Attributes

Attribute	Required	Definition
defaultElement	no	Specifies the name of the element that is automatically inserted if the user tabs to a table cell and then types (or pastes text). This should be a para-type. See “Minimizing “empty” cells”.
displayAttribute	no	String. Allows you to specify that instead of the element name, the value of the attribute named will be displayed to the user. See Table 14-2: Section Attributes.
displayElement	no	Boolean. If set to true , the display name of the element will be the name of the element’s first visible child element. See Table 14-2: Section Attributes.

A CALS Exchange cell requires many more attributes to support it than an HTML cell and about twice as many as the Simple Table cell. Xmlname is different. CALS has Attributes. HTML does not.

Minimizing “empty” cells

In order to ensure a consistent and reliable user experience, the system attempts to ensure that new table cells contain some type of content element rather than create what are considered “empty” cells. If the table cell is configured with a required child element then that child element is instantiated or if the table template included content for the table cell then that content is be placed in the cell. If the neither of those scenarios are true then the system checks to see if the table cell is configured with a default element. If so, then the default element is inserted. This applies whether the user inserted a new table, inserted a new row, tabbed to create a new row, pasted content, etc. See defaultElement in “Table 14-27”.

Paragraph and ChildTextnode Elements

Text is a common content type that users desire to enter in table cells. This content type can be facilitated using two types of content elements: paragraphs and childTextnodes. These elements are configured as children of the table cell. Both support portionmarks. See “Portionmarks”.

Table cell elements are containers only and may not have text stored directly on the cell element. Table cells are strictly containers for child elements. Most table cell configurations use childTextnodes rather than paragraphs. The user experience for paragraphs and childTextnodes is similar. See “childTextnode” for a description of some of the differences.



Quark strongly recommends that your table cell configurations contain either paragraphs or childTextnodes, but not both. Though a configuration containing both is technically possible, the resulting user experience could be confusing. When viewing the list of insertable content, the user would be presented with both paragraph and childTextnodes. Until the user learned the subtle differences, this user experience could be frustrating.

The following sample configuration fragment defines a childTextnode within a TableCell > Choice “unbounded”.

```
<SectionType name="CellChildTextNode"
insertAfter="CellChildTextNode"/>
```

The following sample configuration fragment defines a paragraph within a TableCell > Choice “unbounded”.

```
<SectionType name="CellPara" insertAfter="CellPara"/>
```



If you have a business requirement for text-only table cells you must use childTextnodes. Configurations of text-only tables cells using paragraphs

are not supported. For example, DITA Choice Table should be configured using childTextnodes.

The serialization of paragraphs and childTextnodes is different. See “Table 14-3: Para Attributes” for more information.

childTextNodes

childTextnodes have the following limitations and differences from a paragraph:

- Shift-Enter required to create soft line break
- Adjacent childTextnodes are supported. This allows for the **insertAfter** attribute to be set such that when the user presses Enter in a childTextnode, a sibling childTextnode is automatically inserted after. See Table 14-133: SectionType Attributes for more information on **insertAfter**. This also allows for automatic insertion of childTextnode in the tab and type use case. See default Element and “Table 14-278: TableCell Attributes” for more information.
- The “Change To” feature is supported. See “ChangeToList” for more information.

Referencing

Only certain aspects of a table can be configured for referencing in Quark XML Author.

Configurable:

- tables
- paragraphs
- lists
- images

Not configurable:

- groups
- sections
- rows
- columns
- cells
- textnodes

Cut and Copy

Cut, Copy and Remove is configured the same inside and outside of tables using standard allowCopy and allowCut attributes. See Table 14-1: ElementDef Attributes for more information.

These features will be available when they do not endanger structural integrity. For example, the user is never allowed to cut a cell from inside of a table.

Configurations with multiple table types

Special consideration should be taken when creating a configuration with multiple table types because the user may attempt to paste content from one table to another table of a different type. The system will allow the paste if the content being pasted is compatible with respect to the configurations of each table. For example, the user may not be able to paste header row content into a table that is not configured for header rows.

Given that the system will deny the user to paste content that is incompatible, the user guide and training materials should be written to provide guidance to the user as to why the paste may have been denied and what is allowed for that specific configuration.

Formatting

Formatting is limited to:

- alignment: Table 14-291: tableStyles attributes group
- borders: Table 14-291: tableStyles attributes group, Table 14-302: xpressColumnStyleAttributes group, Table 14-313: xpressRowStyleAttributes group
- orientation: See tableCellContentRotate in “Table 11-8: Field Attributes – datatype values for table cells”

BeforeTable

If you need to define a collection of elements that can exist before a table and are directly associated with that table, then use BeforeTable. <BeforeTable> is a child element of <Table>. See Table.

BeforeTable may contain one or more Sequence or Choice elements, which in turn contain document elements. Each <BeforeTable> node defines a collection of <Sequence> and <Choice> nodes. These are the same Sequence or Choice elements that are used to define content elements outside of tables. See Sequence.

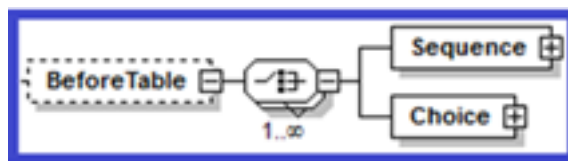


Figure 14-46: Configuration-XSD BeforeTable definition

```

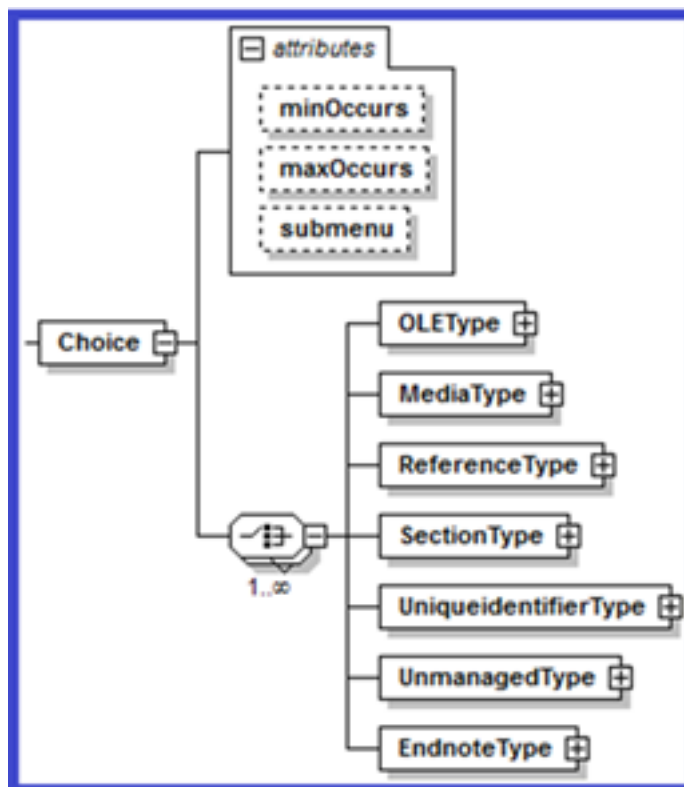
<Table>
  <BeforeTable>
    <Choice minOccurs="0" maxOccurs="unbounded">
      <SectionType name="CalsTableTitle" friendly="CALs Table Title"
        style="Table Title"/>
      <SectionType name="bulletul"/>
    </Choice>
  </BeforeTable>
  <TableGroupType name="CalsGroup"/>
  <AfterTable>
    <Choice minOccurs="0" maxOccurs="unbounded">
      <SectionType name="CalsTableCaption" friendly="CALs Table Caption"
        style="Table Caption"/>
      <SectionType name="numol"/>
    </Choice>
  </AfterTable>
  <TableStyles default="Basic">...</TableStyles>
  <Templates default="General">...</Templates>
</Table>

```

Figure 14-47: BeforeTable defined

Choice

Each <Choice> node defines a collection of content type nodes.



Sequence

Each <Sequence> node defines a collection of content type nodes.

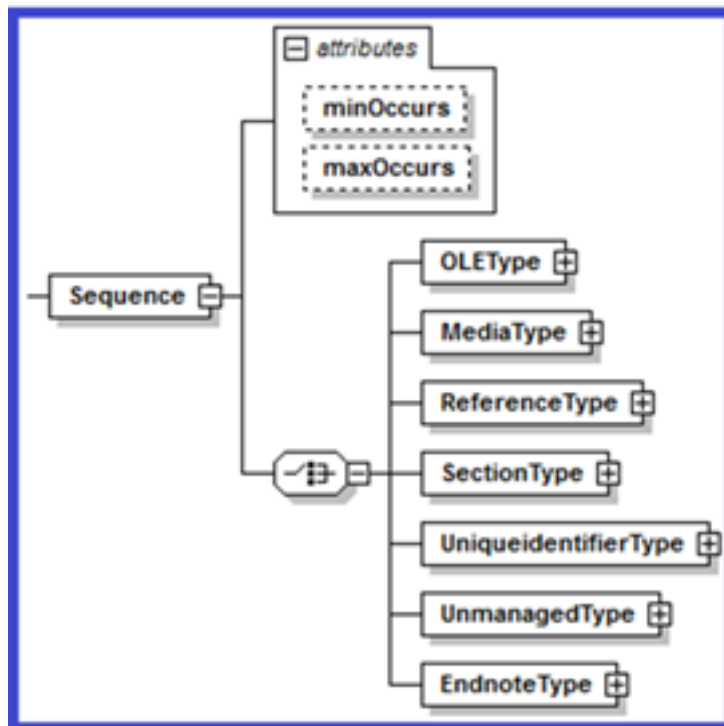


Figure 14-49: Configuration-XSD Sequence definition

AfterTable

If you need to define a collection of elements that can exist after a table and are directly associated with that table, then use AfterTable. <AfterTable> is a child element of <Table>. See Table.

AfterTable may contain one or more Sequence or Choice elements, which in turn contain document elements. Each <AfterTable> node defines a collection of <Sequence> and <Choice> nodes. These are the same Sequence or Choice elements that are used to define content elements outside of tables. See Sequence.

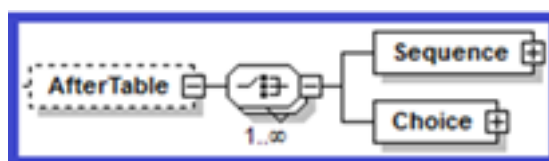


Figure 14-50: Configuration-XSD AfterTable definition

```

<Table>
  <BeforeTable>
    <Choice minOccurs="0" maxOccurs="unbounded">
      <SectionType name="CalsTableTitle" friendly="CALS Table Title"
                    style="Table Title"/>
      <SectionType name="bulletul"/>
    </Choice>
  </BeforeTable>
  <TableGroupType name="CalsGroup"/>
  <AfterTable>
    <Choice minOccurs="0" maxOccurs="unbounded">
      <SectionType name="CalsTableCaption" friendly="CALS Table Caption"
                    style="Table Caption"/>
      <SectionType name="numol"/>
    </Choice>
  </AfterTable>
  <TableStyles default="Basic">...</TableStyles>
  <Templates default="General">...</Templates>
</Table>

```

Figure 14-51: AfterTable defined

Choice

Each <Choice> node defines a collection of content type nodes.

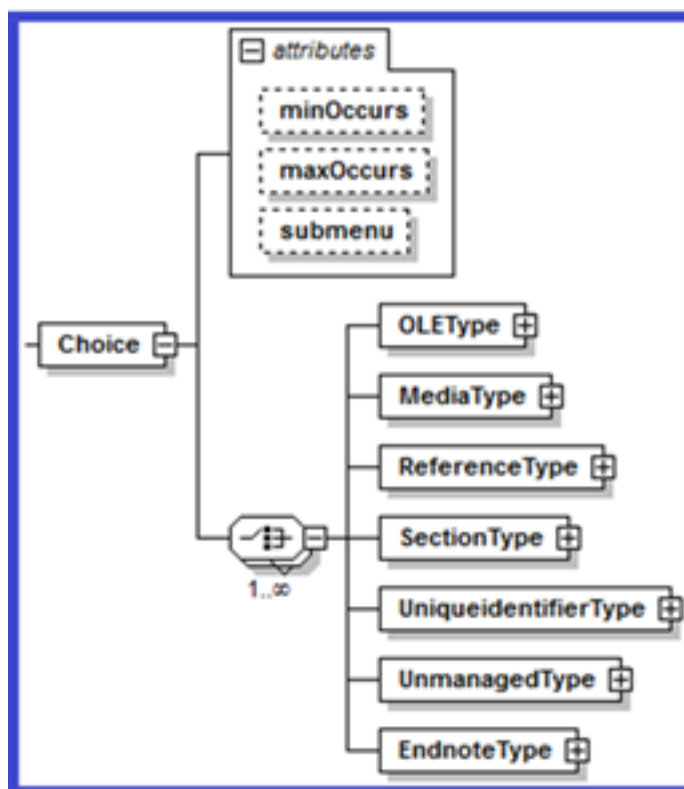
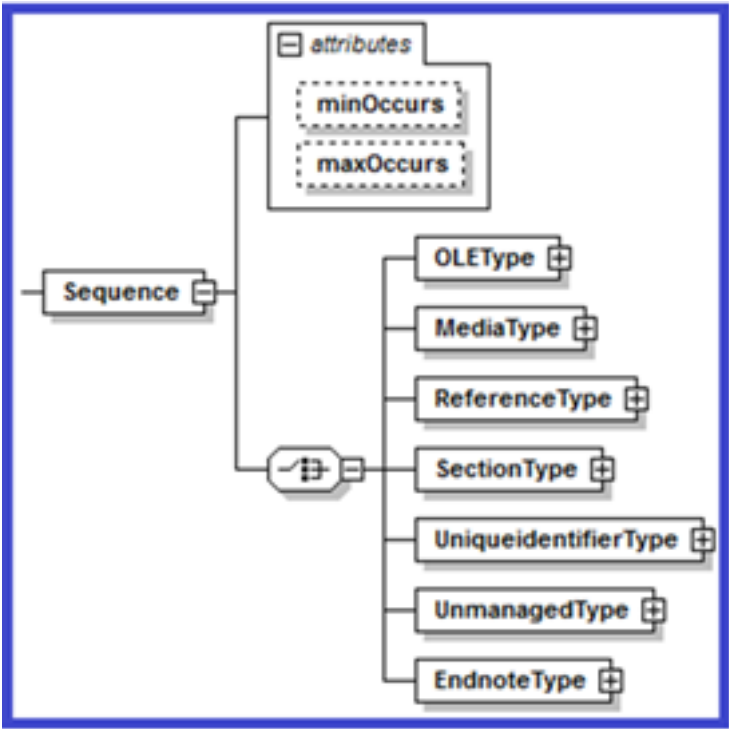


Figure 14-52: Configuration-XSD Choice definition

Sequence

Each <Sequence> node defines a collection of content type nodes.



TableStyles

The <TableStyles> node contains a collection of <TableStyle> child elements that describe general formatting for tables.

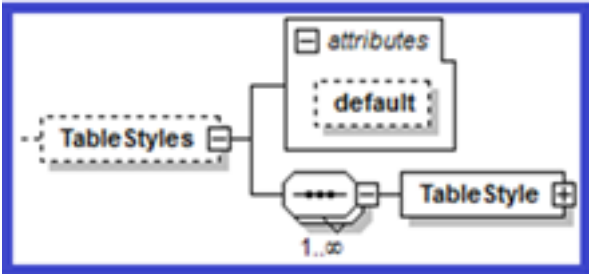


Figure 14-54: Configuration-XSD TableStyles definition

Table 14-289: TableStyles Attributes

Attribute Name	Definition
default	The name of the default TableStyle.

In the following example, the table style “Basic” is defined. Within the collection of table styles, “Basic” is also specified as the default style.

```
<TableStyles default="Basic">
  <TableStyle name="Basic">
    <Table backColor="" columnBorder="FF0000" columnBorderStyle="Single"
      columnBorderWidth="" horizAlign="" rowBorder="LightGray" rowBorderStyle="Single" rowBorderWidth="" vertAlign="">
      <Section name="CalsHeading" horizAlign="center" vertAlign="middle"
        backColor="LightGray" rowBorder="LightGray" rowBorderStyle="Single" rowBorderWidth="">
      <Section name="CalsBody" horizAlign="" vertAlign="" backColor=""
        rowBorder="LightGray" rowBorderStyle="Single" rowBorderWidth="">
    </Table>
  </TableStyle>
</TableStyles>
```

TableStyle

Each <TableStyle> node defines a <Table> node.

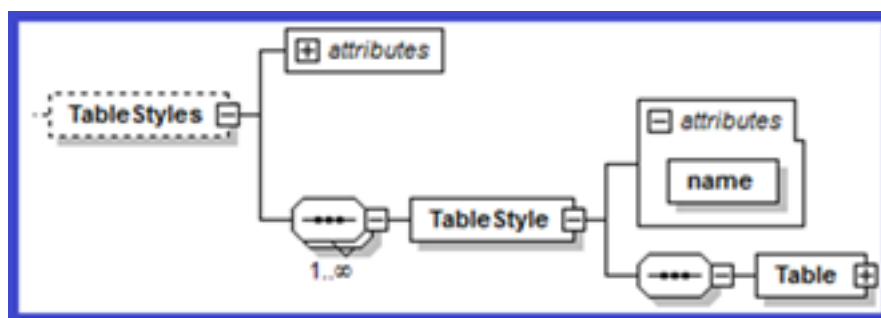


Figure 14-56: Configuration-XSD TableStyle definition

Table 14-30: TableStyle Attributes

Attribute Name	Required	Definition
name	yes	<p>The name of the TableStyle. The table element must have a tableStyle attribute.</p> <pre><AttributeDef name="myTemplate" datatype="tableStyle"/></pre> <p>And in the Template > table definition, set that attribute value to the name of the style. For example, Template/table@myTemplate="My Template", where TableStyle@name="My Template".</p>

The name attribute identifies the table style within Quark XML Author as well as to the user.

In the **Insert Table** dialog, the name is displayed adjacent to "Table Style:" indicating the style that was selected by the user in the **Table Style** dialog. In the following example, the current table style is "Default". Note that the Styles button is not enabled which indicates that only one Table Style is associated with the "Simple Table" table definition and no other styles are available for selection.

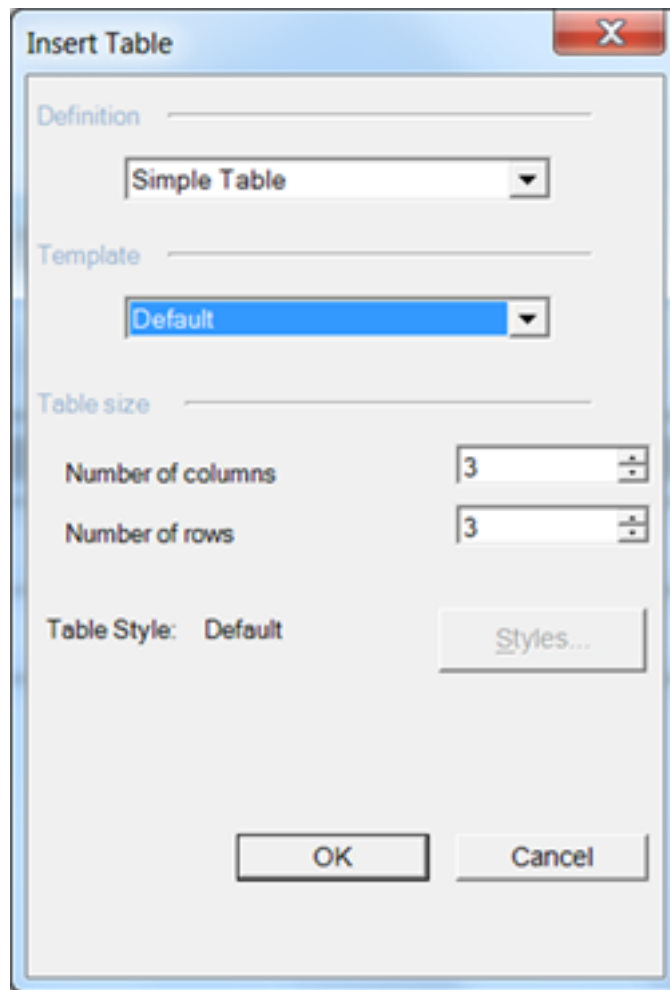


Figure 14-57: Insert Table dialog displays the current Table Style

In the following example, the **Insert Table** dialog displays “No Style Selected”. The user can click **Styles** and in the **Table Style** dialog select from the multiple styles that are available.

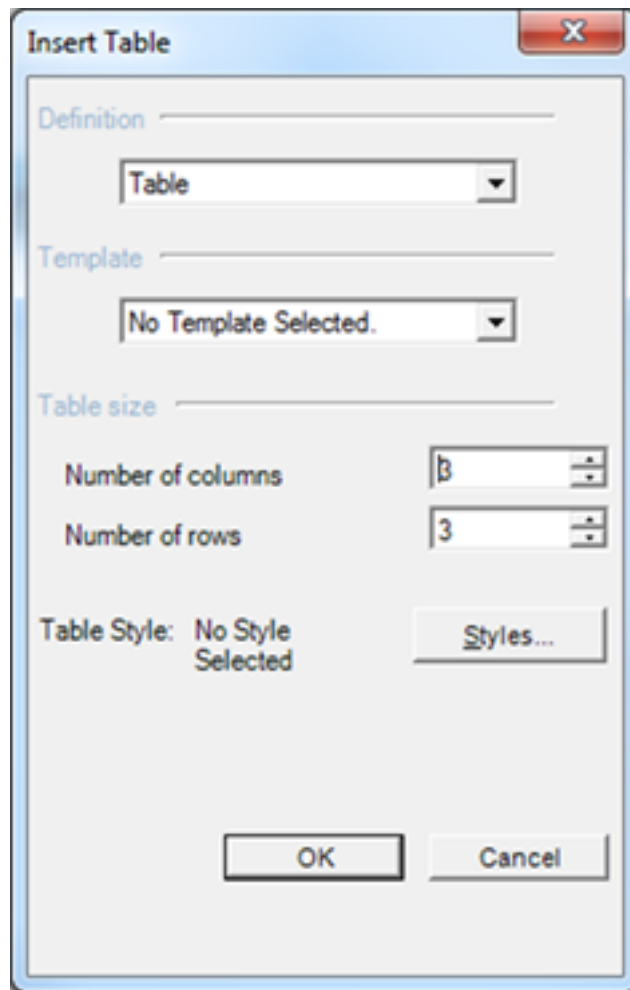


Figure 14-58: Insert Table dialog with no Table Style selected.

The **Table Style** dialog displays the names of all table styles associated with the current table definition.

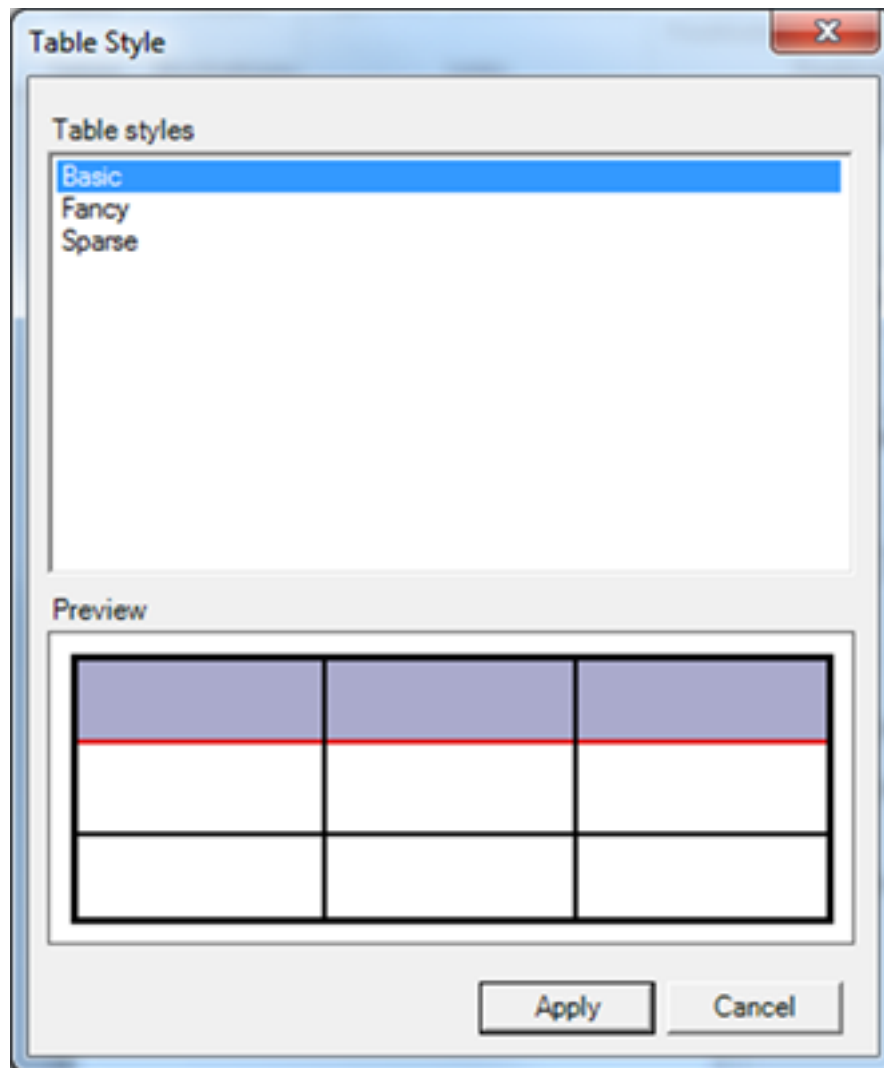


Figure 14-59: Table Style dialog displays all styles available

Table

Each <Table> node defines a collection of <Section> nodes and a general formatting scheme for a table. <Section> nodes define the appearance of heading and body sections.

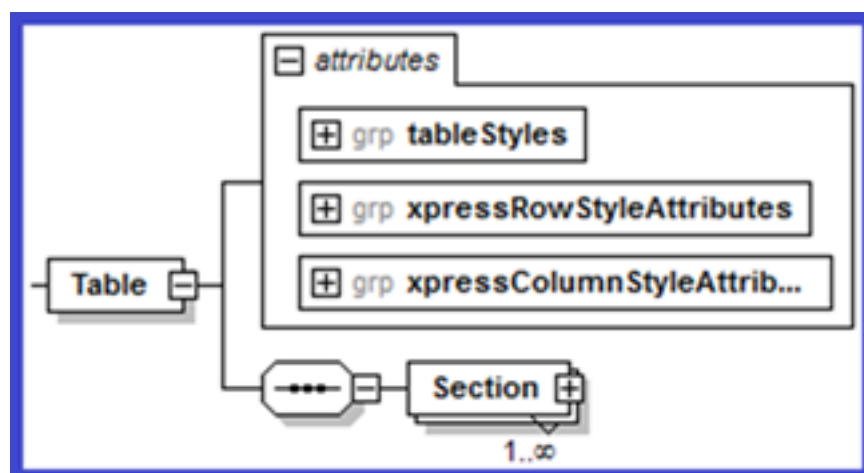


Figure 14-60: Configuration-XSD Table definition

To specify stylistic information for a <Table>, use <tableStyles>, <xpressColumnStyleAttributes> and <xpressRowStyleAttributes>.

See:

The following stylistic attributes may be applied to the <Table>. See Table for more information.

- Table 14-302: xpressColumnStyleAttributes group

The following stylistic attributes may be applied to both the <Table> and <Section>. See Table and Section, respectively.

Table 14-313: xpressRowStyleAttributes group

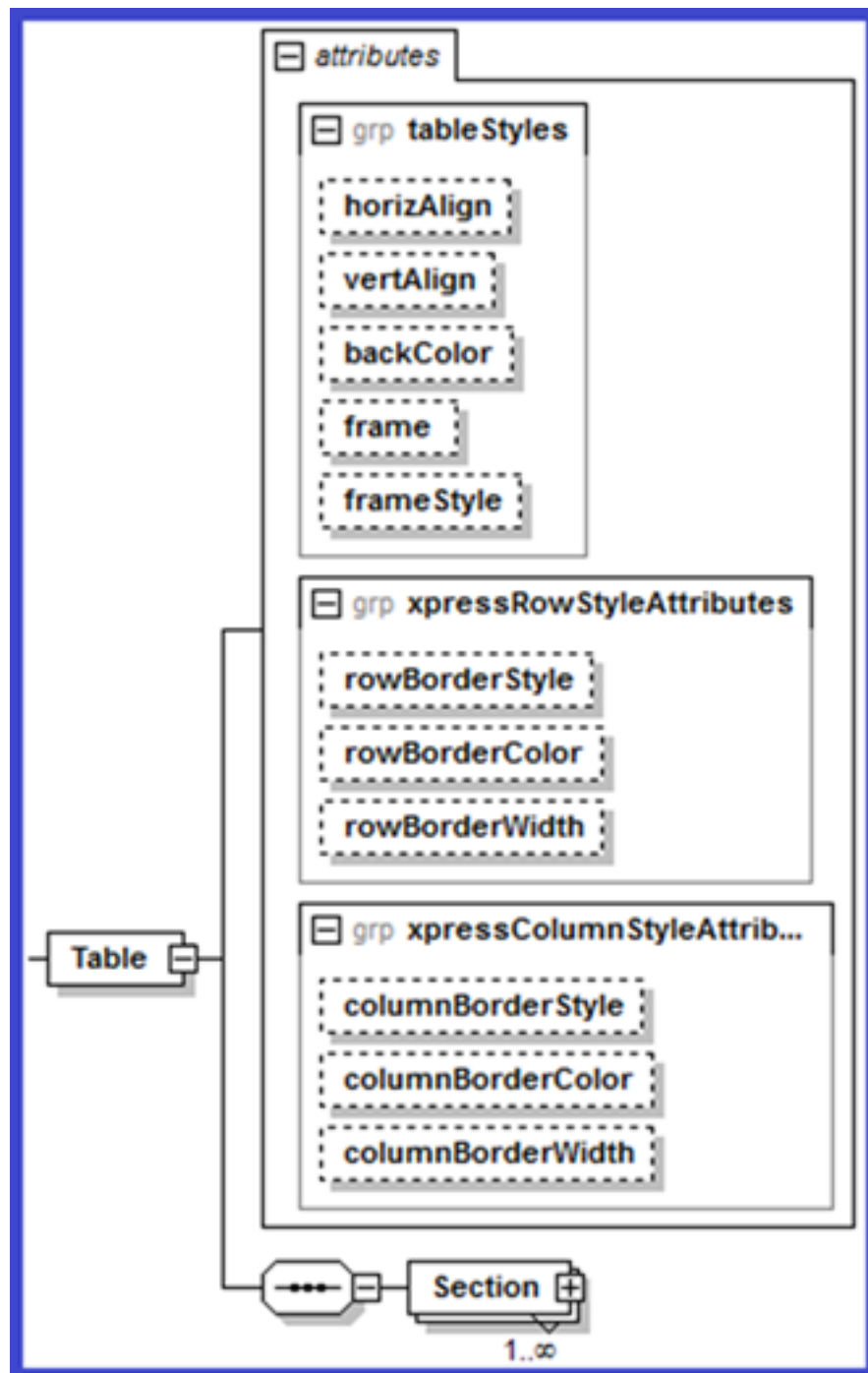


Figure 14-61: Configuration-XSD Table definition

Section

Each <Section> node defines a general formatting scheme for a section.

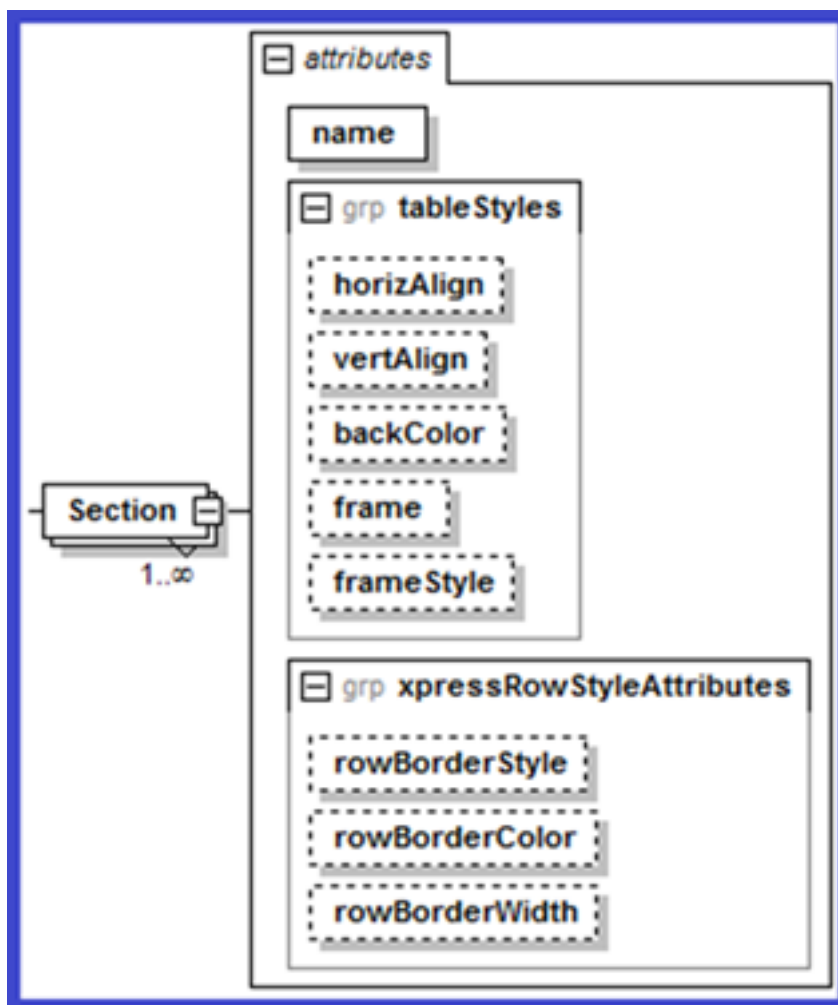


Figure 14-62: Configuration-XSD Section definition

To specify stylistic information for a <Section>, use <tableStyles> and <xpressRowStyleAttributes>.

See:

- Table 14-291: tableStyles attributes group
- Table 14-302: xpressColumnStyleAttributes group, Table 14-313: xpressRowStyleAttributes group

tableStyles attributes group

The following stylistic attributes may be applied to both the <Table> and <Section>. See Table and Section, respectively.

Tables may be configured so that the user can specify a background color. This can be configured at the desired level in the table structure. See **backColor**.

Tables may be configured so that the user can turn borders on and off. This can be configured at the desired level in the table structure. See **frame**.

Table 14-291: tableStyles attributes group

Attribute Name	Required	Definition
backColor	no	The color name to be applied to the background.
frame	no	One of six enumerated values that specify which, if any, of the table element's outer borders will be displayed. The enumerated values are: <ul style="list-style-type: none"> • all. Default. All borders will be displayed. • top. Only the top border will be displayed. • bottom. Only the bottom border will be displayed. • topbot. Only the top and bottom borders will be displayed. • sides. Only the left and right borders will be displayed. • none. No borders will be displayed.
frameStyle	no	Defaults to Single . The line type to be used for the table element's outer borders.
horizAlign	no	Defaults to left . The default horizontal alignment of cells within the given table element. Valid values are: left , right , center and justify .

xpressColumnStyleAttributes group

The following stylistic attributes may be applied to the <Table>. See Table for more information.

Table 14-302: xpressColumnStyleAttributes group

Attribute Name	Required	Definition
columnBorderColor	no	The color name to be applied to the column border. Defaults to 000000(=black).
columnBorderStyle	no	The line type to be used for the table's column borders. Defaults to Single . For valid values, see lineType in the XpressSchema.
columnBorderWidth	no	Integer. Specifies the width of the column

Attribute Name	Required	Definition
		border. Defaults to 1px .

xpressRowStyleAttributes

The following stylistic attributes may be applied to both the <Table> and <Section>. See Table and Section, respectively.

Table 14-313: xpressRowStyleAttributes group

Attribute Name	Required	Definition
rowBorderColor	no	The color name to be applied to the row border. Defaults to 000000(=black) .
rowBorderStyle	no	The line type to be used for the region's row borders. Defaults to Single . For valid values, see lineType in the XpressSchema.
rowBorderWidth	No	Integer. Specifies the width of the row border. Defaults to 1px .
vertAlign	no	Defaults to top . The default vertical alignment of the table. Valid values are: top , middle and bottom .

Templates

The <Templates> node contains one or more <Template> child elements that describe a table structure that may be inserted into the document. This node has one attribute: default, the value of which refers to the name of the <Template> child element that is to be used as the default table template.

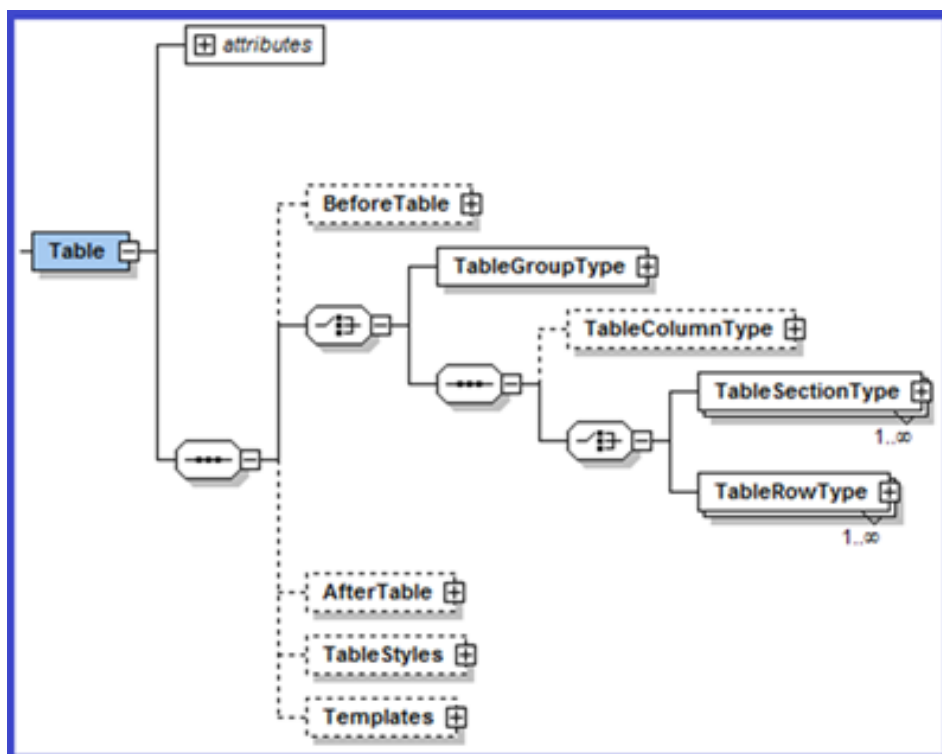


Figure 14-63: Configuration-XSD Table > Templates definition

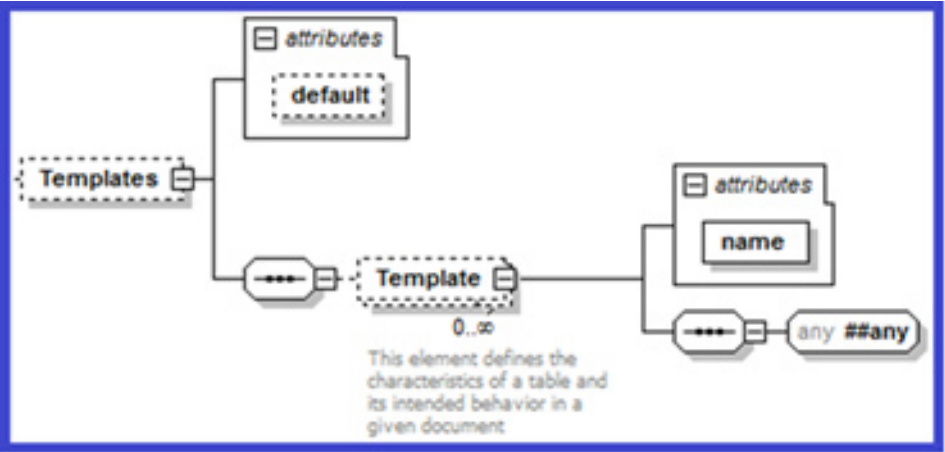


Figure 14-64: Configuration-XSD Templates definition

Table 14-324: Templates Attributes

Attribute Name	Definition
default	The name of the default template.

Template

Each <Template> node defines a table structure. The name attribute identifies the template within Quark XML Author as well as to the user, in the **Insert Table** dialog. In the following example, the **Inset Table** dialog shows that the “TemplateOne” Template that is associated with the “Table” table definition.

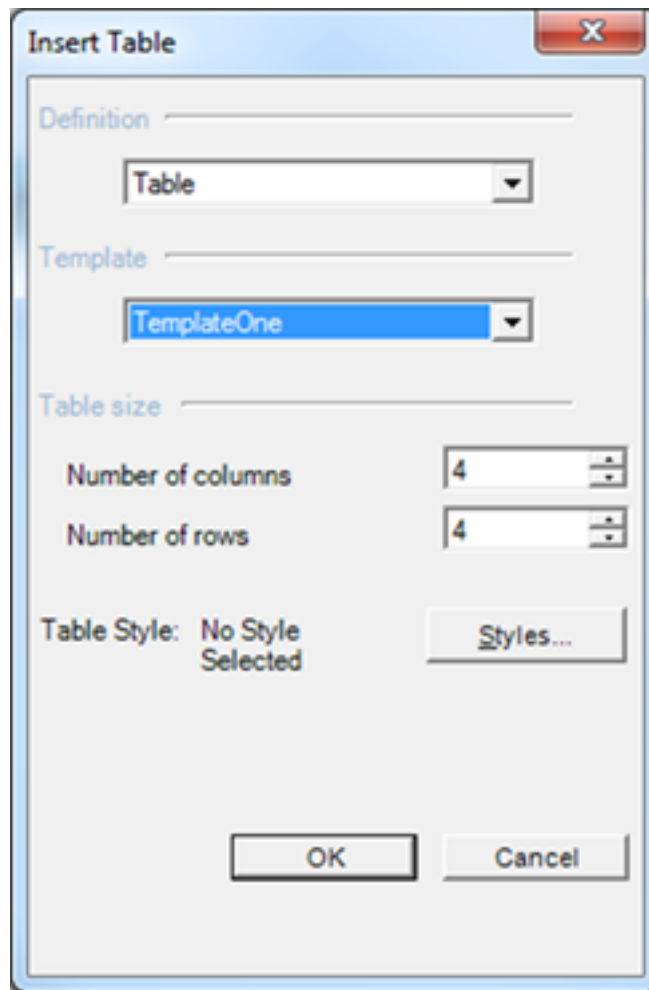


Figure 14-65: Insert Table dialog displays the selected table Template

The descendant nodes of <Template> define the structure of the table and the content types available.

Table 14-335: Template Attributes

Attribute Name	Definition
name	The name of the template.

In the following example, the table template “General” is defined. Within the collection of table templates, “General” is also specified as the default template.

```

<Templates default="General">
  <Template name="General">
    <table>
      <tgroup>
        <colspec colsep="1"/>
        <colspec colsep="1"/>
        <colspec colsep="1"/>
        <thead>
          <row rowsep="1">
            <entry/>
            <entry/>
            <entry/>
          </row>
        </thead>
        <tbody>
          <row rowsep="1">
            <entry/>
            <entry/>
            <entry/>
          </row>
          <row rowsep="1">
            <entry/>
            <entry/>
            <entry/>
          </row>
        </tbody>
      </tgroup>
    </table>
  </Template>
  <Template name="Sample">...</Template>
</Templates>

```

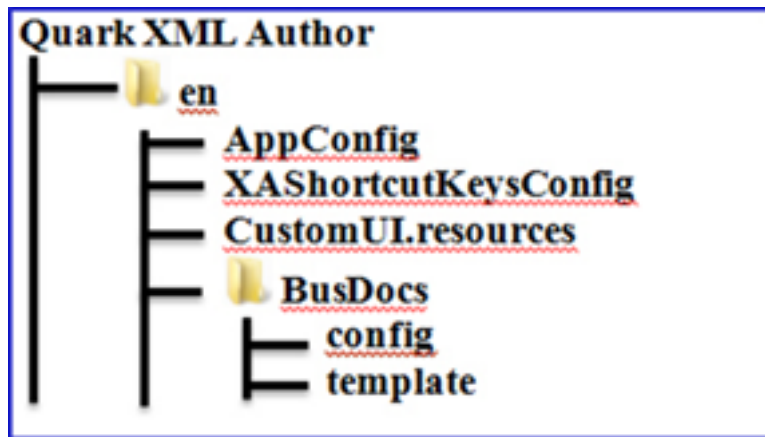
Figure 14-66: Templates defined

Tab and type behavior

The system can be configured so that when the user tabs to a table cell and then types (or pastes text), an element is automatically inserted. The new element receives the text. Quark recommends using this feature. See “Table 14-278: TableCell Attributes” for more information on specifying a **defaultElement** for the TableCell.

Support for language-specific configurations

At startup, Quark XML Author uses the current Microsoft Office Display Language to determine which language to use for authoring sessions. Quark XML Author then loads the appropriate language-specific configuration. To minimize the number of hard coded strings in the configuration files and software, content that should be translated has been centralized in standard Microsoft resource files. The directory structure is shown here:



UI controls and features

This section of the guide discusses features and the UI controls to which they are associated. For example, the **Find & Replace** feature is associated with a menu item, a toolbar button, a ribbon button and a shortcut key.

The text used to display a given UI control is retrieved from the resources file based on the name of the **InternalClass**. You can override this behavior by specifying attributes such as *label* and *supertip*.

For example, in the sample config node, the **InternalClass** is "Paste".

```
<InternalClass name="Paste" />
```

Values for the label, screentip, supertip and keytip components are retrieved from the resource file using the following naming convention for the resource id.

[InternalClass name]_[component name]

The following figure shows the fragment of the resource file for the components associated with the "Paste" InternalClass.

Name	Value
Paste_keytip	V
Paste_label	Paste
Paste_screentip	&Paste (Ctrl+V)
Paste_supertip	Paste the contents of the Clipboard.

Figure 14-67: installation folder comparison

Note that the label and keytip attributes are not used in the config node. Their values are retrieved from the Core resource file. *(These attributes are deprecated, but backward compatible.)* If used these attributes override the Core resource file.

Best Practice: Unless you need to override Core resource values, remove the attributes so that their values are retrieved from the Core resource file. In other words, use Core resources and remove uses of label, title, description, keytip, and screentip.

Best Practice: Id's should match the **InternalClass** name as closely as possible. For example,

id="PasteXA" should be id="Paste".

The naming convention is actually more robust than previously stated.

[InternalClass name][Type modifier][component name]

The resource id may require additional text to differentiate it when an **InternalClass** has multiple functions. For example, **InsertSectionBreak** takes the **breakType** parameter and is used to create several types of section breaks.

- InsertSectionBreakContinuous_label
- InsertSectionBreakNextPage_label

The following figure displays the "Paste" resources in the UI.

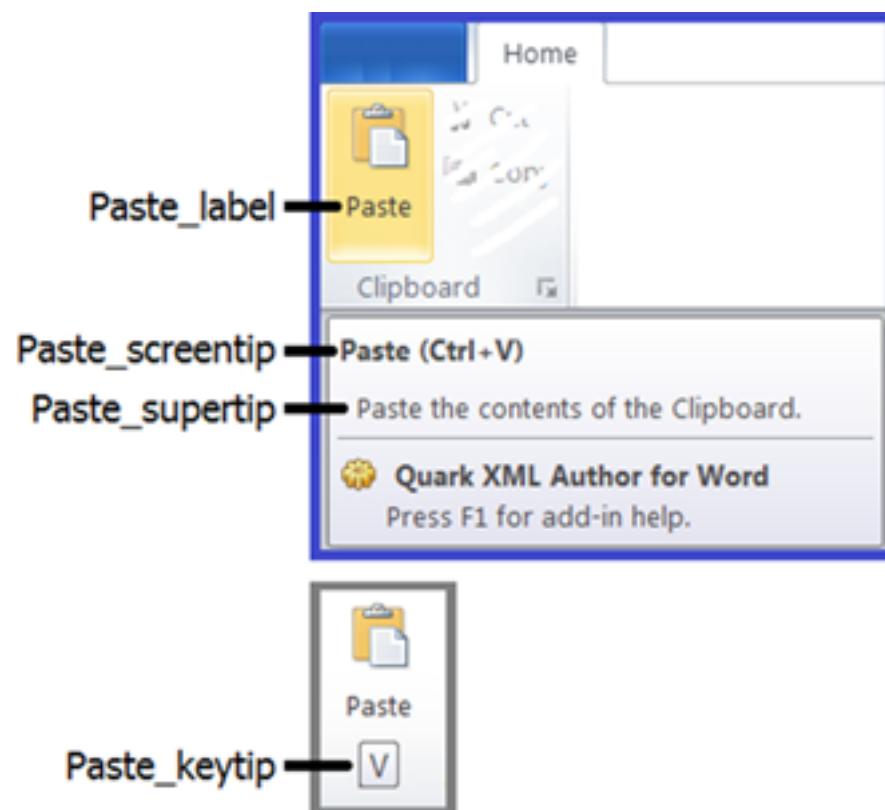


Figure 14-68: Resources needed for the Paste feature

```
<button id="Paste" imageMso="Paste" size="large" >
  <InternalClass name="Paste"/>
  <ShortcutKey key="V" shift="false" ctrl="true"/>
</button>
```

Icons

You can use one of the built in Word icons by setting the imageMso attribute to the id of that icon.

```
<button id="Paste" imageMso="Paste" size="large" >
  <InternalClass name="Paste"/>
  <ShortcutKey key="V" shift="false" ctrl="true"/>
</button>
```

If you need to specify a custom icon, refer to “Custom Icons”.

Context Menus

The Context Menu uses the new language architecture. The Spelling suggestions that are displayed are provided by standard Word for the current language culture. The list of insertable elements is derived from the Friendlies specified in the configuration.

Most of the commands displayed in the ContextMenu come from the following resource file.

ContextMenu.resx

These resources may not be overridden by professional services. These include strings like: Change To, Component, Cut, Edit Comment, Insert Cross-reference, etc.

Resource Files

Only Quark development may add resources to the core resources files. Core resources are compiled into dynamic link libraries (DLLs) that are distributed with the application.

Custom resources may be added by Quark professional services and partners to the custom resource files for (en) and (en-US). Or custom resources may be added to config nodes using attributes that override resource files.

Resource search order

The following describes the process used by the system to locate a specific resource. The system determines if a resource file is present and then looks for a resource in that file. There may be multiple resources available for each language. If the Display Language is English, the system searches resource files in the following order:

Dept. Responsible	Language + Locale	Source	Distributed
Professional Services	en-US		..\en-US\customUI.resources
Professional Services	en		..\en\customUI.resources

Dept. Responsible	Language + Locale	Source	Distributed
Development	en-US	coreUI.en-US.resx	..\en-US\XA.core.resources.DLL
Development	en	coreUI.en.resx	en\XA.core.resources.DLL
Development	(default)	coreUI.resx	XA.core.DLL

Compiled resources from Professional Services are standalone resource files. Compiled resources from Development are embedded in the appropriate DLL.

First, the system searches for custom resources specified as local attributes. This is done for each component that is needed in the current version of Word (label, screentip, keytip, etc.). If a component was not specified, then the system searches for custom resources provided by Professional Services. Files are located in the Quark XML Author installation folder or one its subfolders where noted. The system searches for a locale specific custom resource and if none is found then searches for a non-locale specific custom resource. If no custom resources are found the system searches for core resources provided by Development. The system searches for a locale specific core resource and if none is found then searches for a non-locale specific core resource. If none is found the system uses the default resource. The default resource is the fall back position and if this is not found then an exception is thrown. The example uses English, but if the Display Language were another language the resources would be searched in a similar pattern.

Resource Editor

Custom resource files can be edited by professional services with [.NET Resourcer](#). The download on the website is a CLR 4.0 compatible executable which should NOT be used for building the CustomUI.resources file. There is a CLR 2.0 specific version of this available in TFS (\$/XML Products/dev/Common/Tools/Resourcer/2.0).

Note: The Lutz resourcer displays culture=neutral for the English version of resource files. This is not a problem and can be ignored.

Custom Strings

To add a custom string, you must add a custom resource to the following resource file.

CustomUI.resources

The following figure shows the resource added for the Paste feature.

Name	Value
Paste_keytip	V
Paste_label	Paste
Paste_screentip	&Paste (Ctrl+V)
Paste_supertip	Paste the contents of the Clipboard.

Figure 14-69: String resources added to CustomUI.resources

Best Practice: For a new feature, add strings for all UI components:

- keytip
- label
- screentip
- supertip

Adding a custom string

To add a custom icon to resources:

1. In the resource editor, open CustomUI.resources.
2. Click **Edit > Insert Text**.
3. Set the name match the InternalClass as close as possible plus underscore and then the UI component.

For example, *Paste_label*.

1. Press Tab.
2. Specify the value of the string.
3. Click **File > Save**.

The string is added to the resource.

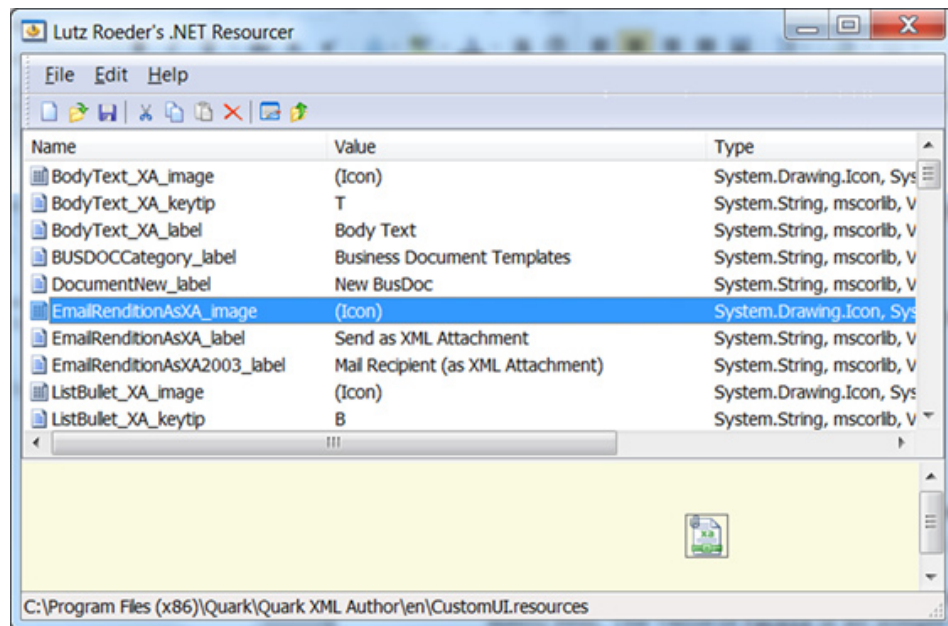
Custom Icons

The following section discusses how to incorporate custom icons.

Many icons displayed in Quark XML Author are the standard Word built in icons.

However, if you need to specify a custom icon for a Word ribbon button or another UI control, you must add the custom icon as a resource to the following icon resource file.

CustomUI.resources



```
<button id="DocumentOpen" image="DocumentOpen_image"
insertAfterMso="FileOpen" visible="true">
  <InternalClass name="DocumentOpen" filter="OpenXA2007QA"/>
</button>
```

Different from strings, the system will search for icons based on an exact match between the faceId value and the icon id/name in the CustomUI.resources file. In this example the system will search for “DocumentOpen_image”.

But be aware that the icon resource file from which the icon is retrieved will be the English specific version of the file.

Adding a custom icon

To add a custom icon to resources:

1. In the resource editor, open CustomUI.resources.
2. Click **Edit > Insert Files**.
3. Navigate to the folder containing the desired icon file.
4. Select the icon file.
5. Click **Open**.
6. Select the icon.
7. Press **F2**.
8. Set the name of the icon to match the InternalClass.

For example, *DocumentOpen_image*.

1. Click **File > Save**.

The icon is added to the resource.

Untouchables

There are some resources that may not be changed.

For example, the string “Empty” (“Vacio” in Spanish).

Modifying file filter resource strings

Filter string syntax

The resource ID identifies a string list of filter types. Each filter type comprises two parts: the filter text and the filter definition, which are separated by the pipe character (|). For example, to specify a filter to create a html rendition, you would use the pair shown below as the value of the “HTML Rendition” resource ID:

HTML Rendition = HTML Document (*.html)|*.html

A *filter type* is comprised of two parts:

- filter UI label
- filter definition

In a string that represents a filter type, the *filter UI label* and *filter definition* are separated by a pipe (|) character.

For example,

HTML Document (*.html)|*.html

Filter types may contain one or more file extensions. Extensions are separated by a semicolon character. These are the file extensions that Windows will search for and display in the dialog or form.

For example,

Tag Image File Format (*.tif; *.tiff) |*.tif;*.tiff

The Open and Save dialogs contain a list box of filters types and therefore expect you to provide this list in a string. Filter types are separated by a pipe (|) character.

The following is an example of a string containing two filter types. Typically these strings are stored as string resources.

Graphics Interchange Format (*.gif)|*.gif|Tag Image File Format (*.tif; *.tiff)|*.tif;*.tiff

Best practice

For the various Open and Save operations, strings that contain lists of filter types are stored as resource strings. Resource string exists in XML Author’s embedded CoreUI resource table and also is stored in the CustomUI.resources file.

When modifying one of these string resources, care must be taken to preserve proper formatting. These strings can be difficult to read as a single long flowing line, therefore it is recommended that you restructure a string into its components with one component per line. The following procedure describes the best way to do this.

Launch a text editor that you can use to edit the string.

Open the CustomUI.resources file using resourcer.

Locate and copy the desired file filter resource.

Paste the resource string into the text editor.

[ORIGINAL STRING]

All Pictures

```
(*.*;*.wmf;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;)|*.emf;*.wmf;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;|Windows Enhanced Metafile (*.emf)|*.emf|Windows Metafile (*.wmf)|*.wmf|JPEG File Interchange Format (*.jpg; *.jpeg; *.jif; *.jpe)|*.jpg;*.jpeg;*.jif;*.jpe|Portable Network Graphics (*.png)|*.png|Windows Bitmap (*.bmp; *.dib; *.rle;)|*.bmp;*.dib;*.rle|Graphics Interchange Format (*.gif)|*.gif|Tag Image File Format (*.tif; *.tiff)|*.tif;*.tiff
```

For each pipe '|' character, place the cursor to the left of the pipe character and press Enter so that each component is on a separate line.

For example,

[INITIAL FORMATTING]

All Pictures

```
(*.*;*.wmf;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;)
```

```
|*.emf;*.wmf;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;
```

```
|Windows Enhanced Metafile (*.emf)
```

```
|*.emf
```

```
|Windows Metafile (*.wmf)
```

```
|*.wmf
```

```
|JPEG File Interchange Format (*.jpg; *.jpeg; *.jif; *.jpe)
```

```
|*.jpg;*.jpeg;*.jif;*.jpe
```

```
|Portable Network Graphics (*.png)
```

```
|*.png
```

```
|Windows Bitmap (*.bmp; *.dib; *.rle;)
```

```
|*.bmp;*.dib;*.rle
```

```
|Graphics Interchange Format (*.gif)
```

```
|*.gif
```

```
|Tag Image File Format (*.tif; *.tiff)
```

```
|*.tif;*.tiff
```

In the next formatting step we will align the filter elements to clearly show how each filter is constructed.

[COMPLETED FORMATTING]

All Pictures

```
(*.*;*.wmf;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;)
```

```
    |*.emf;*.wmf;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;;
```

```
|Windows Enhanced Metafile (*.emf)
```

```

|*.emf
|Windows Metafile (*.wmf)
|*.wmf
|JPEG File Interchange Format (*.jpg; *.jpeg; *.jif; *.jpe)
|*.jpg;*.jpeg;*.jif;*.jpe
|Portable Network Graphics (*.png)
|*.png
|Windows Bitmap (*.bmp; *.dib; *.rle;)
|*.bmp;*.dib;*.rle
|Graphics Interchange Format (*.gif)
|*.gif
|Tag Image File Format (*.tif; *.tiff)
|*.tif;*.tiff

```

Removing support for a file type:

In the following example, let's remove support for Tag Image File Format (*.tif, *.tiff) files.

Remove the individual 'Encapsulated PostScript' filter.

DELETE THIS LINE | Tag Image File Format (*.tif; *.tiff)

DELETE THIS LINE |*.tif;*.tiff

Remove the tif file specification from the 'All Pictures' filter.

EDIT THIS LINE All Pictures
(*.emf;*.wmf;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;)

EDIT THIS LINE
|*.emf;*.wmf;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;

Adding support for a file type:

In the following example, let's add support for compressed Windows Metafiles (*.EMZ and *.WMZ)

Add entries for EMZ and WMZ files to the 'All Pictures' filter.

EDIT THIS LINE All Pictures
(*.emf;*.emz;*.wmf;*.wmz;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;)

EDIT THIS LINE
|*.emf;*.emz;*.wmf;*.wmz;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;
;

Add entries to each of the Windows Metafile filters.

EDIT THIS LINE |Windows Enhanced Metafile (*.emf;*.emz;)

EDIT THIS LINE |*.emf;*.emz;

EDIT THIS LINE |Windows Metafile (*.wmf;*.wmz;*)

EDIT THIS LINE |*.wmf;*.wmz;*

After adding and/or editing each of the appropriate filters, examine the results. Note that the file specifications are still aligned with their corresponding filter descriptions. We are now ready to restructure the components into single filter string by deleting all of the carriage returns that we added.

[POST EDIT RESULTS]

All Pictures

(*.*emf;*.emz;*.wmf;*.wmz;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;*)

|*.emf;*.emz;*.wmf;*.wmz;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;
;

|Windows Enhanced Metafile (*.emf;*.emz;)

|*.emf;*.emz;

|Windows Metafile (*.wmf;*.wmz;*)

|*.wmf;*.wmz;*

|JPEG File Interchange Format (*.jpg; *.jpeg; *.jif; *.jpe)

|*.jpg;*.jpeg;*.jif;*.jpe

|Portable Network Graphics (*.png)

|*.png

|Windows Bitmap (*.bmp; *.dib; *.rle;)

|*.bmp;*.dib;*.rle

|Graphics Interchange Format (*.gif)

|*.gif

|Tag Image File Format (*.tif; *.tiff)

|*.tif;*.tiff

Left align each filter element and remove blank lines as shown here.

[LEFT ALIGNED]

All Pictures

(*.*emf;*.emz;*.wmf;*.wmz;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;*)

|*.emf;*.emz;*.wmf;*.wmz;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff;
;

|Windows Enhanced Metafile (*.emf;*.emz;)

|*.emf;*.emz;

```
|Windows Metafile (*.wmf;*.wmz;*)
|.wmf;*.wmz;*
|JPEG File Interchange Format (*.jpg; *.jpeg; *.jif; *.jpe)
|.jpg;*.jpeg;*.jif;*.jpe
|Portable Network Graphics (*.png)
|.png
|Windows Bitmap (*.bmp; *.dib; *.rle;)
|.bmp;*.dib;*.rle
|Graphics Interchange Format (*.gif)
|.gif
|Tag Image File Format (*.tif; *.tiff)
|.tif;*.tiff
```

Join all of the components into a single line. Make sure that all but the very first line have a pipe '|' character at the beginning of the line prior to joining. Note that the following line may be wrapped, but is a single line of text.

[REASSEMBLED FILTER STRING]

All Pictures

```
(*.*;*.wmf;*.wmz;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff)|*.wmf;*.wmz;*.jpg;*.jpeg;*.jif;*.jpe;*.png;*.bmp;*.dib;*.rle;*.gif;*.tif;*.tiff|Windows Enhanced Metafile (*.emf;*.emz;)|*.emf;*.emz;|Windows Metafile (*.wmf;*.wmz;)|*.wmf;*.wmz;|JPEG File Interchange Format (*.jpg; *.jpeg; *.jif; *.jpe)|*.jpg;*.jpeg;*.jif;*.jpe|Portable Network Graphics (*.png)|*.png|Windows Bitmap (*.bmp; *.dib; *.rle;)|*.bmp;*.dib;*.rle|Graphics Interchange Format (*.gif)|*.gif|Tag Image File Format (*.tif; *.tiff)|*.tif;*.tiff
```

The final step is to update the resource file.

copy the edited file filter string in the text editor and paste in back to CustomUI.resources and save the resource file.

Extensibility Methods

With some elements, you may want to provide a way for users to access a tool such as a metadata wizard via a command in the context menu. An element definition may use an extensibility method to provide this functionality.

The syntax for using Extensibility Methods is shown below:

```
<ExtensibilityMethods>
  <ExtensibilityMethod id="method name" friendly="friendly name"
showInComponentContextMenu="true or false"
showInContextMenu="true or false" faceID="#" />
</ExtensibilityMethods>
```

Table 14-346: ExtensibilityMethod Attributes

Attribute Name	Required	Definition
enableXpath	no	<p>Value is a Boolean XPath expression which evaluates to a node-set in which false is returned for an empty node-set and true otherwise. When applied to the runtime element, indicates whether to enable the method on the Context Menu.</p> <p>Namespaces are not supported in the xpath syntax.</p> <p>For the Ribbon, enableXpath on evaluates against root notes. Against current node is not supported.</p>
faceID	no	<p>Allows you to specify an icon that will be displayed next to the context menu item. The value of faceId is an integer corresponding to the ID number of an icon in the Word template FaceID.dot.</p>
friendly	no	<p>Specifies a name for the context menu command.</p>
id	yes	<p>Specifies the ID of the EI method to be called.</p>
showInComponentContextMenu	No	<p>Defaults to true. When set to false, the command will not appear in the Component context menu.</p>
showInContextMenu	No	<p>Defaults to true. When set to false, the command will not appear in the main context menu, but may still appear in the Component context menu if showInComponentContextMenu is set to true.</p>
showXPath	no	<p>Value is an XPath expression which evaluates to a node-set, and when applied to the current element, indicates whether to include the method on the context menu: if a node-set is returned, the method is included; if an</p>

Attribute Name	Required	Definition
		<p>empty node-set is returned, the method is omitted. When applied to the runtime element, indicates whether to include the method on the Context Menu. The XPath expression is applied to the XOM therefore the names are XOM element names. The XPath expression will be evaluated against the current runtime element and thus must be relative to it.</p> <p>Namespaces are not supported in the xpath syntax. For example, <code>showXPath="self::node()[local-name()='Section']"</code>.</p> <p>XML Author does not support dynamic show/hide of Ribbon items, therefore showXPath is not applicable for Ribbon items.</p>

See Section 6 for more information on how to configure Extensibility Interface methods.

Starter Documents and Templates

For each document class you define, a “Starter Document” must be created that contains the basic root element of the document. We call them “Starter Document” rather than “template” to distinguish these files from the Word Templates that must also be created for each document class. Both files are discussed in this chapter.

This must be done for each language culture in which you are creating solutions.

Starter Document

The Starter Document is an XML document, and so must begin with an XML declaration. The XML declaration may also be followed by a Processing Instruction (PI) node that identifies values for certain items of document metadata.

The PI node must be named **Xpress**, and takes the following format:

```
<?Xpress attribute="value" attribute2="value2"?>
```

It would, of course, have as many or as few attributes as your organization’s needs require. Later, as the document is edited and revised, your content management system may insert additional attributes. Quark XML Author will retain whatever attributes and values are placed in this node, but will only look for and process the value of one attribute: **AccessMode**. If this attribute is not present, the **AccessMode** will default to **Author**, but if the **AccessMode** is present, Quark XML Author will open the document in whatever mode is specified by its value.

Quark XML Author provides two built-in PI attribute: **productLine** and **accessMode**. Both are optional. The first can be used to identify the document class, and its value must match the value of a **productLine** attribute in an **AppConfig** file namespace node. The latter, **accessMode**, would define the level of access for the document as a whole. See section 5.54 for more information on access modes.

Document Content

The Starter Document’s root node is the root node defined in the XAS. Within that root node, any required elements must also be defined, and any of their required child elements must also be defined, and so on. For example, if a **Chapter** root node required a **Title** and at least one **Section**, which in turn must contain at least one **Paragraph**, you would need to define all four of these elements, and the starter document might look something like the example shown below.

```
<?xml version="1.0" encoding="UTF-8" ?>
<Chapter>
  <Title>Your Title Here</Title>
  <Section>
```

```
<Paragraph>Default first paragraph</Paragraph>
</Section>
</Chapter>
```

Of course, this is a very simple structure that doesn't take into account attributes—any mandatory attributes for these elements would also need to be added, with default values supplied.

Namespace

In addition to any attributes that may be required by the schema, the root element may also have default or prefix namespace attributes.

DOCTYPE declarations

Quark XML Author provides the following support for DTD DOCTYPE declarations within Quark XML Author documents.

External

Externally referenced DOCTYPEs are fully supported. We support addition of an externally referenced doctypes in documents not for validation purposes. There are 2 reasons for including DocType declarations in a XML document from a XML Author perspective:

1. Using entity declarations that might be declared in those DTDs.
2. Using default values for attributes that might be used in those DTDs

Internal

Quark XML Author supports a subset of the full language specification for internal DOCTYPE declarations.

A declaration must be valid in terms of XML syntax.

Comments are not allowed within the DOCTYPE declaration.

Only the !ENTITY definition is supported. Definitions such as !ELEMENT or !ATTLIST are not supported.

If Quark XML Author encounters an unsupported DOCTYPE declaration, the system will prompt the user with error message stating that the document cannot be opened.

The Quark XML Author parsing logic does support additional aspects of DOCTYPE declaration. If you have a question about whether or not your declaration is supported, please contact technical support.

Microsoft Word Templates

In Word, the ribbon interface is defined by the <Ribbon> node in the AppConfig and DocConfig files, rather than via Word templates. The global template is not used. The document-level template (.dotx file) is necessary to contain paragraph

styles that the document class will use, but no special tool is needed to create this Word template.

If your **Styles** button is not available, see “Enabling the Styles button in standard Word”.

Preparing to Use TemplateManager

For each culture, several things must be done before using TemplateManager:

- Prepare AppConfig.xml. Menus and CommandBars defined in AppConfig.xml will be used to create Xpress.dot.
- Prepare a DocConfig file for each document class. Menus and CommandBars defined in the DocConfig files will be used to create the document class Word templates.
- Close all instances of Word running on the computer. If Outlook is configured to use Word as the email editor, you must also close Outlook.
- “Removing temporary files”
- Place TemplateManager.exe and its support files in the Quark XML Author application folder. The support files are:
 - “CollectControls.exe”
 - Quark.XA.TemplateUtil.exe
 - Quark.XA.TemplateUtilLib.dll
- Set the Microsoft Office culture to the culture of the configuration.
- “To Launch TemplateManager”
- “Removing temporary files again”

Removing temporary files

To ensure that all temporary template information is removed:

IMPLEMENTER:

Close all winword processes

Go to %AppData%\Microsoft\Templates\

Del Normal.dot

Del ~*.dot

Close all winword processes

Go to Application installation folder\

Del Wxyz.dot

Del ~xyz.dot

END USER:

Close all winword processes

Go to %AppData%\Microsoft\Templates\

Del Normal.dot

Del ~*.dot

Go to ... \Program Files\Microsoft Office\Templates\{folder#}

Where {folder#} equals the desired language:

- 1033 = English
- 3082 = Spanish
- 1041 = Japanese
- 1036 = French

Del ~*.dot

Go to ... \Program Files\Quark XML Author\{language}

Del ~*.dot /s

Go to ... \Program Files\Quark XML Author\{language}\{solution name (i.e. BUSDOCS)}

Del ~*.dot /s

If you are upgrading from a previous version of Quark XML Author, you may also want to verify that Normal.dot does not include legacy, unused menu items. If it does, rename Normal.dot and then open and close Word so that it creates a default Normal.dot file.

Once you have performed all these tasks, you are ready to run TemplateManager and create the Quark XML Author templates for a given culture. Afterwards, be sure to “Removing temporary files” as the final step.

CollectControls.exe

This utility is used to find the ControlIds for various functions in Word for use in the docConfig.

1. Specify the function in **Filter**.
2. Click **Show Ids**.

The IDs for the specified function are displayed.

For example, for “print”, the following figure shows the IDs available. If you wanted Print on the File menu, the ID is 4.

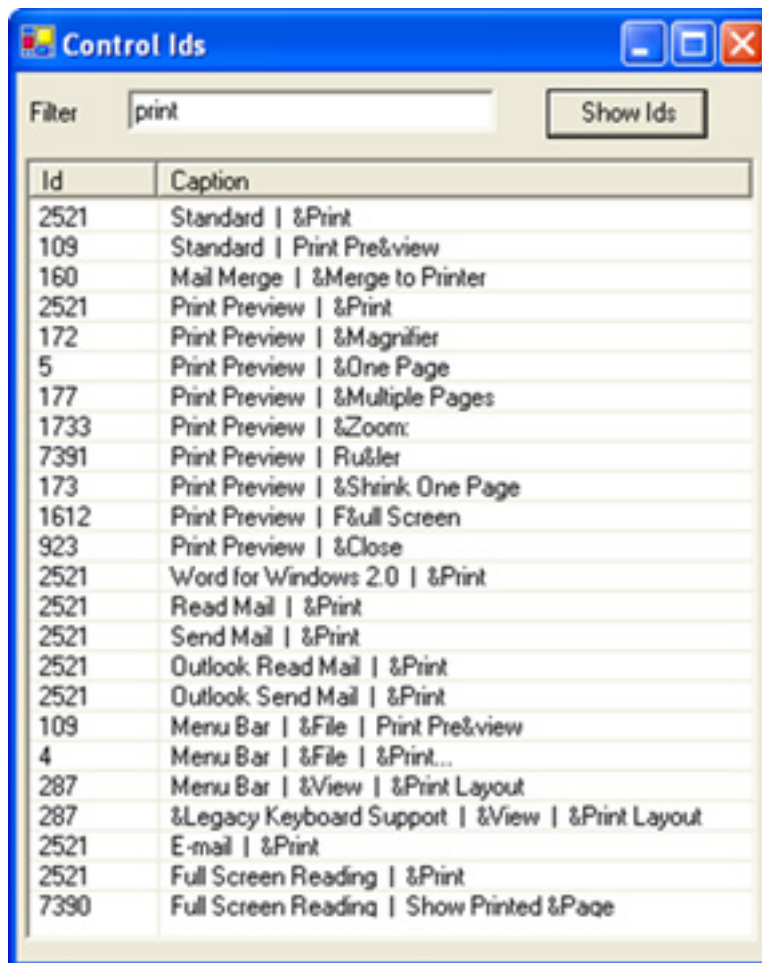


Figure 15-1: Collect Controls utility

To Launch TemplateManager

To launch TemplateManager:

1. Set the Microsoft Office culture to the culture of the configuration.
2. Right-click on TemplateManager.exe. In the shortcut menu, click **Run as administrator**.
3. In the Select Language dialog, select the language of the culture and configuration.
4. Click OK. The XML Author Template Utility application displays.
5. Select the **Application Config** tab.

Application Config Tab

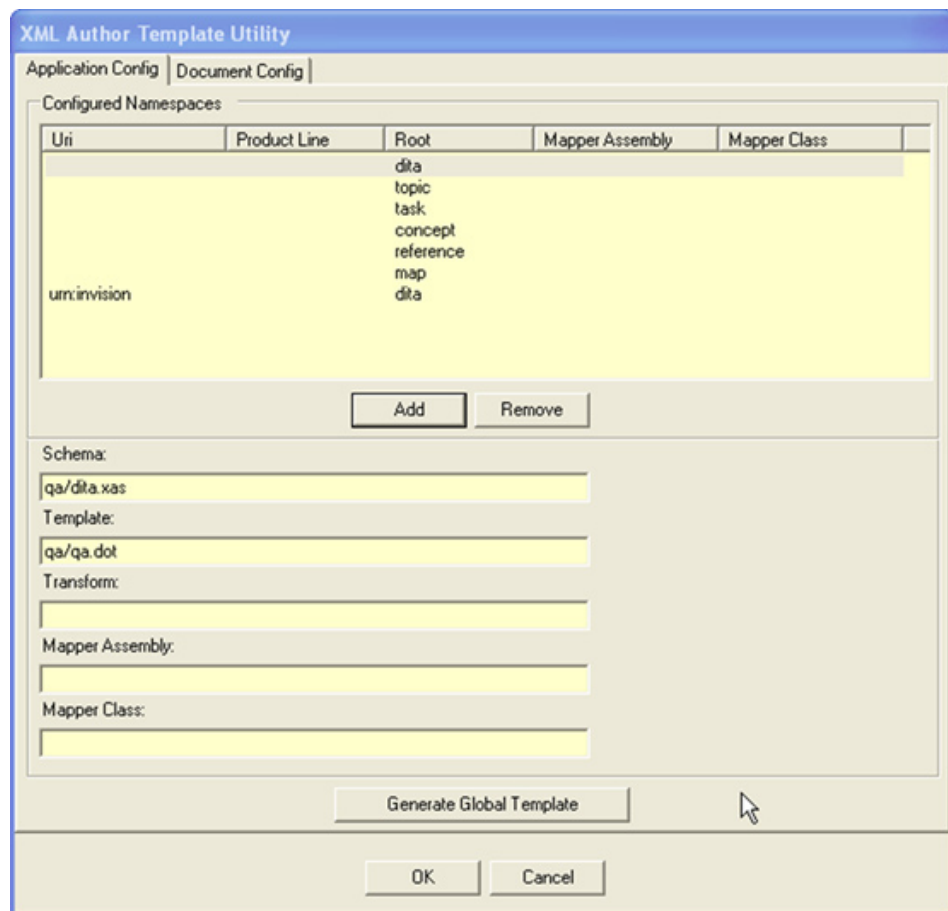


Figure 15-2: TemplateManager Application Configuration Window

The Application Config tab is displayed. The Configured Namespaces list is populated with each pre-configured namespace in the AppConfig Namespaces node

Edit Pre-Configured Namespace

To edit a pre-configured namespace, select it in the list. Make any changes necessary in the following fields: Schema, Template, Transform, Mapper Assembly, and Mapper Class. Click **OK** to commit the changes to the AppConfig file.

Add a Namespace

You can add new namespace configurations to the AppConfig file on the Application Config tab. Click **Add** to display the Add Namespace window (Figure 15-3).

The image shows a dialog box titled "Add Namespace" with a blue header bar. It contains ten text input fields, each with a label to its left: "Uri", "Product Line", "Root", "Config", "Schema", "Template", "Transform", "Mapper Assembly", and "Mapper Class". A mouse cursor is pointing at the "Schema" field. At the bottom of the dialog are two buttons: "OK" and "Cancel".

Figure 15-3: Add Namespace Window

Complete the form and click **OK**. Changes are not be committed to the AppConfig file until you click **OK** on the main window.

Remove a Namespace

To remove a namespace, select it in the list and click **Remove**. A confirmation message will be displayed. Click **Yes** to delete the namespace. Click **No** to cancel the operation.

Create Xpress.dot

Once you have made any namespace changes necessary, click **Generate Global Template**. A window will be displayed that lists the changes being made (Figure 15-4) to Normal.dot and Xpress.dot will be created. Word will open with the Xpress.dot template active. Click **OK** to save the template and close the Activity window. If are editing Xpress.dot, you will have to close the template manually; otherwise, the template closes automatically.

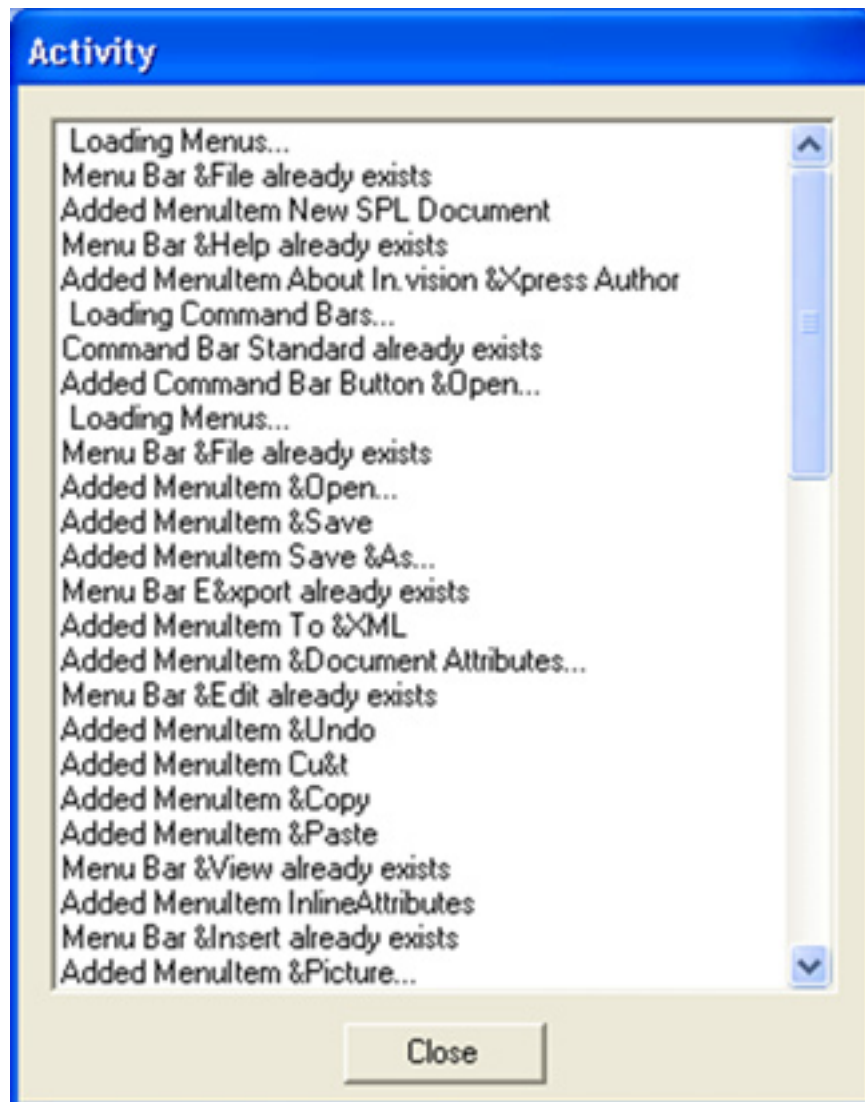


Figure 15-4: Template Under Construction

New and reprogrammed menu items and toolbar buttons specified in AppConfig will be placed at the bottom of menus and the far right of toolbars. Open Xpress.dot in Word, select Tools > Configure, and move these items to the proper locations. For example, if you have replaced Word's Open functionality with Quark XML Author's functionality, you'll need to move the Open button on the Standard toolbar back to its customary place. When done, save and close Xpress.dot

Document Config Tab

Use the Document Config tab to create and modify document class templates. To work with a document class, select the appropriate namespace in the Configured Namespace list on the Application Config tab, then click the Document Config tab.

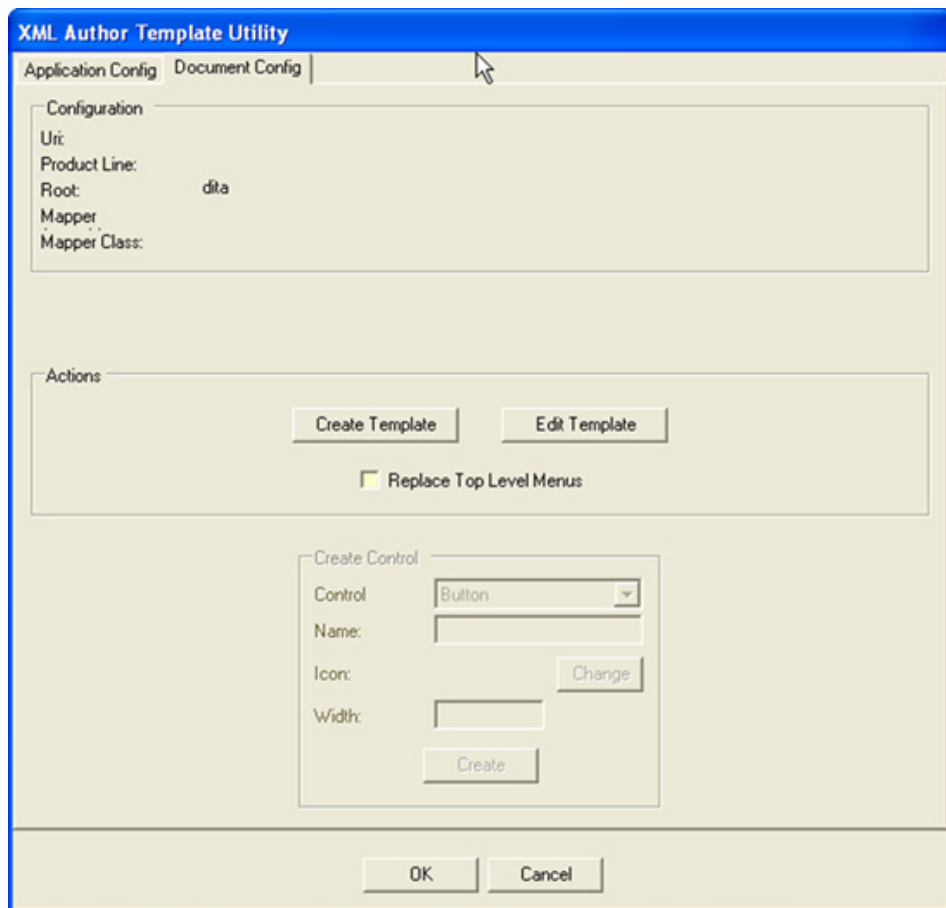


Figure 15-5: Document Config Tab

Create Template

To create the document template, check the Replace Top Level Menus checkbox and click Create Template. The template will be created with the name specified in the namespace declaration and will be placed in the directory specified. Word will open the template. Save and close the template.

Edit Template

To edit an existing template, click Edit Template. The template will open. Use the Create Control panel to create new controls. Make sure the Name matches the resourceId name for the control in the configuration file. When you click Create, the new control will be added to a temporary toolbar. Move it to the proper location before saving and closing the template.

Final Step

Remove temporary files again. See “Removing temporary files”.

Setting the Proofing Language

You can set the default proofing language of the XML Author template to account for different cultures. For example, you may need to change the language from US English to United Kingdom English.

1. Open the template in a package editor (for example oXygen), or you can rename the template to .zip and extract the file you want to change.
2. Open word/styles.xml
3. Change the lang@val to the desired language. For example, in the figure below, set lang@val to “en-GB”.



Figure 15-6: Setting the Proofing Language

Word: New templates

For Word solutions, when you create a new template, extraneous style information causes an issue in Quark XML Author smart paste feature. This causes the paste feature, to use global styling information instead of local styling information.

To remove the style information:

1. Use the Word **Save As** feature to save the .dotx Word Template file as a .dot Word 2003 Template file.
2. Use the Word **Save As** feature to save the .dot Word 2003 Template file back to a .dotx Word Template file.

The extraneous style information is removed from the template.

Technical details

This section provides the technical details on the “new template” issue described in the preceding section.

This issue is happening because there is an overriding default paragraph property being applied at the document level. The following is a snippet of the styling node from a sample dotx:

```

<pkg:part pkg:name="/word/styles.xml"
pkg:contentType="application/vnd.openxmlformats-
officedocument.wordprocessingml.styles+xml">
<pkg:xmlData>
  <w:styles
xmlns:r="http://schemas.openxmlformats.org/officeDocument/2006/re
lationships"
xmlns:w="http://schemas.openxmlformats.org/wordprocessingml/2006/
main">
    <w:docDefaults>
      <w:rPrDefault>
        <w:rPr>
          <w:rFonts w:asciiTheme="minorHAnsi"
w:hAnsiTheme="minorHAnsi" w:eastAsiaTheme="minorHAnsi"
w:cstheme="minorBidi" />
          <w:sz w:val="22" />
          <w:szCs w:val="22" />
          <w:lang w:val="en-US" w:eastAsia="en-
US" w:bidi="ar-SA" />
        </w:rPr>
      </w:rPrDefault>
      <w:pPrDefault>
        <w:pPr>
          <w:spacing w:after="200" w:line="276"
w:lineRule="auto" />
        </w:pPr>
      </w:pPrDefault>
    </w:docDefaults>
    <w:latentStyles w:defLockedState="0"
w:defUIPriority="99" w:defSemiHidden="1" w:defUnhideWhenUsed="1"
w:defQFormat="0" w:count="267">

```

A default paragraph property is being applied to any paragraph that doesn't have a pPr.

The following explanation is from the OpenXML SDK documentation:

docDefaults (Document Default Paragraph and Run Properties)

This element specifies the set of default paragraph and run properties which shall be applied to every paragraph and run in the current WordprocessingML document. These properties are applied first in the style hierarchy; therefore, they are superseded by any further conflicting formatting, but applied if no further formatting is present.

If this element is omitted, then the document defaults shall be application defined by the hosting application.

Consider the following definition for the document defaults for a WordprocessingML document:

```

<w:docDefaults>
  <w:pPrDefault>
    <w:pPr>
      <w:jc w:val="center"/>
    </w:pPr>
  </w:pPrDefault>
  <w:rPrDefault>
    <w:rPr>
      <w:b/>
    </w:rPr>
  </w:rPrDefault>
</w:docDefaults>

```

The child elements of **docDefaults** specify a default paragraph property of centered text and a default run property of bold text. Consider what happens when this formatting is applied to the following fragment from the main document part of the same document:

```
<w:body>
  <w:p>
    <w:r>
      <w:t>Hello world!</w:t>
    </w:r>
  </w:p>
</w:body>
```

This paragraph contains no formatting properties; therefore, using the style hierarchy, the document default paragraph and run properties are applied as specified within the **docDefaults** element, and the resulting paragraph is centered as specified in the **jc** element as well as bold as specified via the **b** element.

Here is an example of the OpenXML that we paste for an intra-para paste:

```
<w:p>
  <w:r>
    <w:rPr />
    <w:t xml:space="preserve">two </w:t>
  </w:r>
</w:p>
```

The OpenXML does not contain a pPr node. This is done so that the content being pasted “inherits” the styling of the target paragraph. But apparently having this default pPr in the docDefaults node overrides that and applies this global styling which is undesirable. The solution is to remove the highlighted pPr node from the dotx which is accomplished by the workaround described in the preceding section.

Converting Word 2003 configurations to the latest supported version

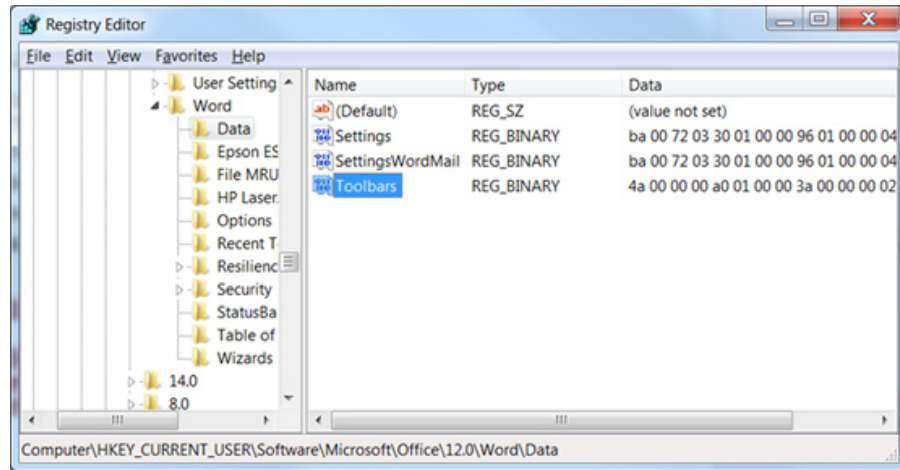
1. Converting Word 2003 configurations to the latest supported version

Enabling the Styles button in standard Word

Occasionally and for reasons unrelated to Quark XML Author, the standard Word Styles button can become unavailable.

To enable the **Styles** button:

1. Close all instances of Word if it is open.
2. Launch Regedit.
3. In Regedit, go to the following location:
 \HKEY_CURRENT_USER\Software\Microsoft\Office\12.0\Word\Data



4. At that location, delete “Toolbars”.
5. Close Regedit.
6. Open Word.

The **Styles** button is available.

Structured Authoring

This section explores structured authoring and constrained authoring and how the user interacts with a document, its structure, components and elements. Working with simple text may be discussed occasionally, but the primary focus is on working with elements.

Quark XML Author versions 4.0 and later allow for mixed content in table cells. Occasionally the behavior of actions upon a given content type are different depending on whether the element is located outside or inside of a table. These differences are noted where applicable.

Some content features act different depending on whether the content element is inside or outside of a table. These differences are discussed in the applicable sections.

Deleting Content Elements and Structure

Deleting content in a structured authoring tool can be significantly different than in Microsoft Word. Deleting simple text is the same in both. But when working with elements, specific behaviors and rules for delete are needed in order to ensure the integrity of an XML document and its structure.

Most content in a standard Word document is not required and therefore may be deleted. However, most XML schemas have some required content. A schema may define a parent element with a required child element that may not be deleted. Some content elements have required elements regardless of the schema. For example, a bulleted list requires at least one item and a table requires at least one row or column or cell. Therefore, the Component Cut, Remove and Delete features are unavailable in these scenarios or the feature may delete a parent (or ancestor) element.

This applies to the delete feature in Quark XML Author that is executed through the Component Cut, Delete and Remove commands. And, this applies whether the commands are executed via the toolbar, Context Menu or shortcut keys.

Backspace and Delete in Structured Authoring

The behavior of the Backspace and Delete keys depends on what is currently selected and the location of the cursor. The behavior is also controlled by structured authoring rules. See “Deleting Content Elements and Structure”.

The following applies to content both outside and inside of tables.

Selection Exists

If the selection contains only text, the text is deleted.

If the selection contains only optional elements, the optional elements are deleted.

Outside of Tables: If the selection contains any required elements, the optional elements are deleted and text in the required elements is deleted.

Inside of Tables: If the selection contains any required elements, the optional elements are deleted and text in the required elements is deleted. This applies to selections that span any table element such as cells, rows, sections, etc.

For a paragraph type cell, an empty paragraph remains. For other cell types, an insertion point remains.

No Selection

In these use cases, the cursor is within an element, but nothing is selected. The cursor is not in an insertion point or Empty.

Pressing the Backspace key will delete text until no text remains to the left of the cursor.

In an optional element, pressing the Backspace key at the leftmost cursor position merges the text with previous element and deletes the current element. This also applies when the current element is a list item.

In a required element, pressing the Backspace key at the leftmost cursor position does not cause a change.

Pressing the Delete key deletes text until no text remains to the right of the cursor. Pressing the Delete key at the rightmost cursor position does not cause a change.

Entering / Pasting Text - Advanced

The behavior of the typing and pasting text depends on what is currently selected.

If the selection only includes text within a single element, that selection is replaced with the character typed or the text that is pasted. The selection can include all text of that single element.

If the selection includes more than one element and a printable character is typed or text is pasted, no change occurs.

Typing in an Empty

When the user types in an Empty, the system uses the **insertAfter** setting of the previous element. See “Table 14-2: Section Attributes.” If there are no previous sibling elements, then the parent element setting is used. For example, the first visible Empty in a table cell. See “Paragraph and ChildTextNode Elements”.

Tab in Structured Authoring

The following describes the behavior of the Tab key and various Tab key combinations within the Quark XML Authoring experience. This applies to para-

type elements: heading, paragraph, list, etc. The behaviors are organized by inside of tables and outside of tables.

Table 16-1: Tab Behavior Outside of Tables

Key Press	<i>cursor or selection starts at beginning of paragraph</i>	<i>cursor or selection DOES NOT start at beginning of paragraph</i>
Tab	increase indent (if configured); otherwise insert tab	insert tab
Shift+Tab	decrease indent (if configured); otherwise no action	no action
Ctrl+Tab	insert tab	insert tab
Ctrl+Shift-Tab	no action	no action

Table 16-2: Tab Behavior Inside of Tables

Key Press	<i>cursor or selection contained within 1 cell (NOT rightmost bottom-most cell)</i>
Tab	navigate to next cell
Shift+Tab	navigate to previous cell
Ctrl+Tab	insert tab (if within 1 para-type); otherwise no action
Ctrl+Shift+Tab	no action

When the cursor or selection is in the rightmost bottom-most cell, the Tab behavior is to “add a row”, if the configuration permits that action.

Cut, Copy, Paste in Tables

This section discusses rules and behaviors for cut, copy and paste in tables. The integrity of the document structure must be maintained the same inside of tables as it is outside.

General

Whether an element can be pasted requires that the definition of the element and it's children definitions are compatible with the definition of the destination element. There may be ancestor definitions that prevent paste. For example, the user cannot paste a row into a header row definition with max occurs of 1. This general statement holds true whether you are pasting content into a table cell from outside a table, pasting content outside a table from a table cell, or pasting table cell to table cell.

Pasting Table Elements

This is an advanced discussion of the Paste table elements feature. Table elements have been placed on the clipboard using either the cut or copy table element feature.

The Paste table elements features functions differently depending on several factors. Are any table elements selected at the paste location? If table elements are selected, are the dimension of the selection different than the dimensions of the table elements you are pasting?

The discussion is organized into the following sections:

- pasting into a selection
- pasting without a selection

Pasting Table Elements Into a Selection

When you are pasting into a selection of table elements and the dimensions of the selection are different than the dimensions of the table elements you are pasting, then either repeating or truncation of table cells can occur. The following examples illustrate this logic.

In the first example, the clipboard contains a selection of table cells with the dimension 3 x 2.

A1	B1	C1
A2	B2	C2

At the paste location, you select 2 cells x 3 cells.

When you paste, the results are as follows.

A1	B1
A2	B2
A1	B1

In the second example, the clipboard contains a selection of table cells with the dimension 2 x 3.

A1	B1
A2	B2

A3	B3
----	----

At the paste location, you select 3 cells x 2 cells.

When you paste, the results are as follows.

A1	B1	A1
A2	B2	A2

Pasting Table Elements Without a Selection

When you are pasting into a selection of table elements and the dimensions of the selection are different than the dimensions of the table elements you are pasting, then either repeating or truncation of table cells can occur. The following examples illustrate this logic.

In the first example, the clipboard contains a selection of table cells with the dimension 1 x 3 and the content 1, 2, 3.

1	2	3
---	---	---

The paste location is a table with the dimension 1 x 3 and the content A, B, C.

A	B	C
---	---	---

Place the cursor in the first cell of the paste location. The asterisk indicates the cursor position. No text or table elements are selected.

*A	B	C
----	---	---

When you paste, the results are as follows. The content A, B, C becomes 1, 2, 3.

1	2	3
---	---	---

In the second example, the clipboard contains a selection of table cells with the dimension 1 x 3 and the content 1, 2, 3.

1	2	3
---	---	---

The paste location is a table with the dimension 1 x 3 and the content A, B, C.

A	B	C
---	---	---

STRUCTURED AUTHORING

Place the cursor in the second cell of the paste location. The asterisk indicates the cursor position. No text or table elements are selected.

A	*B	C
---	----	---

When you paste, the results are as follows. The content B, C becomes 1, 2 and an additional column is added to the table. This is true no matter how many rows are in the table at the paste location.

A	1	2	3
---	---	---	---

In the third example, the clipboard contains a selection of table cells with the dimension 3 x 1 and the content 1, 2, 3.

1
2
3

The paste location is a table with the dimension 3 x 2 and the content A, B, C, D, E, F.

A	D
B	E
C	F

Place the cursor in the second cell of the paste location. The asterisk indicates the cursor position. No text or table elements are selected.

*A	D
B	E
C	F

When you paste, the results are as follows. The content A, B, C becomes 1, 2, 3.

1	D
2	E
3	F

In the fourth example, the clipboard contains a selection of table cells with the dimension 3 x 1 and the content 1, 2, 3.

1
2
3

The paste location is a table with the dimension 3 x 2 and the content A, B, C, D, E, F.

A	D
B	E
C	F

Place the cursor in the second cell of the paste location. The asterisk indicates the cursor position. No text or table elements are selected.

A	D
*B	E
C	F

When you paste, the results are as follows. The content B, C becomes 1, 2 and an additional row is added to the table. This is true no matter how many columns are in the table at the paste location.

A	D
1	E
2	F
3	

Creating Tables from Copied Table Elements

From the Quark XML Author user guide.

The user may insert a table that is filled with table elements that they have already copied to the clipboard.

You may copy a variety of elements to the clipboard including the following: single and multiple cells, rows or sections. The copied elements and their content are used to create a new table at any allowed location. The new table receives a copy of the metadata of the original table.

If the paste command is not available, this indicates that either a table is not allowed at this location or that a table is allowed, but the system cannot create a table at that location, using the copied elements, that is compatible with allowed table configurations.

"strict XML" execution of Cut and Copy

In Quark XML Author configurations prior to 4.0, the Cut and Copy commands located in the Component submenu performed a "strict XML" execution of these commands. In versions 4.0 and later, these commands will execute the same feature as the Cut and Copy commands at root-level of the Content Menu.

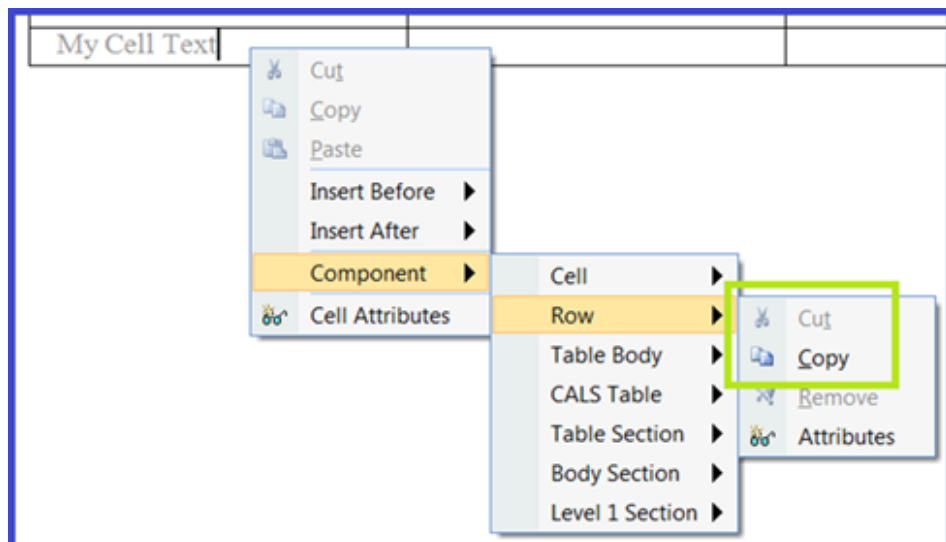


Figure 16-1: Component > Row submenu: Cut and Copy commands

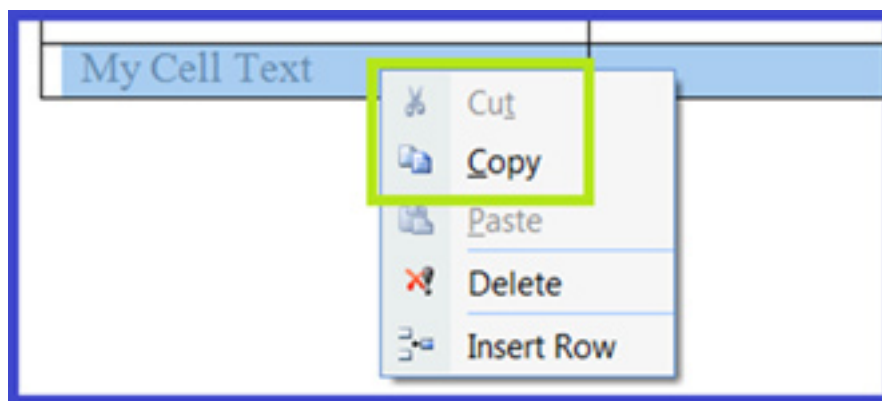


Figure 16-2: Row root-level Context Menu: Cut and Copy commands

Previously the “strict XML” execution of Cut and Copy placed XML on the clipboard that contained text, string and Unicode, but none of the Quark XML Author formats. This feature was used to paste XML into other XML tools. However, this feature is deprecated in Quark XML Author version 4.0.

Multiple element - Cut, Copy, Paste

The overview and detailed description of this feature is included the user guide and is the same for each version of the application. This document contains a brief description as well as some additional details applicable to a configuration engineer.

The Cut, Copy, Paste feature works with selections that include multiple elements and multiple content types such as text, images and tables. This includes all metadata and emphasis within the selection as well as child elements.

The Cut and Copy features are supported by:

- InternalClass: Ribbon buttons
- Select/Component Context Menus
- Shortcut keys: CTRL + X, CTRL + C, Shift + DEL, CTRL + Insert

The following provides details about the scenarios and use cases that the Cut, Copy, Paste feature supports. These are in addition to those listed in the user guide:

- Selection of inline text may be pasted as a new element of the same type as the source element, if valid.
- Selection of complete text element may be pasted as inline text, if valid.
- Selection of a block image may be pasted to replace other block images, if valid.
- Selection of a block image may be pasted to replace an inline content selection. Destination selection may contain text, images and/or equations, if valid.
- Selection of a block image may be pasted as a new inline image, if valid.
- Selection of block equation may be pasted to replace an inline content selection. Destination selection may contain text, images and/or equations, if valid.
- Selection of a block equation may be pasted as a new inline equation, if valid. Selection of inline content may contain images and/or equations. Selection may be pasted as a new element of the same type as the source element, if valid.
- Selection of an inline image or inline equation may be pasted as a new block image/new block equation element. This feature cannot yet be demonstrated DITA BusDocs, because in the BusDocs configuration the image and equation element require a Figure wrapper/parent element.
- Selection of an inline equation may be pasted to replace an inline equation.
- Selection of an inline equation may be pasted as a new inline equation.

A single element selection that includes the Empty above and the Empty below will also invoke the GetComponentId EI, if configured.

Integration with Content Management Systems

Communication with a Content Management System (CMS) for content upload and download is handled externally from Quark XML Author. Quark XML Author passes the content and information for handling that content to an external DLL that performs the upload and download tasks. This chapter explains:

- How to configure the Quark XML Author Internal Class **SaveToRepository** that is used to integrate with the external DLL.
- The upload interface that the external DLL must implement in order for SaveToRepository to communicate with it.
- The download interface that the external DLL must expose in order for Quark XML Author to be able receive content from the repository.

SaveToRepository

Quark XML Author uses the SaveToRepository Internal Class to call the external DLL that performs the upload operation. The SaveToRepository internal class is structured as shown below. Table 17-1 defines the attributes used for the InternalClass element.

```
<InternalClass name="SaveToRepository" nativeFormat="true"
assembly="assemblyName" class="className">
  <Argument type="Tokens">
    <Token>Token Data as needed, one datum per token.</Token>
  </Argument>
  <Argument type="Delegates">
    <Delegate>Delegates as needed.</Delegate>
  </Argument>
</InternalClass>
```

Table 17-1: SaveToRepository Attributes

Attribute	Required	Definition
assembly	yes	Specifies the file name of the .NET class library that implements the upload process (without the .dll extension).
class	yes	Specifies the fully namespace-qualified class name within the .dll file being called for the external process

Attribute	Required	Definition
name	yes	Name of the class to be invoked. In this case, SaveToRepository .
nativeFormat	no	Boolean. Defaults to false . When set to true , Word binary is included with the document.

The InternalClass element is used in the configuration file, usually for the document class.

Example

Here is an example of how SaveToRepository might be used in a DocConfig file. Like the other examples in this chapter, it assumes that we are communicating with a CMS called Tritus.

```
<MenuItem resourceId="SaveToTritus">
  <InternalClass name="SaveToRepository" assembly="Tritus"
class="fullNamespaceOfTheClass.Tritus" nativeFormat="false">
    <Token>enableLogging=true</Token>
    <Token>credentials=windows</Token>
  </InternalClass>
</MenuItem>
```

The example above maps the SaveToRepository internal class to a menu item. When the menu item is selected, SaveToRepository will dynamically load the Tritus.dll assembly and look for the upload() implementation in the specified class. (The upload() interface is described below). If the assembly or any of its dependencies do not load or the specified class does not exist or does not implement the upload() signature, an error message will be displayed and Quark XML Author will not perform any upload.

Upload() Interface

In order to allow Quark XML Author to check documents into a repository, the external DLL must implement one of the following interfaces:

```
bool upload(System.String fileContents, string filename, string[] tokens)
```

```
bool upload(System.String fileContents, string filename, string[] tokens, Delegate[]
delegates)
```

fileContents

The complete XML document that currently has focus. The XML will contain any changes occurring after the document was opened or last saved locally.

filename

The name of the XML document that currently has focus.

tokens

A list of tokens that are specified in the InternalClass that calls the upload interface. (See SaveToRepository on page 92. The string array is made up of the contents of each of the <Token> nodes specified as a child of the InternalClass node. If no tokens are provided in the Internal Class, the token array will be null. The tokens are completely arbitrary and can be any text string that an upload() function needs in order to execute successfully.

delegates

An optional list of predefined delegates supported by Quark XML Author. Supported delegates are discussed in section 6.4.

The upload() interface returns a Boolean value which indicates whether an upload succeeded or failed. For example:

```
<MenuItem resourceId="SaveToTritus">
  <InternalClass name="SaveToRepository" assembly="Tritus"
class="fullNamespaceOftheClass.Tritus" nativeFormat="false">
    <Tokens>
      <Token>enableLogging=true</Token>
      <Token>credentials=windows</Token>

      <Token>webservice=http://soap.tritus.com/schemas/TritusWebService
s.wsdl</Token>
      <Token>iniFileLocation =C:\Program Files\XML
Author\TritusCMS.ini</Token>
    </Tokens>
  </InternalClass>
  <InternalClass name="DocumentClose"/>
</MenuItem>
```

The menu item shown above has two actions associated with it. Normally, both of them will execute in sequence when the menu item is clicked, but a return of false from the upload() method causes any subsequent actions to be ignored. Thus, a return value of false will cancel the execution of the DocClose event which simply cancels the closing of the document and keeps it open in Quark XML Author if upload failed.

Example 1: Upload without delegates

Below is a very simple upload() function that implements the simpler signature:

```
bool upload(System.String fileContents, string filename, string[] tokens)
```

It uses a published web service used by the Tritus CMS to perform the upload function. If there are any other details embedded in the document, we can use that in the upload function as well. In the example below, the document root has an embedded **id** attribute (this **id** is being assigned by Tritus CMS when it provided the document), which is extracted and used for the upload.

```
namespace fullNamespaceOftheClass
{
  public class Tritus
  {
    bool upload(string fileContents, string fileName, string[]
tokens)
    {
      try{
        //These are completely arbitrary and not required.
        //read the tokens - Order of tokens are preserved.
```

```

        bool log = tokens[0].Split('=')[1].Equals("true"); //to
turn logging on.
        string cred = tokens[1].Split('=')[1]; //use windows
credentials
        string url = tokens[2].Split('=')[1]; //web service URL.
        string iniFile = tokens[3].Split('=')[1]; //This
location will point to any CMS specific information required for
the upload() function.

        //get credentials for access to the CMS
        ICredentials credentials =
CredentialCache.DefaultCredentials; //since the token[1] says
gets windows Credentials

        //read the document to get some details from it.
        XmlDocument doc = new XmlDocument();
        doc.LoadXml(fileContents);
        string id = doc.DocumentElement.GetAttribute("id");
        //if there is no ID we consider this to be a brand new
document, or if there is one if we consider this to be an
existing document.

        Tritus.CMS.WebServices service = new
Tritus.CMS.WebServices();
        sservice.Credentials = credentials;
        int newID = 0;
        if(id ==String.Empty){
            //New document
            newID = service.ImportDocument(fileContents, fileName,
credentials);
        }
        else{
            //Existing document - attempt to invoke the Checkin
webservice.
            newID = service.CheckinDocument(id, fileContents,
fileName, credentials);
        }
        return success;
    }catch(Exception e)
    {
        //Good idea to log any specific log errors.
        //Can be either thrown back to Quark XML Author or a
false return to signify failure
        return false;
    }
}
}
}
}

```

Example 2: Upload with delegates

The other upload() signature that can be used is:

```
bool upload(System.String fileContents, string fileName, string[] tokens, Delegate[]
delegates);
```

Here is an example of how this signature can be called by a menu item:

```

<MenuItem resourceId="SaveToTritus">
    <InternalClass name="SaveToRepository" assembly="Tritus"
class="fullNamespaceOftheClass.Tritus" nativeFormat="false">
        <Tokens>
            <Token>enableLogging=true</Token>
            <Token>credentials=windows</Token>

```

```
<Token>webservice=http://soap.tritus.com/schemas/TritusWebService
s.wsdl</Token>
    <Token>iniFileLocation =C:\Program Files\XML
Author\TritusCMS.ini</Token>
    <Tokens>
        <Delegates>
            <Delegate>InvokeInternalClass</Delegate>
        </Delegates>
    </InternalClass>
</MenuItem>
```

An InvokeInternalClass delegate is used inside the InternalClass definition; this delegate will be passed to the upload() method defined below. (Supported delegates are discussed in section 6.4.)

```
bool upload(string fileContents, string fileName, string[]
tokens)
{
    try{
        //Remember these are completely arbitrary and not required.
        If there are no tokens in the config file, tokens will be null.
        //read the tokens - Order of tokens are preserved.
        bool log = tokens[0].Split('=')[1].Equals("true"); //to
turn logging on.
        string cred = tokens[1].Split('=')[1]; //use windows
credentials
        string webserviceURL = tokens[2].Split('=')[1]; //web
service URL.
        string iniFile = tokens[3].Split('=')[1]; //This location
will point to any CMS specific information required for the
upload() function.
```

```
Invision.Xpress.ExtensibilityDelegates.InvokeInternalClass
invokeInternalClassDelegate = delegate[0] as
Invision.Xpress.ExtensibilityDelegates.InvokeInternalClass;
```

```
//get credentials for access to the CMS
```

```
ICredentials credentials = CredentialCache.DefaultCredentials; //since the
token[1] says gets windows Credentials
```

```
//read the document to get some details from it.
```

```
XmlDocument doc = new XmlDocument();
```

```
doc.LoadXml(fileContents);
```

```
string id = doc.DocumentElement.GetAttribute("id");
```

```
//if there is no ID we consider this to be a brand new document, or if there is
one if we consider this to be an existing document.
```

```
Tritus.CMS.WebServices service = new
Tritus.CMS.WebServices();
service.Url = webserviceURL;
service.Credentials = credentials;
int newId = 0;
if(id ==String.Empty){
    //New document
    newId = service.ImportDocument(fileContents, fileName,
```

```

credentials);
    }
    else{
        //Existing document - attempt to invoke the Checkin
        webservice.
            newId = service.CheckinDocument(id, fileContents,
            fileName, credentials);
        }
        //Before returning, check with user whether to close
        Document.
            if(success && invokeInternalClassDelegate != null &&
            MessageBox.Show("Close Document?", "Tritus",
            MessageBoxButtons.YesNo, MessageBoxIcon.Warning) ==
            DialogResult.Yes){
                //The second parameter are for any other parameters for
                DocumentClose.Please refer to DocumentClose documentation for all
                possible parameters
                invokeInternalClassDelegate("DocumentClose", new
                Hashtable()); //This will initiate the closing of the document.
            }
            else
                //Do nothing. The document will remain open in the UI.

                //As seen above, the upload() method can initiate closure
                of a document as well through the above delegate call. This makes
                it a little more flexible in terms of closing document for
                //various scenarios. In the above case, I close the
                document only if user chooses Yes in the dialog.

                return success; //since there is no event following the
                upload() in the MenuItem, the return value here does not hold any
                relevance.
            }catch(Exception e)
            {
                //Good idea to log any specific errors.
                //Can be either thrown back to Quark XML Author or a
                false return to signify failure
                return false;
            }
        }
    }
}

```

Example 3: Upload a document fragment

In the previous examples, the upload was for a complete document. There are multiple ways to upload/import partial fragments into a CMS by making use of the Extensibility Interface. (A complete discussion of the Extensibility Interface can be found in section 6.) For example, let's say the user wants to make a List in the document reusable by importing it into the CMS as a brand new fragment. Here is a sample configuration that will do it:

```

<!-- Method definition (in the doc-config) file-->
<Method id="UploadAnyFragment" assembly="Tritus"
class="fullNamespaceOftheClass.Tritus" method="UploadFragment">
    <Argument type="XomCurrentNode"/>
    <Argument type="Tokens">
        <Token>enableLogging=true</Token>
        <Token>credentials=windows</Token>

        <Token>webservice=http://soap.tritus.com/schemas/TritusWebService
        s.wsdl</Token>
    </Argument>
    <Argument type="Delegates">
        <Delegate>ToggleEditableRegion</Delegate>
    </Argument>

```



```

</Method>

<!-- Method reference (in the xas file)-->
<ElementDef name="List" friendly="List" visible="false">
  <Section>
    <Sequence maxOccurs="unbounded">
      <SectionType name="ListItem"/>
    </Sequence>
  </Section>
  <Attributes>
    <AttributeType name="id"/>
  </Attributes>
  <ExtensibilityMethods>
    <!-- HERE IS THE ACTUAL REFERENCE-->
    <ExtensibilityMethod id="UploadAnyFragment" friendly="Reuse
Component"/>
  </ExtensibilityMethods>
</ElementDef>

```

There are two parts to any Extensibility Interface implementation—definition and reference. The definition in this case is in the document configuration file. The reference is in the Quark XML Author Structure (xas). The reference is allowing us to access the UploadFragment from the UI through the **Component->List Context Menu**. So if we want to make an existing list in a document a reusable component in the CMS, we will right-click on the List (or any of its children), access the **Component->List** menu and we should see the **Reuse Component** menu item. Clicking that will invoke the UploadFragment() method. As shown in the definition, it takes the **XomCurrentNode**, which in this case will be the List instance. The **ToggleEditableRegion** delegate is passed in to mark the region non-editable (indicated by color shading the background of the region) after the upload is successful. This reference must be repeated for each element that can be marked as a reusable fragment. For example, if tables in the document can be marked as reusable, the reference would need to appear in the table element definition as well.

The upload method that would be used in conjunction with the example above is shown below:

```

namespace fullNamespaceOftheClass
{
    public class Tritus
    {
        public void UploadFragment(XmlNode nodeToBeUploaded, string[]
tokens, Delegate[] delegates)
        {
            //Remember these are completely arbitrary and not
required.If there are no tokens in the config file, tokens will
be null.
            //read the tokens - Order of tokens are preserved.
            bool log = tokens[0].Split('=')[1].Equals("true"); //to
turn logging on.
            string cred = tokens[1].Split('=')[1]; //use windows
credentials
            string webserviceURL = tokens[2].Split('=')[1]; //web
service URL.

            Invasion.Xpress.ExtensibilityDelegates.ToggleEditableRegion
toggleEditableRegionDelegate = delegates[0] as
Invasion.Xpress.ExtensibilityDelegates.ToggleEditableRegion;

            //get credentials for access to the CMS
            ICredentials credentials =
CredentialCache.DefaultCredentials; //since the token[1] says
gets windows Credentials

```

```

        Tritus.CMS.WebServices service = new
Tritus.CMS.WebServices();
        service.Url = webserviceURL;
        service.Credentials = credentials;
        bool success = false;
        int newId = 0;
        string id = nodeToBeUploaded.GetAttribute("id");
        if(id ==String.Empty){
            //New fragment
            newId = service.ImportFragment(fileContents,
credentials);
        }
        else{
            //Existing fragment - attempt to invoke the Checkin
webservice.
            newId = service.CheckinFragment(id, fileContents,
credentials);
        }
        //Make the uploaded node non-editable and also toggle the
region in the UI to make it read-only.
        if(newId > 0 && toggleEditableRegionDelegate != null)
        {
            XmlNode[] nodesToBeMadeReadonly = new XmlNode[1];
            nodesToBeMadeReadonly.Add(nodeToBeUploaded);
            //For all nodes append an inv:access=readonly attribute
to make sure the nodes cannot be modified.
            XmlAttribute accessAttr =
nodeToBeUploaded.OwnerDocument.CreateAttribute("inv", "access",
"urn:xpressauthor:xpressdocument");
            accessAttr.Value = "read-only";
            nodeToBeUploaded.Attributes.Prepend(accessAttr);
            //Assign the new id to the id attribute.
            nodeToBeUploaded.SetAttribute("id", newID);
            toggleEditableRegionDelegate(nodesToBeMadeReadonly,
true, true); //This will notify the UI to make the List and all
its region non-editable
        }
    }
}

```

Download and Content Reuse

Two types of operations for retrieving content from a CMS need to be considered: opening an entire document and importing content fragments into an already open document.

Open a CMS Document in Quark XML Author

Quark XML Author has built-in classes to open a document from a local or network drive, or from a network WEBDAV folder. If the location is on a proprietary CMS, there are two possible ways to support opening a file:

- Leveraging an existing hook in the external CMS. For example, the Documentum CMS, through its Webtop interface, exposes a way to launch an external editor to edit XML files. Quark XML Author can be configured as such an external editor through the File types association in Windows Explorer.
- From within Quark XML Author.

The simplest way to do the latter would be to write an extensibility method to download the entire document into a temp location on disk and launch Quark XML Author to open the downloaded file. Here is a simple example that does just that:

```
<Method id="OpenFromTritus" assembly="Tritus"
class="fullNamespaceOftheClass.Tritus" method="OpenFromTritus">
  <Argument type="Tokens">
    <Token>credentials=TritusLogin.xml</Token>

    <Token>webservice=http://soap.tritus.com/schemas/TritusWebService
s.wsdl</Token>
    <Token>openLocation=c:\Tritus\Export</Token>
  </Argument>
</Method>
```

This method definition invokes the OpenFromTritus method exposed in the class shown below:

```
namespace fullNamespaceOftheClass
{
    public class Tritus
    {
        public void DocumentOpen(string[] tokens)
        {
            //read tokens
            string openLocation = tokens[3].Split('=')[1];
            string webserviceURL = tokens[1].Split('=')[1]; //WSDL
location
            //Connect to the Tritus web service
            Tritus.CMS.WebServices service = new
Tritus.CMS.WebServices();
            service.Url = webserviceURL;
            service.Credentials = credentials;

            //Launch some sort of File Explorer UI to get the file
path and download it. Remember this is just an example
            //and the implementation can differ. The ultimate
objective is to download the file to a local temp location and
have Quark XML Author launch
            //this temporary file.
            stream content = service.OpenFromRepository(ref
returnContentType, ref filename); //get the content type, and the
opened object's name
            if(content != null &&
returnContentType.IndexOf("text/xml") > -1) //Only interested in
XML content
            {
                //Save it to a local location with the returned
filename
                string tempFileName = openLocation + "\\\" +
DateTime.Now.Ticks + "\\\" + filename + ".xml";
                FileInfo tempF = new FileInfo(tempFileName);
                if (!tempF.Directory.Exists)
                    tempF.Directory.Create();
                StreamWriter writer = new
StreamWriter(tempF.FullName, false, "UTF-8");

                System.IO.StreamReader sr = new
System.IO.StreamReader(content, Encoding.GetEncoding("UTF-8"));
                //Assuming encoding as UTF-8
                string fileContents = sr.ReadToEnd();
                sr.Close();
                //If the file or part of it needs to be launched in
```

```
readonly mode, the nodes to be launched in read-only mode can be
assigned an Invision defined
    //attribute called inv:access like below.
    XmlDocument doc = new XmlDocument();
    doc.LoadXml(fileContents);
    XmlAttribute accessAttr = doc.CreateAttribute("inv",
"access", "urn:xpressauthor:xpressdocument");
    accessAttr.Value = "read-only";
    doc.DocumentElement.Attributes.Prepend(accessAttr);
    writer.Write(doc.OuterXml);
    writer.Close();

    if(File.Exists(tempFileName)
        Process.Start(tempFileName); //This assumes that
the text/xml mimetype in the file type associations is mapped to
Quark XML Author.

    }
}
}
```

Importing content into existing documents

External content can be imported into Quark XML Author by dragging and dropping hyperlinks to the external content into the document. Content links can be dropped in “Empty” locations in the document or onto a Reference element that has previously been inserted.

“Empty” locations are nothing but an empty paragraph that Quark XML Author uses to delineate separate content elements. These paragraphs are assigned the “Empty” style and cannot hold any content. When the user drops a link onto an Empty location, Quark XML Author inserts the dropped content and surrounds the new element(s) with Empty paragraphs. The dropped content becomes a part of the document and can be edited in-line without affecting the source. This content is therefore a ‘copy’ and does not retain any relationship with the original content.

Reference elements can be defined so that content can be imported that is not editable within Quark XML Author. A reference container is simply an XML container element with an **href** attribute; the attribute points to the resource being referenced. See section 14.7 for information on defining reference elements in the Quark XML Author Structure.[1]

For either method of importing content, the ResolveReferences node in the AppConfig file must be present and configured to accept data from the CMS. A typical configuration from the AppConfig.xml looks like this:

```
<ResolveReferences>
    <Reference pattern="?objId=x-en-tritus:" assembly="Tritus"
class="IfullNamespaceOftheClass.Tritus">

<Token>webservice=http://soap.tritus.com/schemas/TritusWebService
s.wsdl</Token>
    <Token>credentials=TritusLogin.xml</Token>
    </Reference>
    <!-- This is the default fallback pattern (.) that matches
every url. -->
    <Reference pattern=".*" assembly="SomeOtherDll"
class="SomeOtherNamespace.SomeOtherClass"/>
</ResolveReferences>
```

ResolveReferences can hold multiple Reference nodes, each of which can invoke a different download() interface to get content from one or more content management systems. The download() interface itself is explained in the next section.

Table 17-2 defines the attributes used for the Reference node.

Table 17-2: Reference Attributes

Attribute	Required	Definition
assembly	yes	Name of the assembly which contains the download() interface implementation.
class	yes	Name of the class which contains the download() interface implementation.
pattern	yes	<p>The pattern attribute's value must be a regular expression that contains a matching pattern for the URI that refers to the CMS. When an object is dragged into Quark XML Author, Quark XML Author searches for a pattern that can be matched against the URI for the object being dragged into it.</p> <p>In the example above, if the URL being processed contains ?objId=x-en-tritus: then the IfullNamespaceOftheClass.Tritus.download() method is invoked. For all other URLs, SomeOtherNamespace.SomeOtherClass.download() is invoked.</p> <p>To allow all URIs to match a single Reference node, the wildcard character phrase (.*) can be used as the value of the pattern attribute.</p>

[1] It is possible to import content in non-editable form without using reference elements, but this requires additional external resources. For example, the DocumentOpen event could be used to call an external DLL that scans the document, searches for content that should not be edited, and marks it accordingly. However, the content still “lives” within the document, whereas with a reference element, the content “lives” in the CMS and is merely referenced by the document.

Download Interface

The external DLL that calls the CMS must implement one of the following interfaces:

Stream download(string uri, string[] tokens, out string returnContentType)

Stream download(string uri, string[] tokens, out string returnContentType, out string alternateText)

Table 17-3: Download Interface Parameters

Parameter	Definition
uri	The actual URI used for the download.
tokens	<p>A string array of tokens used in the AppConfig file (see the example in the previous section). The token array may allow a zero-length string array to be passed in the event that no tokens are provided.</p> <p>The current document's filename is always passed to the download method as a separate token, so that if the ResolveReference node has three tokens, the actual number of tokens provided to the download method will be four.</p>
returnContentType	The content type of the downloaded stream that the method is returning to Quark XML Author.
alternateText	An optional parameter which can be used exclusively for image references; allows Quark XML Author to assign and display an alternate text value for the image.

The method must return a stream of the downloaded content.

Here is a very simple download() example:

```
namespace fullNamespaceOftheClass
{
    public class Tritus
    {

        public Stream download(string url, string[] tokens, out
string returnContentType)
        {
            //Using the first Reference node as an example..
            string webserviceURL = tokens[0].Split('=')[1]; //WSDL
location
            string cred = tokens[1].Split('=')[1]; //use windows
credentials
            //get credentials for access to the CMS
            ICredentials credentials =
CredentialCache.DefaultCredentials; //since the token[1] says
gets windows Credentials
            Tritus.CMS.WebServices service = new
Tritus.CMS.WebServices();
```

```

        service.Url = webserviceURL;
        service.Credentials = credentials;

        //The url that comes in is the actual URL being resolved
        by Quark XML Author. Let's say it's of the following form:
        //http://hyperbole.hyperactive.net/Documents?objId=x-en-
        tritus:TT0000000000509154:0000:2.
        //We want to pull the objId from the URL and use it in
        the GetObject() web service call.
        string objId = "x-en-tritus:TT0000000000509154:0000:2";
        //Have some sort of UI to see if user wants it checked
        out, referenced OR a copy. ALternately this information can come
        directly from the URL itself.
        //For example, there can be 3 seperate URLs -
        Edit(..?objId=x-en-tritus:TT0000000000509154:0000:2&type=edit). ,
        Copy(..?objId=x-en-tritus:TT0000000000509154:0000:2&type=copy),
        Reference(..?objId=x-en-
        tritus:TT0000000000509154:0000:2&type=reference)
        bool checkout= false;
        bool readonly = true;
        stream content = service.GetObject(objId, checkout,
        readonly, out contentType);
        returnContentType = contentType; //Before returning, make
        sure the returnContent mimetype is set, for ex:- text/xml,
        image/jpg etc.
        if(returnContentType == "text/xml" && readonly)
        {
            System.IO.StreamReader sr = new
            System.IO.StreamReader(content, Encoding.UTF8); //Assuming
            encoding as UTF-8
            string fileContents = sr.ReadToEnd();
            sr.Close();
            XmlDocument doc = new XmlDocument();
            doc.LoadXml(fileContents);
            //if the url is dropped on reference links instead of
            on "empty", there is no need to perform the next step of tagging
            elements as read-only.
            //Any content within the reference is automatically
            tagged as read-only within the Quark XML Author UI and stripped
            off when the document is serialized
            //leaving behind a reference container.
            XmlAttribute accessAttr = doc.CreateAttribute("inv",
            "access", "urn:xpressauthor:xpressdocument");
            accessAttr.Value = "read-only";
            doc.DocumentElement.Attributes.Prepend(accessAttr);
            //Assign this attribute to every node recursively.
            //The fragment should have some CMS attributes (at
            the very least it should have the id which will help Quark XML
            Author identify it as a CMS fragment and NOT a local fragment)
            writer.Write(doc.OuterXml);
            writer.Close();
            return new
            MemoryStream(Encoding.UTF8.GetBytes(doc.DocumentElement.OuterXml)
            );
        }
        else
            return content; //For all other content, return as is.
        Quark XML Author has to support the incoming content type at the
        current location.
    }

```

Example 1: Drag and Drop External Content

Consider a simple use case situation: an existing Quark XML Author document is open and we want to drag an object from the Tritus CMS into the document. The URL for the object has the following format:

```
http://hyperbole.hyperactive.net/Documents?objId=x-en-
tritus:TT0000000000509154:0000:2.
```

The URL being dropped is matched against all possible Reference patterns defined in the AppConfig. The first matching Reference node is selected (in our example above, the first Reference node) and its corresponding download() method is invoked with the dropped URL and any other tokens specified in the Reference node. The download needs to communicate with the CMS and get the requested object. Once the object is downloaded, the contentType of the downloaded content has to be assigned and the downloaded object has to be passed back to Quark XML Author as a stream.

Once Quark XML Author receives the stream, depending on the returned content's contentType, it determines whether this is an XML fragment or image content. If this is an XML fragment, Quark XML Author checks to see if that fragment is insertable at the given location. If there are multiple contexts of insertion (for example, if a table is dragged on an insertable, and there are two possible tables that can be inserted—one as a child of the Section and the other as a child of the Section's parent, Chapter), the user is shown a context menu that allows the user to choose the correct context of information. If the downloaded XML fragment cannot be inserted at the dragged location or is not XML, an error dialog is shown to the user and the fragment is discarded. Once the fragment is inserted and attached to the document, any CMS attributes present in the fragment are preserved within the document.

If Quark XML Author receives any other content (other than contentType=text/xml), at least one of the insertable elements defined for the location of the drop must support the same mimetype. For example, if the returning stream's content type is image/jpeg, there has to be at least one insertable at the drop position that supports the image/jpeg format. See section 14.7.1.1 for more information on mimetypes that can be associated with Element definitions.

Example 2: Using Paste to Insert External Content

A paste operation can be performed as well. The user can copy the actual URL of the object onto the clipboard by some means (for example, browsing to the object in Internet Explorer, right-clicking, and selecting **Copy Shortcut**) and right click on the appropriate Empty location in a Quark XML Author document. If the downloaded fragment is insertable at that position, the context menu will contain a Paste *element name* menu item (where *element name* is the friendly name of the object as defined in the Quark XML Author Structure) along with other insertables for that location. If the downloaded fragment cannot be pasted into that location, no Paste menu option appears. Alternately, the user may right-click on a Reference element that matches the downloaded fragment type (not the Empty location) and click **Assign reference** on the context menu. The Reference element's href attribute will be assigned the URL of the downloaded fragment.

This paste is an alternative implementation to the drag-drop scenario above but the implementation is nearly the same. The only difference in case of Copy and Paste is that once a valid URL is copied to the clipboard, the fragment is also downloaded and copied to the clipboard along with the original URL, so the user can actually click around the document to determine the right insertion spot in the document. The fragment remains on the clipboard until another copy operation is performed.

Example 3: Opening a Document Referencing External Content

When Quark XML Author opens a document that references external content, it recognizes any element that is a reference container from its definition in the Quark XML Author Structure and uses the element's **href** attribute to invoke the appropriate `download()` method and display the content.

For example, suppose a document being opened has the following structure:

```
<Document>

  <Section>

    <Title> Section name </Title>

    <ComponentReference
href="http://hyperbole.hyperactive.net/Documents?objId=x-en-
tritus:TT0000000000509154:0000:2.">

  </Section>

</Document>
```

In this example, `ComponentReference` is a reference element; its **href** attribute points to an external object. Quark XML Author displays that content in-line. If the object cannot be downloaded, the rest of the document continues to open and the reference container is flagged with an error label in the Quark XML Author document.

Configuring Smart Paste

Smart Paste is a Quark XML Author feature that allows you to copy content from a standard Word document and convert that content to structured XML to be pasted into a Quark XML Author document. This chapter explains how to configure Smart Paste.

Smart Paste Overview

Smart Paste is accomplished through the use of three stylesheets, an Extensibility Interface (EI) method, and an EI call from a document event. Here is an overview of what happens when an author uses smart paste:

1. In an external document, the author selects content and copies it to the clipboard. If the external document is Microsoft Word, then Word pastes XHTML onto the clipboard.
2. The author switches to a Quark XML Author document. Quark XML Author is now involved in the process. The Activated event fires and calls the EI method defined to handle Smart Paste.
3. The EI method determines that there is content on the clipboard that needs to be processed. It calls the external Smart Paste.
4. SP_Filter.xslt filters the XHTML content on the clipboard. This performs a level of normalization.
5. SP_Clipboard.xslt converts the XHTML content into intermediate format that conforms to the Smart Paste schema. See “The Intermediate Schema”. This performs an additional level of normalization. The normalized content is ready for configuration-specific processing.
6. A configuration-specific transform transforms the content into the XML structure that matches the target document schema. See “Configuration-specific transforms”.
7. An additional configuration-specific transform can be used to further transform tabular content. See “Configuration-specific table transforms”.

When the user subsequently right-clicks in a document location, if the content on the clipboard matches a legal document element as defined in the configuration, the context menu will contain a **Paste {element}** entry corresponding to the copied content.

Smart Paste Extensibility Method

SmartPaste.dll, which facilitates the Smart Paste feature, is a standard component of Quark XML Author and is located in the Quark XML Author application folder. In order to use Smart Paste, you must define an EI method in the DocConfig file. The EI method will call the **Start** method in the **Invision.Xpress.Extensible.SmartPaste** class. Detailed information on configuring EI methods can be found in section 6, but an example of the Method element and its attributes is shown below:

```
<MethodInfo>
  <Method id="SmartPaste" assembly="Quark.XA.SmartPaste"
    class="Invision.Xpress.Extensible.SmartPaste" method="Start">
    <!-- Arguments are discussed below -->
  </Method>
</MethodInfo>
```

Of course, you may name the EI Method anything you wish; typically **SmartPaste** is used as the Method's id value for ease of reference.

The **Start** method requires one argument that contains three tokens. All three tokens must be provided in name=value format, where name identifies the name of the parameter being passed and value is the parameter value.

The first token identifies the location of the stylesheet that maps intermediate content to the Quark XML Author Structure. It takes the following format:

```
ImportXslt=quark\Renditions\smart-paste.xsl
```

The left side of the name=value pair must be **ImportXslt**. The right side of the pair is a path relative to the Quark XML Author application folder and must include the full name of the stylesheet file.

The second token defines how images are to be handled if any are present in the source content. Use Image=embed if the XAS specifies that images are embedded in the document and Image=reference if the XAS defines images as referenced content.

The third token is the debug setting. For production systems, this will always be set to Debug=false. If set to Debug=true, the system will display a window containing four tabs, each of which contains the results of one stage of the transformation process.

Configuration-specific transforms

A typical, fully-defined EI Method for Smart Paste is shown below:

```
<MethodInfo>
  <Method id="SmartPaste" assembly="Quark.XA.SmartPaste"
    class="Invision.Xpress.Extensible.SmartPaste" method="Start">
    <Argument type="Tokens">
      <Token>ImportXslt=quark\Renditions\smart-paste.xsl</Token>
      <Token>Image=embed</Token>
      <Token>Debug=false</Token>
    </Argument>
  </Method>
</MethodInfo>
```

The following is an example of a DITA XML specific transform call:

```
<!-- this snippet is found in doc config: DITA/sme-config.xml -
->
<ExtensibilityInterface>
```

```

<MethodInfo>
  <Method id="SmartPaste" assembly="Quark.XA.SmartPaste"
class="Invision.Xpress.Extensible.SmartPaste" method="Start">
    <Argument type="Tokens">
      <Token>ImportXslt=DITA/xml_smartPaste_dita.xsl</Token>
      ...
    </Argument>
  </Method>

```

Configuration-specific table transforms

An additional configuration-specific transform can be used to transform tabular content.

The following is an example of a DITA XML specific table transform call:

```

<!-- this optional snippet would be found in doc config:
DITA/sme-config.xml -->
<ExtensibilityInterface>
  <MethodInfo>
    <Method id="SmartPaste" assembly="Quark.XA.SmartPaste"
class="Invision.Xpress.Extensible.SmartPaste" method="Start">
      <Argument type="Tokens">
        ...
        <Token>CustomTableXslt=[PATH] / [FILENAME] .xsl</Token>
        ...
      </Argument>
    </Method>

```

Enabling Smart Paste

As with any EI method, you must also define some means to call it. The Smart Paste method is called by Quark XML Author's **Activated** event, which is fired whenever a Quark XML Author document receives focus. The Smart Paste method must be defined in the DocConfig file, similar to the EI method. Add the following element to the ExtensibilityInterface node of the DocConfig file:

```

<Activated>
  <ExtensibilityMethod id="SmartPaste"/>
</Activated>

```

The value of the id attribute must equal the value of the id value for the Smart Paste method.

The Intermediate Schema

The intermediate table schema for 4.0 is XHTML. When table content is pasted from Word, the format is XHTML and does not have to be converted.

Earlier versions of Quark XML Author used CALS as the intermediate schema which required the table content from Word to have to be converted from XHTML to CALS.

Smart Paste uses three stylesheets. Two standard stylesheets ship with Quark XML Author and are typically not modified[1]. These stylesheets are SP_Clipboard.xslt and SP_Filter.xslt, and are located in the Quark XML Author application folder. These files perform the first stages of the transformation process. The result is an XML document that conforms to the Smart Paste schema. It consists of an Import

element that contains the copied content elements. An overview of the schema is provided here.

Import can contain one or more of any of the following elements: Heading, Para, ListItem, Table, and Image.

The third stylesheet is configuration-specific and resides in the application \ configuration folder. For example, {application folder} \DITA\xml_smartPaste_dita.xsl.

[1] Although these stylesheets are typically not modified, they can be changed to support other types of source documents such as HTML, Microsoft Excel, or even other types of XML Author documents.

TextType Elements

The first three elements (Heading, Para, and ListItem) are all TextType elements and, as such, have the same substructure (described in section 18.4.1.1, “TextType Structure”).

Table 18-1: TextType Elements

Element	Definition
Heading	Heading elements are created from any Word paragraph that has a style of Heading # , where # is any number (Heading 1, Heading 2, etc.) Heading has no special attributes.
ListItem	<p>ListItem elements are created from any Word paragraph that is a list style (bullets or numbered). The style attribute specifies whether the list item type is bullet (style="ul") or numbered (style="ol"). For example:</p> <pre><ListItem style="ul">List item text</ListItem></pre> <p>Note that ListItem elements are not wrapped in any sort of parent List element; this must be accomplished in the final transform stylesheet.</p>
Para	Any text paragraph in the source content that does not meet the criteria for a Heading or a ListItem element is transformed into a Para element. Para has no special attributes.

TextType Structure

TextType elements comprise either text or a Hyperlink child element.

Text can be mixed with any combination of Bold, Italic, and Underline elements. Superscript and Subscript are also available, but may not contain any further markup.

Hyperlink contains text that also may be mixed with the markup elements listed above. It also has the attribute xlink:href, which contains the URL to which the hyperlink points.

Table

Smart Paste may only be configured for one table definition per configuration. The final table definition is determined by the final transformation of the intermediate table format (illustrated below) into the desired format (CALS, HTML, etc.).

The output from `sp_clipboard.xslt` converts the filtered Word XHTML into a table with an HTML-like structure. This is an example of a 2x2 Word table after transformation:

```
<Import xmlns:v="urn:schemas-microsoft-com:vml"
xmlns:o="urn:schemas-microsoft-com:office:office"
xmlns:w="urn:schemas-microsoft-com:office:word"
xmlns:sp="http://www.w3.org/TR/REC-html40"
xmlns:spx="urn:xpressauthor:smartpaste:extensions"
xmlns:inlinext="urn:convert"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xt="urn:xpressauthor:trackchanges">
  <Table>
    <tbody>
      <tr>
        <td tempid="EDDDBA" colspan="" rowspan=""
tableRowColumnSize="239.4pt" tableBorderBottomStyle=""
tableBorderTopStyle="" tableBorderRightStyle=""
tableBorderLeftStyle="">
          <Para>Table cell one</Para>
        </td>
        <td tempid="EBDDBA" colspan="" rowspan=""
tableRowColumnSize="239.4pt" tableBorderBottomStyle=""
tableBorderTopStyle="" tableBorderRightStyle=""
tableBorderLeftStyle="">
          <Para>Table cell two</Para>
        </td>
      </tr>
      <tr>
        <td tempid="EDBDBA" colspan="" rowspan=""
tableRowColumnSize="239.4pt" tableBorderBottomStyle=""
tableBorderTopStyle="" tableBorderRightStyle=""
tableBorderLeftStyle="">
          <Para>Table cell three</Para>
        </td>
        <td tempid="EBBDBA" colspan="" rowspan=""
tableRowColumnSize="239.4pt" tableBorderBottomStyle="single"
tableBorderTopStyle="" tableBorderRightStyle="single"
tableBorderLeftStyle="
none">
          <Para>Table cell four</Para>
        </td>
      </tr>
    </tbody>
  </Table>
</Import>
```

Image

The attributes and content of the Image element depend on the value of the Image token in the EI Method. If the token content is Image=reference, then the Image element is empty and the href attribute specifies the location of the referenced image. If the token content is Image=embed, then the Image element contains the Base 64 encoding for the image content and the href attribute is not used.

Table 18-2: Image Attributes

Attribute	Required	Definition
href	no	Value is the URL of a referenced image. If the file no longer exists at its original location when the copy operation is performed, the value will be a temporary location.
filename	yes	Value is the file name of the image.
coordinates	yes	Value is four comma-separated integers specifying, in this order: X position, Y position, width, and height of the image.

Final Transformation Stylesheet

The final transformation stage is performed by the stylesheet identified in the first Token described above. Although this stylesheet can have any file name, we will refer to it here as the “Smart Paste stylesheet”.

The Smart Paste stylesheet maps the intermediate document to the Quark XML Author Structure so that the content can be pasted into the Quark XML Author document.

Multipart Wrapper

If the user selects more than one paragraph[1] in the source document, the XML that is added to the clipboard must be contained in a Multipart wrapper element. Consider the very simple example of an Import document with two sibling paragraphs, as shown below:

```
<Import>
  <Para>Test1</Para>
  <Para>Test2</Para>
</Import>
```

These sibling elements must be placed within the Multipart wrapper. In the example below, the Para element from the intermediate stylesheet has been mapped to the <p> element:

```
<Multipart>
  <p>Test1</p>
  <p>Test2</p>
</Multipart>
```

The Smart Paste stylesheet must implement logic to determine whether or not the Multipart wrapper is necessary. The XSLT template below returns a value of true or false to specify whether or not the Multipart wrapper should be used. The template checks the intermediate XML content to see if the selection includes multiple sibling elements, including Heading and child elements:

```
<!--
*****
```

```

*****
** Import Template ++ Decide whether to use the Multipart
parent container

*****-->
<xsl:template name="IsMultipart">
  <xsl:choose>
    <xsl:when test="count(//Heading) > 1">
      <xsl:value-of select="true()" />
    </xsl:when>
    <xsl:when test="count(//Heading) = 1 and
count(//Import/*[position() = 1 and local-name() != 'Heading']) >
0">
      <xsl:value-of select="true()" />
    </xsl:when>
    <xsl:when test="count(//Heading) = 0 and count(//para) > 1">
      <xsl:value-of select="true()" />
    </xsl:when>
    <xsl:when test="count(//Heading) = 0 and count(//para |
//InvisionTable) > 1">
      <xsl:value-of select="true()" />
    </xsl:when>
    <xsl:when test="count(//Heading) = 0 and count(//para) = 1
and count(//Import/*[position() = 1 and local-name() != 'para'])
> 0">
      <xsl:value-of select="true()" />
    </xsl:when>
    <xsl:when test="count(//Heading) = 0 and count(//para) = 1
and count(//Import/*[position() = 1 and local-name() = 'para']) =
1 and count(//Import/*[local-name() != 'para' and local-name() !=
'listitem']) > 0">
      <xsl:value-of select="true()" />
    </xsl:when>
    <xsl:when test="count(//Heading) = 0 and count(//para) = 0
and count(//Import/*) > 1 and count(//Import/*[local-name() !=
'listitem']) = 0">
      <xsl:value-of select="false()" />
    </xsl:when>
    <xsl:when test="count(//Heading) = 0 and count(//para) = 0
and count(//Import/*) > 1">
      <xsl:value-of select="true()" />
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="false()" />
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

```

The rest of the stylesheet must be defined to map the intermediate elements, as described previously, to the appropriate elements in the Quark XML Author Structure.

[1] In this context, “paragraph” refers to any object in the Word document: paragraphs, tables, images, etc., and any combination of these.

Example: Intermediate Heading > Quark XML Author Section

In the following template, the stylesheet transforms any Heading element it finds into a Section element. It examines the Heading’s subsequent sibling elements and

places them within the section element until it finds another Heading element or reaches the end of the document.

```
<xsl:template match="Heading">
  <xsl:element name="section">
    <xsl:if test="(count(../*) > 0) or (string-length(text()) >
0)">
      <title>
        <xsl:apply-templates select="text() | Bold | Italic |
Underline | Superscript | Subscript | Hyperlink"/>
      </title>
    </xsl:if>
    <xsl:variable name="thisPos" select="count(preceding-
sibling::*) + 1"/>
    <xsl:variable name="lastPos">
      <xsl:choose>
        <xsl:when test="not(following-sibling::Heading)">
          <xsl:value-of select="count(//Import/*)+1"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:for-each select="following-sibling::Heading[1]">
            <xsl:value-of select="count(preceding-sibling::*) +
1"/>
          </xsl:for-each>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <xsl:for-each select="//Import/*">
      <xsl:if test="position() > $thisPos and position() <
$lastPos">
        <xsl:choose>
          <xsl:when test="local-name() = 'listitem' and
count(preceding-sibling::*[position() = 1][local-name() =
'para']) = 1"/>
          <xsl:when test="local-name() = 'listitem' and
count(preceding-sibling::*[position() = 1][local-name() =
'listitem']) = 0">
            <xsl:apply-templates select="."/>
          </xsl:when>
          <xsl:when test="local-name() = 'listitem' and
count(preceding-sibling::*[position() = 1][local-name() =
'listitem']) > 0"/>
          <xsl:otherwise>
            <xsl:apply-templates select="."/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:if>
    </xsl:for-each>
  </xsl:element>
</xsl:template>
```

Example: Handling List Items

The following template contains logic to handle list item elements.

```
<xsl:template match="listitem">
  <xsl:choose>
    <xsl:when test="count(preceding-
sibling::*[position()=1][contains(',listitem,', local-name())]) =
0">
      <itemizedlist>
        <xsl:copy>
          <xsl:apply-templates/>
        </xsl:copy>
        <xsl:apply-templates select="following-
sibling::*[position() = 1][local-name() = 'listitem']"/>
      </itemizedlist>
    </xsl:when>
  </xsl:choose>
```

```

        </itemizedlist>
    </xsl:when>
    <xsl:when test="count(following-
sibling::*[position()=1][local-name() = 'listitem']) > 0">
        <xsl:copy>
            <xsl:apply-templates/>
        </xsl:copy>
        <xsl:apply-templates select="following-
sibling::*[position() = 1][local-name() = 'listitem']"/>
    </xsl:when>
    <xsl:otherwise>
        <xsl:copy>
            <xsl:apply-templates/>
        </xsl:copy>
    </xsl:otherwise>
</xsl:choose>
</xsl:template>

```

Example: Handling Table Rows

The following template validates that a row should only be transformed if it contains table data (i.e. cell).

```

<xsl:template match="tr">
    <xsl:if test="descendant::td">
        <row>
            <xsl:apply-templates select="@*[name() != 'xmlns']"/>
            <xsl:apply-templates select="td"/>
        </row>
    </xsl:if>
</xsl:template>

```

Configuring MetaFormsBridge

In some instances, a Windows form is a better user interface for adding and editing XML data than the Word canvas. Metadata forms, called MetaForms, can be created using the **MetaForms Designer** tool. For more information about creating MetaForms, see the MetaForms Designer User Guide. This section explains how to create the Extensibility Interface method that calls a MetaForm.

Quark XML Author provides an external class assembly, MetaFormsBridge, which can be called to link a visual form with a Quark XML Author document for inputting non-narrative XML data. The class information is listed below:

Full class name: Invision.MetaForms.MetaFormsLaunch

Assembly: MetaFormsBridge

Method: XpressRun

Like all external methods, XpressRun is called via an Extensibility Interface(EI) method in the document configuration file. Because XpressRun would only be used in conjunction with a Quark XML Author document, its EI method would only exist in a DocConfig file, and not in the AppConfig file. In the example below, a method named DocumentAttributes calls XpressRun:

```
<Method id="DocumentAttributes" assembly="MetaFormsBridge"
class="Invision.MetaForms.MetaFormsLaunch" method="XpressRun">
  <!-- Method info -->
</Method>
```

For more information about building EI methods, see Section 6.

XpressRun Method

XpressRun returns a Boolean value. As with all EI methods, a return value of false halts processing of any further EI methods that may be associated with the command that called XpressRun. There are six signatures, shown below:

```
public bool XpressRun(string accessMode, XmlNode[] sourceNodes,
string[] tokens, Delegate[] delegates)
public bool XpressRun(string accessMode, XmlNode[] sourceNodes,
string[] tokens)

public bool XpressRun(string accessMode, XmlNode sourceNode,
string[] tokens)
public bool XpressRun(string accessMode, XmlNode sourceNode,
string[] tokens, Delegate[] delegates)

public bool XpressRun(XmlNode sourceNode, string[] tokens)
public bool XpressRun(XmlNode sourceNode, string[] tokens,
Delegate[] delegates)
```

The parameters that XpressRun may accept are explained below.

Table 19-1: XpressRun Parameters

Parameter	Definition
accessMode	Passes the access mode of the document. If the document's accessMode is set to Review , the document is set to read only. The MetaForm can still be displayed and viewed, but any changes made while the form is open will not propagate back to the document when the form is closed.
Delegate(s)	Provides a delegate or list of delegates that will be supplied to XpressRun. If <code>Invision.Xpress.ExtensibilityDelegates.UpdateAt</code> tributes is one of the delegates supplied, then it is executed on completion of form modifications. For information on Quark XML Author delegates, see Section 6.4.
sourceNode(s)	Provides the node or an array of nodes to be processed.
tokens	Specifies an array of values that can be passed to XpressRun to define where the MetaForm files are located, among other things. Tokens are explained in detail in the <i>XpressRun Tokens</i> section, below.

XpressRun Tokens

The following tokens may be supplied via the Tokens parameter:

Table 19-2: XpressRun Tokens

Parameter	Definition
Config= path	Replace path with the path to the MetaForms configuration file. This is the main file defining the form. The path is relative to the Quark XML Author application folder.
Rules= path	Replace path with the path to the rules file for the MetaForm. The path is relative to the Quark XML Author application folder.
Validate= true/false	If set to true , directs Quark XML Author to run a validation script if one is specified in the configuration file. Defaults to false .
ValidateMessage= message	Replace message with the text of the message to show on a validation error.
xmlns= namespace	Replace namespace with the namespace for the MetaForm.
XPath= path	Replace path with XPath into the source node

Parameter	Definition
	to use as the start node for processing.

MetaForms Files

The files that define a MetaForm can be placed in any location, but Quark recommends placing them in a subfolder of the folder containing the configuration files for the document class. For example, given a configuration for a document class called **Book**, the configuration files for Book might be located in:

C:\Program Files\Quark\XML Author\Book

The MetaForm files should be placed in a subdirectory of Book. Again, the name of the folder can be anything, but Quark recommends naming the folder **MetaForms**:

C:\Program Files\Quark\XML Author\Book\MetaForms

Example

The example below illustrates one possible use of MetaFormsBridge. In this example, the method is used to call a document attributes form.

```
<Method id="DocumentAttributes" assembly="MetaFormsBridge"
class="Invision.MetaForms.MetaFormsLaunch" method="XpressRun">
  <Argument type="AccessMode"/>
  <Argument type="XomRoot"/>
  <Argument type="Tokens">
    <Token>Config=Book\Metaforms\DocumentAttributes.xml</Token>
    <Token>XPath=./DocumentMetadata</Token>
  </Argument>
  <Argument type="Delegates">
    <Delegate>UpdateAttributes</Delegate>
  </Argument>
</Method>
```

This example implements the fourth XpressRun interface shown above, and reproduced here:

```
public bool XpressRun(string accessMode, XmlNode sourceNode, string[] tokens,
Delegate[] delegates)
```

When the EI method is invoked, it passes the document access mode and the entire document structure to XpressRun (XomRoot—see section 6.3.1 for general information on XOM and Section 6.3 for the XomRoot enumerated value) to XpressRun. The tokens point to the MetaForm's configuration file and the XPath location of the source node to begin processing in the document. The configuration file is located in the Metaforms subdirectory of the document configuration file's location, and is called DocumentAttributes.xml. The XPath points to the DocumentMetadata node.

Finally, the UpdateAttributes delegate is supplied. When the user finishes making changes to the form, UpdateAttributes is triggered.

Implementing Cross References

A cross reference is a navigable link within a document either to another location in the document or to an external document (another Quark XML Author document, a URL, or another type of document) stored in a content management system or other file storage system.

Implementing cross-references requires configuration in the AppConfig, DocConfig, Cross Reference EI Config and Schema.

AppConfig

A <Reference> node is necessary to resolve the cross reference link. See Section 7.1 for details on the <Reference> node. Implementing a cross reference in a DITA environment can use the dita-cms assembly shown in the example below.

```
<ResolveReferences>
  <Reference pattern=".*" assembly="dita-cms"
class="Invision.Xpress.CMS.DitaCMS">
    <Token>errorLogging=true</Token>
    <!-- The following specifies tooltip format:
{0}=filename and {1}=title -->
    <Token>alternateTextFormat={1} - {0}</Token>
  </Reference>
</ResolveReferences>
```

DocConfig

There are two components to configure in the DocConfig file: an EI Method and the ComponentCopy node.

EI Method

Create an Extensibility Method that points to the assembly, class, and method that returns the cross reference string information. Implementing a cross reference in a DITA environment can use the dita-cms assembly shown in the example below.

```
<Method id="GetComponentId" assembly="DITA-cms"
class="Invision.Xpress.CMS.DitaCMS" method="GetComponentId">
  <Argument type="Filename"/>
  <Argument type="XomCurrentNode"/>
</Method>
```

ComponentCopy Node

Implement the <ComponentCopy> node (Section 9.5) with the getReferenceId attribute. The value of this attribute is the value of the id attribute of the

Extensibility Interface <Method> defined above. If transform attribute is present in ComponentCopy node, then the cascading stylesheets within the ComponentCopy node will not be honored.<ComponentCopy getReferenceId="GetComponentId">

```
<Stylesheet xslt="..\CopyTransform.xslt" />
<Stylesheet xslt="..\CopyTransform1.xslt" />
</ComponentCopy>
```

ElementDef

You must specifically designate each element that you want to be able to target as a cross reference by assigning a value to the referenceAttribute attribute. See “Table 14-1” for information on referenceAttribute.

Only elements that have been assigned a referenceAttribute value will be available for cross referencing. For example, if your schema has a **section** element that may contain **paragraph** elements, and you define a referenceAttribute value to the section element definition but not the paragraphs, only the section will be available as a cross reference target, and not the paragraphs within it.

An example is provided below.

```
<ElementDef name="subuli2" xmlns="li" friendly="Sub-Sub List
Item" style="List Bullet 3" referenceAttribute="conref">
  <Section>
    <Para defaultHyperlink="xrefemph@format='html'">
      <Emphasis>
        <!-- Omitted for clarity -->
      </Emphasis>
    </Para>
    <Choice minOccurs="0" maxOccurs="unbounded">
      <SectionType name="lp"/>
      <ReferenceType name="image"/>
      <SectionType name="childTextNode"/>
    </Choice>
  </Section>
  <Attributes>
    <!-- Omitted for clarity -->
  </Attributes>
</ElementDef>
```

EmphasisDef

Cross references in Quark XML Author are represented to the user using a specialized emphasis definition. See Section 13.1.1 for information on building the specialized emphasis. Note that the <Custom> link type must be specified, and its assembly and class attributes must point to the external application that will handle the link behavior. In a DITA environment, the Invision.Xpress.CustomLink.DITA assembly shown in the example below can be used.



The external application must implement the ICustomLink interface. Details of this interface are beyond the scope of this document; if you

need to implement a custom external application, please contact Quark for technical support.

```
<EmphasisDef name="xref" friendly="Cross Reference"
xmlname="xref" excludeFromEmphasisCombo="true">
  <Specialized>
    <Link>
      <Multiple>
        <Custom substyle="dita"
assembly="Quark.XA.EI.ReferenceResolver"
class="XA.EI.ReferenceResolver.FileBasedReferenceResolver"/>
        <External substyle="html"/>
      </Multiple>
    </Link>
  </Specialized>
</Style/>
<SubStyles key="format">
  <SubStyle value="dita" friendly="Cross-Reference">
    <Style foreColor="Blue" underline="true"/>
  </SubStyle>
  <SubStyle value="html" friendly="Hyperlink">
    <Style foreColor="Blue" underline="true"/>
  </SubStyle>
</SubStyles>
<Attributes>
  <AttributeType name="href"/>
  <AttributeType name="format"/>
</Attributes>
<ExtensibilityMethods>
  <ExtensibilityMethod id="EditCrossReference" friendly=" -
Edit" showInContextMenu="true" enableXPath="self::*[local-
name()='xref' and @referencetype='Text' and
@*[local-name()='read-only']]">
  <ExtensibilityMethod id="UpdateReferences" friendly=" -
Update" showInContextMenu="true" enableXPath="self::*[local-
name()='xref' and @synctargettext='true']"><ExtensibilityMethod
id="MakeCrossReferenceReadOnly" friendly=" - Mark Read-Only"
showInContextMenu="true" enableXPath="self::*[local-name()='xref'
and @synctargettext='false' and not(@*[local-name()='read-
only'])]">
  </ExtensibilityMethods>
</Emphasis/>
</EmphasisDef>
```

Configuring the Cross-reference dialog

Cross-references is a highly configurable feature. The potential cross reference target can be configured in **Quark.XA.EI.CrossReferences.dll.config**. Items in the new non-modal cross-reference dialog drop-down lists can be customized. The length of the cross-references listed in the dialog can be also be defined. Authoring is now possible while the dialog is open.

Configuring dialog listings based on target element names (XMLNames) or XPath.

References can be configured in the Cross-reference EI configuration to get desired reference types listed in the cross-reference dialog.

Table 20-1: Configuration attribute values

Parameter	Definition
Name	Attribute Values will be used as names for grouping elements under their value. For example “Configured Numbered List” will be shown in the Reference Type combo box.
TargetElements	XML names of Elements defined in the XAS that need to be listed as reference types are to be added as comma separated values.
xpath	For advanced users, the XPath can be configured to list elements with certain properties.
AllowedProperties	<p>This attribute contains values that are a part of the Insert Reference To field. The following properties are available in the drop-down list:</p> <ul style="list-style-type: none"> • Paragraph number • Page number • Text <p>From these, Text is mandatory and will be part of the selection irrespective of the configuration definition.</p>

Examples

Listing all list items that do not have child elements as placeholders:

Specific xpaths can be mentioned for these cases as follows:

```
<ReferenceType Name ="Configured Numbered List" TargetElements= ""
xpath="/descendant::*[local-name()='oli' and not(descendant::*[local-
name()='placeholder'])]" AllowedProperties="Paragraph number,Page number"/>
```

Note:

1. In Case both **XPath** and **targetElements** are present, the precedence of **XPath** will be higher and the list will be populated on the basis of the **XPath**.
2. Rules defined in the **No Configuration** case section will be equally applicable to the configured reference types.

Configuring dialog listing based on the name of target elements (XML Names) Or XPath.

In the Cross-reference EI config file (Quark.XA.EI.CrossReferences.dll.config), the **References** configuration can be built to get the desired types of references listed in the dialog. The tag **ReferenceType** is used to configure the desired reference types.

```
<ReferenceSection>
  <References>
    <ReferenceType Name ="Configured Numbered List"
```

```

TargetElements= "oli, oli2, oli3, oli-feature" xpath=""
AllowedProperties="Paragraph number, Page number,"/>
</References>
</ReferenceSection>

```

Or

```

<ReferenceSection>
  <References>
    <ReferenceType Name ="Configured Numbered List"
TargetElements= "" xpath="="/>descendant::*[local-name()='oli' and
not(descendant::*[local-name()='placeholder'])]" "
AllowedProperties="Paragraph number, Page number,"/>
  </References>
</ReferenceSection>

```

If no configuration is provided, all elements present in the XML Author document with the following properties will get listed, grouped by their friendly Element Definition names as specified in the XAS:

1. **referenceAttribute** defined with the value **conref** in **element def**.
2. **ID** or the **UniqueIdentifier** attribute is configured for the element in **elementDef**.
3. The element should be configured as **visible**(which is default property of **elementdef**).

Insert Reference To will contain properties such as **paragraph number**, **Page Number** and **Text** by default in the **No configuration** mode.

Table 20-2: Configuring MaxLengthOfTextExtractedFromReference and HrefAttributeOfCrossReferenceNodeName

Parameter	Definition
MaxLengthOfTextExtractedFromReference	This Attribute value signifies the maximum length of a string in Text reference property as shown in a caption text box or a list box. The value is the maximum number of characters allowed.
HrefAttributeOfCrossReferenceNodeName:	This is a required attribute value to enable the cross-references functionality. Its value contains the name of the href attribute name of the xref element as used in the environment under consideration. For example, BUSDOCS uses xlink:href .

Extensibility Interface Use Case Study

Inserting Elements Through EI in Quark XML Author

New elements can be inserted in Quark XML Author in the following ways:

1. Right-click on an insertion point (Empty Style) and select an item from context menu.
2. Select an item from the Styles drop-down list when the selection is at an insertion point (Empty Style).
3. Select a special menu item under the Insert menu or a button on the toolbar. **Insert Table** is one example.
4. Split the current element (if configured in the schema) into two elements via the Enter key. Another variation of this is to type in the empty below an element which supports splits and Quark XML Author will automatically insert the configured Element and paste the text into the newly added element.

The behavior for each of these operations is exactly the same. When a user chooses a particular element to insert through any of the above means, Quark XML Author looks up the XAS for the definition and builds an XML fragment to be attached to the XOM. In addition, the UI is updated to display this new element if it is defined as visible in the schema. Also any required descendants and attributes are also picked up from the schema and inserted. If the XAS specifies default values to certain elements and attributes, Quark XML Author incorporates them into the fragment that it builds.

Another way of adding new content into Quark XML Author is to paste a valid fragment at a valid insertable position. This option is available when (this is one way, there are other ways) a user performs a Component->Copy operation and copies an XML fragment to the clipboard. The paste operation is slightly different from an insert because the paste does not build the fragment from the schema; it already has the fragment, Quark XML Author validates the fragment against its definition specified in the XAS at the current insertable location and only inserts the fragment if it is valid and insertable at the current spot.

This case study discusses an alternate mechanism of inserting new content into an existing document. In the first three scenarios above, Quark XML Author will support addition of new content externally by exposing an Insert hook. This “hook” can be created by adding the externalMethodId attribute to an **insertable**

ElementDef. The fourth scenario, splitting an element, will work just as before—even if there is an externalMethodId specified in the ElementDef of the element being split, the split will ignore the externalMethodId property and insert a fragment from the schema. The insertable requirement means that if an external ElementDef ends up being a required child of another ElementDef, that ElementDef will also have to be defined as external. Quark XML Author will throw an exception if the above condition is not met. This ensures that only insertable elements can have externalMethodId attributes on them. If a potential insertable element contains externalMethodIds in its descendants, the top most externalMethodId is the only one invoked. Thus, this ‘hook’ allows Quark XML Author to essentially perform a paste operation as explained above with an insert operation.

Example 1

Here is a simple ElementDef for an Image element as defined in the XAS defined for DITA:

```
<ElementDef name="image" friendly="Image" visible="false"
style="Heading 8" externalMethodId="getImage">
  <Reference altAttribute="alt">
    <Media xmlns="image" friendly="Image" required="true">
      <Mimetypes>
        <Mimetype type="image/bmp"/>
        <Mimetype type="image/gif"/>
        <Mimetype type="image/jpeg"/>
        <Mimetype type="image/jpg"/>
        <Mimetype type="image/png"/>
        <Mimetype type="image/tif"/>
        <Mimetype type="image/tiff"/>
      </Mimetypes>
    </Media>
  </Reference>
  <Attributes>
    <AttributeType name="href"/>
    <AttributeType name="keyref"/>
    <AttributeType name="alt"/>
    <AttributeType name="longdescref"/>
    <AttributeType name="height"/>
    <AttributeType name="width"/>
    <AttributeType name="align"/>
    <AttributeType name="placement"/>
    &univ-atts;
    <AttributeType name="class"/>
    <AttributeType name="outputclass"/>
  </Attributes>
</ElementDef>
```

As shown above, the image definition specifies an additional attribute called **externalMethodId**. This property holds the value of an EI method which will be invoked when an image element is inserted. Specifying an externalMethodId on an ElementDef implies that that element will always be inserted by this externalMethodId. The only exception as noted above is when the element is created by the “split” operation. The externalMethodId has the responsibility to return an XmlNode which will match the structure specified in the schema for the element being inserted.

Since externalMethodId property can only be applied to insertable elements, this means that any ElementDef which references this (external) ElementDef as a required child will also have to be external.

For example,

```
<ElementDef name="imagemap" friendly="Image Map" visible="true"
style="Tag 2" externalMethodId="getImageMap">
  <Section>
    <Sequence minOccurs="1" maxOccurs="1">
      <SectionType name="image"/>
    </Sequence>
    <Sequence maxOccurs="unbounded">
      <SectionType name="area"/>
    </Sequence>
  </Section>
</Attributes>
...
</Attributes>
</ElementDef>
```

In the above definition, the `imagemap` element has two required child elements (specified by the `minOccurs="1"` value): `image` and `area`. According to the restriction above, the `imagemap` element will also have to be external since it is a referencing image which is a required child and is defined to be external as above. If the `image` were optional, then `imagemap` would not need to be external. If this restriction is not enforced in the schema, Quark XML Author will throw an exception when a document referencing this schema is first opened.

The External Method definition is like any other EI definition. Here is a sample element definition:

```
<ElementDef name="imagemap" friendly="Image Map" visible="true" style="Tag 2"
externalMethodId="getImageMap">
...</ElementDef>
```

The `externalMethodId` references this EI method:

```
<Method id=" getImageMap " assembly=" InsertEI"
class="Invision.Xpress.Extensible.InsertElementManager"
method="GetImageMap"/>
```

The `GetImageMap()` EI (and all other EIs specified in the `externalMethodId` attribute) must return an `XmlElement`. Here is what the `GetImageMap()` method for DITA image element would look like:

```
public XmlElement GetImageMap()
{
  string xml = "<imagemap><image href=\"C:\\..\\My
Pictures\\4.bmp\" height=\"358\" width=\"286\"/> " +
    "<area> " +
    "<shape>Shape</shape>" +
    "<coords >Coordinates</coords>" +
    "<xref >Cross-reference</xref>" +
    "</area>" +
    "</imagemap>";
  XmlDocument doc = new XmlDocument();
  doc.LoadXml(xml);
  return doc.DocumentElement;
}
```

The method simply returns a valid `imagemap` `XmlElement`. The method above does not use any arguments but it can also use any predefined arguments that Quark XML Author exposes. Here is another example:

```
<Method id=" getImageMap " assembly=" InsertEI"
class="Invision.Xpress.Extensible.InsertElementManager"
```

```
method="GetImageMap">
  <Argument type="Filename"/>
  <Argument type="Tag"/>
</Method>
```

Here is how the method looks now:

```
public XmlElement GetImageMap(string filename, string
insertableCaption)
{
  string file = filename; //name of the file that is currently
open
  string insertCaption insertableCaption; //Caption that's shown
to user in the insertables context menu.
  string xml = "<imagemap><image href=\"C:\\..\\My
Pictures\\4.bmp\" height=\"358\" width=\"286\"/> " +
  "<area> " +
  "<shape>Shape</shape>" +
  "<coords>Coordinates</coords>" +
  "<xref>Cross-reference</xref>" +
  "</area>" +
  "</imagemap>";
  XmlDocument doc = new XmlDocument();
  doc.LoadXml(xml);
  return doc.DocumentElement;
}
```

In particular, Argument type="Tag" is a special Argument available for any External method specified in the externalMethodId attribute on an ElementDef. This argument specifies the actual caption that Quark XML Author displays in the context menu that was just clicked. So for example, in Figure 21-1 below, there are two insertable contexts for an image: **Image(Child of paragraph)** OR **Image(Child of Topic Body)**. Both will call the same EI method, getImage. Tag will contain the actual caption which the user clicked. It might be required by an EI for context.

```
<Method id=" getImage " assembly=" InsertEI"
class="Invision.Xpress.Extensible.InsertElementManager" method="GetImage"/>
```

Here is how the method looks:

```
public XmlElement GetImage()
{
  string xml = "<imagemap><image href=\"C:\\..\\My
Pictures\\4.bmp\" height=\"358\" width=\"286\"/> " ;
  XmlDocument doc = new XmlDocument();
  doc.LoadXml(xml);
  return doc.DocumentElement;
}
```

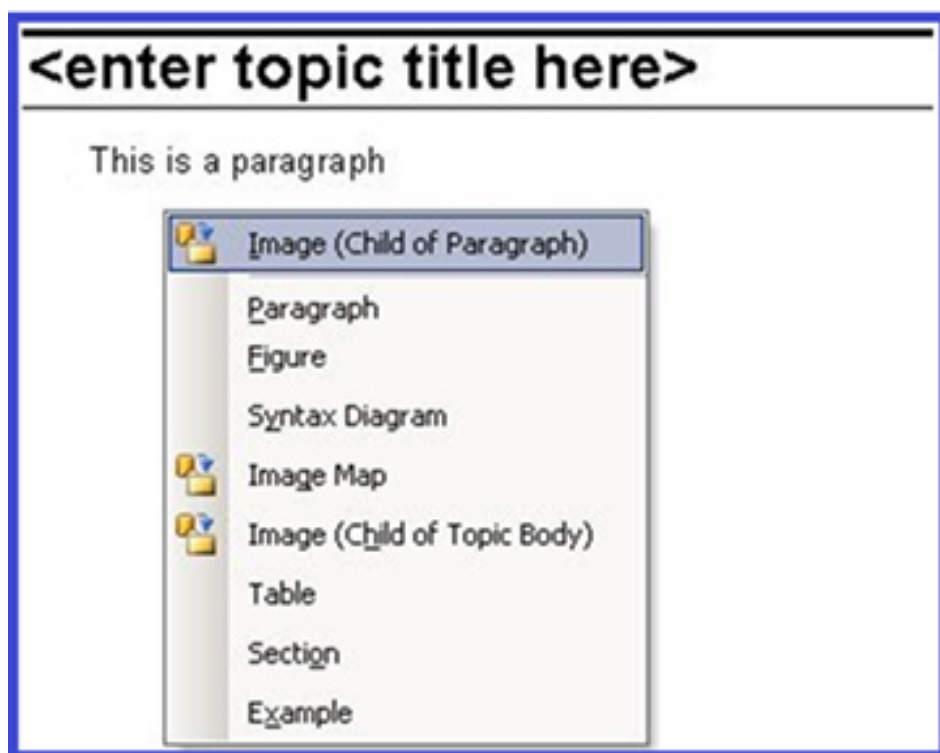


Figure 21-1: Image Contexts

In the above case, if user chooses the Image (Child of paragraph), the Tag argument type will have the value “Image (Child of paragraph)”. This argument is not required to be used in an EI as seen in the first GetImageMap() example.

After the external method returns, Quark XML Author will react as specified below:

1. If the external method returns null OR a non-XmlElement, Quark XML Author will treat that as a user cancel and simply cancel the Insert operation.
2. If the external method throws back an exception, Quark XML Author will consider that as a critical error and show an error dialog with the error message contained in the thrown exception. The Insert operation is cancelled.
3. If the external method returns an XmlElement but the structure of the returned XML does not match the structure specified in the schema specified for that element, Quark XML Author will throw an error dialog and cancel the Insert operation.
4. If the returned XmlElement has a valid structure it is imported into the document and rendered just like a conventional Insert.

Also as seen in the figure above, to indicate that an element is insertable externally, the context menu puts a small icon (with faceID=2109) in front of it.

Also if an externalMethodId is specified on an ElementDef, and if it has a required Choice Collection, the submenu which is normally shown will not be shown. In Figure 21-2, the Image Map element uses the following definition, which does not make an external method call:

```
<ElementDef name="imagemap">
  <Section>
    <Choice maxOccurs="unbounded">
      <SectionType name="image"/>
    </Choice>
  </Section>
</ElementDef>
```

```

    <SectionType name="area"/>
  </Choice>
</Section>
<Attributes>
  ...
</Attributes>
</ElementDef>

```

In Figure 21-3, Image Map uses the following definition, which does make an external method call:

```

<ElementDef name="imagemap" externalMethodId="getImageMap">
  <Section>
    <Choice maxOccurs="unbounded">
      <SectionType name="image"/>
      <SectionType name="area"/>
    </Choice>
  </Section>
  <Attributes>
    ...
  </Attributes>
</ElementDef>

```

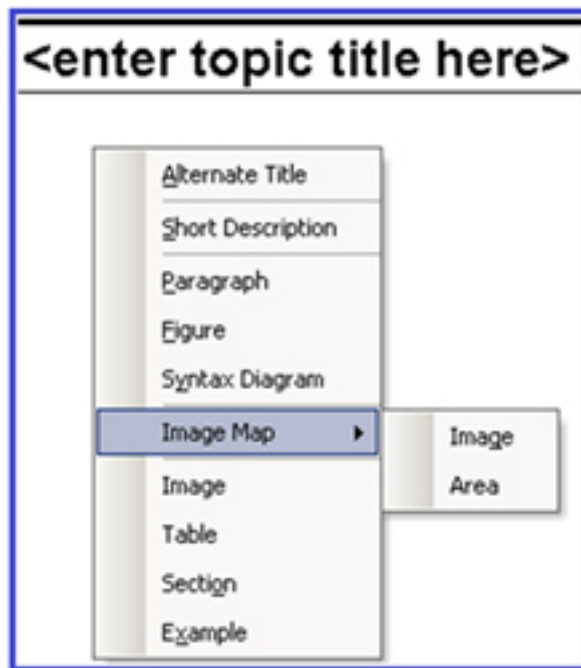


Figure 21-2: Image Map, no EI

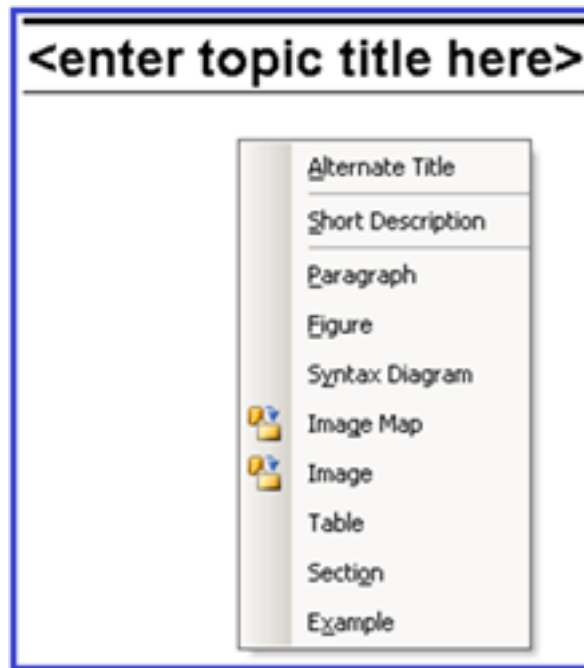


Figure 21-3: Image Map, with EI

In such cases, the EI has the responsibility to show the dialog prompting the user for the child fragment and construct the fragment with the correct child element in it.

Normally, the fragment that is imported externally will be completely editable within the Quark XML Author UI. If there is a requirement for the fragment to be left alone by Quark XML Author for the current editing session, the external method will have to add the `inv:access="read-only"` attribute (where the `inv` prefix refers to the `urn:xpressauthor:xpressdocument` namespace) to every element in that fragment. Here is a simple EI which does just that:

```
public XmlElement GetImageMap()
{
    string xml = "<imagemap
xmlns:inv='urn:xpressauthor:xpressdocument' inv:access='read-
only'><image inv:access='read-only' href='\"C:\\..\\My
Pictures\\4.bmp\" height='\"358\" width='\"286\"/> \" +
    "<area inv:access='read-only'> \" +
    "<shape inv:access='read-only'>Shape</shape>\" +
    "<coords inv:access='read-only'>Coordinates</coords>\" +
    "<xref inv:access='read-only'>Cross-reference</xref>\" +
    "</area>\" +
    "</imagemap>\";
    XmlDocument doc = new XmlDocument();
    doc.LoadXml(xml);
    return doc.DocumentElement;
}
```

This attribute is retained by Quark XML Author within the XOM and is stripped off by Quark XML Author when the document is serialized or when accessing the `ExportedRoot/ExportedCurrentNode` argument in an EI method.

Example 2

In this use case, the cursor is at an Empty and element “p” is valid insertable at this location.

In the first solution, the configuration and code fragment facilitate inserting a paragraph from the Context Menu. In the second solution, the configuration and code fragment facilitate inserting a paragraph from a Ribbon button. Each requires a different solution.

Context Menu

```
<Method id="InsertParaElementFromContextMenu" assembly="EITest"
class="EITest.EITestInsertNode"
method="InsertParaElementFromContextMenu">
    <Argument type="XomRoot"/>
</Method>

<ElementDef name="p"
externalMethodId="InsertParaElementFromContextMenu">
..
</ElementDef>
public XmlNode InsertParaElementFromContextMenu(XmlNode xomRoot)
{
XmlNode newNode =
xomRoot.OwnerDocument.CreateNode(
XmlNodeType.Element, "p", xomRoot.NamespaceURI);
newNode.InnerText = "Some Text";
return newNode;
}
```

Ribbon Button

```
<Method id="InsertParaElementFromMenu" assembly="EITest"
class="EITest.EITestInsertNode"
method="InsertParaElementFromMenu">
    <Argument type="XomPreviousNode"/>
    <Argument type="Delegates">
        <Delegate>InsertNode</Delegate>
    </Argument>
</Method>
<command idMso="Help" visible="xa">
    <!--<ExtensibilityMethod id="About"/>-->
    <ExtensibilityMethod id="InsertParaElementFromMenu"/>
</command>
public void InsertParaElementFromMenu(XmlNode previousNode,
Delegate[] delegates)
{
    InsertNode insertNodeDelegate
= (delegates != null && delegates.Length > 0)
? delegates[0] as InsertNode : null;
    if(insertNodeDelegate != null && previousNode !=
null)
    {
        XmlNode newNode =
previousNode.OwnerDocument.CreateNode
(XmlNodeType.Element, "p", String.Empty);
        newNode.InnerText = "Some Text";
        insertNodeDelegate(previousNode, newNode, null);
// the last parameter is null assuming there is no multiple
contexts // for the fragment being inserted.
    }
}
```

MathType Integration

Implementing MathType Support

Quark XML Author can be configured to support the insertion of Design Science MathType equations. In order to include MathType equations in a Quark XML Author document, MathType must be installed on the local machine. The version of MathType that is required depends on the version of Word and Quark XML Author being used. For information on which version of MathType is required, refer to the Quark XML Author ReadMe documentation.

To configure MathType support, add an element definition that facilitates MathType equations to the XAS.

The Element Definition uses the <OLE> node as a child of <Section>. Details on this node are located in Section 14.5. An example of a MathType element definition is shown below.

```
<!-- MathType support -->
<ElementDef name="content" friendly="Math Equation">
  <OLE readonly="false">
    <MathTypeEquation exportFormat="GIF"
      saveOLEObject="true" translator="MathML2 (no namespace).tdl"
      includeTranslator="true" includeMathTypeData="true"/>
  </OLE>
  <Attributes>
    <!-- Attribute information omitted -->
  </Attributes>
</ElementDef>

<ElementDef name="foreign" friendly="Math Equation Section"
  style="Paragraph_InV">
  <Section>
    <Sequence>
      <OLEType name="content"/>
    </Sequence>
  </Section>
  <Attributes>
    &univ-atts;
    <AttributeType name="class"/>
    <AttributeType name="outputclass"/>
  </Attributes>
</ElementDef>
```

Quark XML Author can read and write both MTEF and MathML data formats.

All error codes returned by MathType are recorded by Quark XML Author and they can be looked up in the MathType SDK help manual. Also, if MathType is not installed, or the installed copy is a prior version or not valid, the presence of a MathType definition in the Quark XML Author configuration files will result in

non-editable images of the equations displayed on the Word canvas. Equation content will be properly preserved and round-tripped.

For a given translator, if you encounter an issue where characters, translator names or MathType data are not saved, please contact Design Science technical support for assistance. If a translator is not producing valid XML, that XML will not be saved by Quark XML Author.

Serialization

The `saveOLEObject` attribute must be used instead of `saveMTEF` to reliably preserve and render MathType equation dimensions.

Developer Notes

Serialization of equations has been changed to use the same methodology as embedded Word documents using Open XML. The attributes for height, width, heightDpi, widthDpi will continue to be serialized, but will be extracted from the Open XML whenever the XML Author document is saved or rendered for viewing. See “Image sizing logic”.

This should help to improve performance for save and open operations and will stabilize XML Author to render correctly sized equations. Do note that there may still be some ‘undefined’ behavior related to equation sizing. This is particularly true of Word 2003 especially when opening the first XML Author instance document during a Word session. Though not as severe as the behavior in 2003, Word 2010 exhibits undesirable, first instance behavior when opening legacy documents where equations exceed the canvas margins (galley).

OLE Word Document Integration

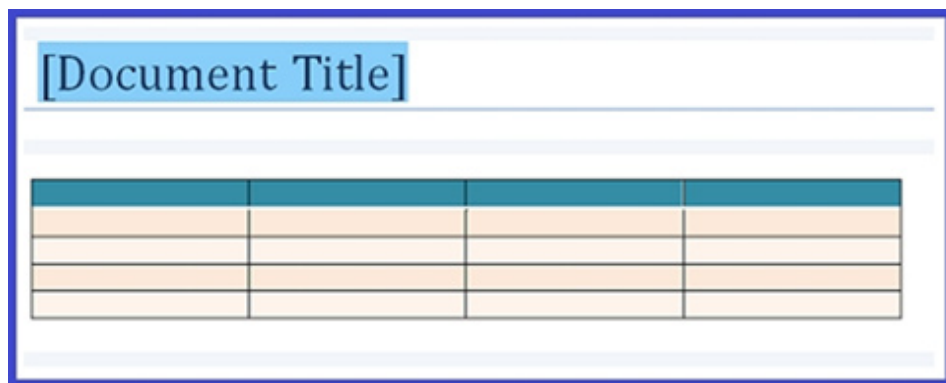
Allows the user to embed a fully functional Word document within an XML Author document. The user can edit the document without restrictions, but the intended use of this feature is to support embedded large/complex tables. The embedded table may be any size with a maximum of 63 columns.

This OLE object can be inserted directly into an XML Author document, but when the user needs to edit, editing is performed in a separate Word window.

All Word table features are available including all tables style features. Banding, rotated text, etc.

Embedded Table User Experience

The user is able to insert Word tables at locations within the document structure that are allowed according to configuration. At these locations, the Context Menu and insertables list displays “Word Table”. When the user inserts a Word Table, the system displays a scaled rendering of the embedded table on the canvas in XML Author.



[Document Title]			

Figure 23-1: New Word Table on the XML Author canvas

The scaled rendering displays the content of the first “printed” page of the table.



Figure 23-2: First page of a Word Table on the XML Author canvas

Editing

To edit, double-click the table image. A new Word window is launched containing the table content.

The Word table is ready for editing.

Viewing

If the XML Author document or the embedded Word document is designated as read-only (in the XML or in the schema definition), the new Word window is launched in a password protected read-only mode.

Copy and Paste

User is able to copy content from other applications, such as Excel, and paste that content into the Word table editor. This allows the user to maintain content in applications other than Word.

Implementing Embedded Table Support

To configure OLE Word Document support, add an element definition that facilitates OLEWordDocuments to the configuration. Because the user can edit the Word document “without restrictions”, the configuration does not have to specify all elements possible, but instead only has to specify table styles and settings and metadata and inline element definitions.

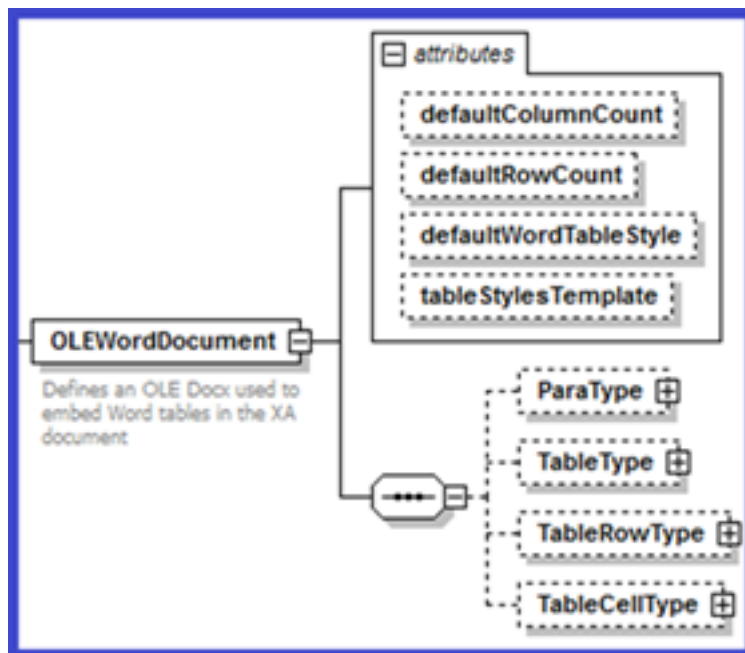


Figure 23-3: Configuration-XSD OLEWordDocument definition

Quark XML Author supports the following OLEWordDocument elements:

ParaType, TableType, TableRowType, TableCellType: used to specify the metadata and inline element definitions for the associated element.

OLEWordDocument contains the following attributes:

The following attributes specify the initial formatting and styling of an embedded Word table. <TableStyles> defined in the configuration are not available.

Table 23-1: OLEWordDocument Attributes

Attribute Name	Required	Definition
defaultColumnCount	no	Default number of columns. Valid values are 1-63. If a number greater than 63 is specified, then 63 is used. If not specified, the default is 3.
defaultRowCount	no	Default number of rows. If not specified or not a valid number, the default is 3.
tableStylesTemplate	no	Specifies the location and filename of the Word template (.dotx file) for custom table styles. The location may be specified as a fully qualified path or as a relative path to the main XML Author template (.dotx file). Only table styles in this tableStylesTemplate are

Attribute Name	Required	Definition
		available to the user in this scenario. <TableStyles> defined in the configuration are not available. However, you can refer to “TableStyle” for an example of the user interface.
defaultWordTableStyle	no	Specifies a style in the tableStylesTemplate that is used as the default table style when a user initially inserts a table. If not specified or if a table style is specified that is not in the template, the system uses the TableGrid style. Built-in Microsoft Word table styles are not directly supported, but you can derive a new custom style from a built-in style.

Starter documents

The custom table style template acts as a starter document for the table because it can include starter content for the embedded Word table. For more information, see “Starter Documents and Templates”. The template may include headers and footers, but these are not rendered on the XML Author Word canvas unless the text boxes are included for content such as page number. It is also recommended that the top and bottom margins of the table template be smaller by about 1/4 inch than the corresponding margins used in the template.dotx in order to minimize the creation of unnecessary blank pages.

Sample configuration

In this sample configuration, the Element Definition uses the <OLE> node as a child of <Section>. For details on this node, see “OLE” and “OLEWordDocument”.

```
<!-- Sections -->
<ElementDef name="body" friendly="Section Body" visible="false">
  <Section>
    <Choice minOccurs="0" maxOccurs="unbounded">
      <!-- Other Choice items elided -->
      <OLEType name="wordTable"/>
    </Choice>
  </Section>
</ElementDef>
```

An example of an element definition using OLEWordDocument is shown below.

When the embedded file is created, it is initialized with a default table according to the configuration-specified attributes.

```
<!-- OLE Table -->
```



```

<ElementDef name="wordTable" friendly="Word Table"
referenceAttribute="conref">

  <OLE readonly="false">

    <OLEWordDocument defaultColumnCount="4"

      defaultRowCount="5"

      tableStylesTemplate="table-template.dotx"

      defaultWordTableStyle="My Style" />

  </OLE>

  <Attributes>

    <AttributeType name="id"/>

    <AttributeType name="conref"/>

  </Attributes>

  &extensibility;

</
ElementDef
>

```

Images

Images that are inserted in the embedded Word document are stored at a location specified in the configuration. See “ImageFilePath”.

Emphasis

The OLEWordDocument element can be configured to allow the user to apply emphasis to text-based elements.

The following shows a emphasized paragraph within an embedded Word document.

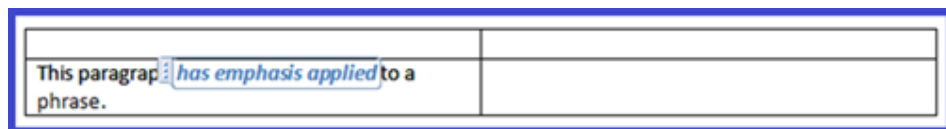


Figure 23-4: A paragraph with emphasis applied

The <ParaType> node contains a collection of <EmphasisType> nodes that specify the allowed emphasis.

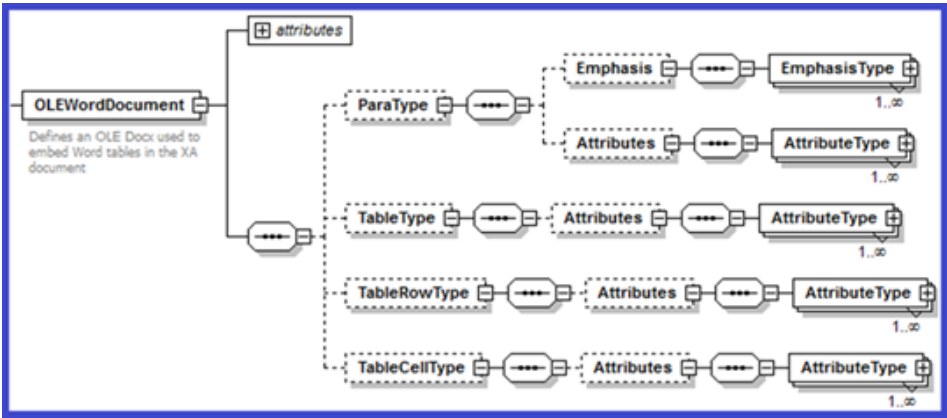


Figure 23-5: Configuration-XSD OLEWordDocument > EmphasisType definition

The <EmphasisType> node contains the following attributes.

Table 23-2: OLEWordDocument > ParaType> EmphasisType Attributes

Attribute Name	Required	Definition
name	Yes	Contains the name of an EmphasisDef defined earlier in the configuration. An EmphasisDef that is defined for an XML Author document may be used/repurposed for this or a separate EmphasisDef may be defined that is only used for OLEWordDocument.
charStyle	Yes	<p>Specifies the “Word character style” that should be applied to text that receives this emphasis. This may be a custom character style that you have put in the “tableStylesTemplate” in <OLEWordDocument>.</p> <p>This only applies to Emphasis in ParaType. The other child elements are table elements that do not directly contain text and therefore do not require this style setting.</p> <p>Word has character and paragraph styles. In the Word Styles tool, the icon to the right is an “a” for character styles or the para marker icon or it can be both. A style can</p>

Attribute Name	Required	Definition
		act as both a character and a paragraph style. A style that is both can be used in our configuration, but not a style that is only a paragraph style.

In the sample configuration, the following emphasis are made available for paragraphs: Intense Reference and Intense Emphasis.

```
<ParaType>
  <Emphasis>
    <EmphasisType name="cite" charStyle="Intense Reference"/>
    <EmphasisType name="example" charStyle="Intense Emphasis"/>
  </Emphasis>
  <Attributes>...</Attributes>
</ParaType>
```

Figure 23-6: Paratype > EmphasisType definitions

Nested emphasis is supported, but configuration rules for what and how are ignored. Style is ignored. Friendly and Attributes defined for the emphasis are used/respected/honored. Read-only settings are honored.

Metadata

The OLEWordDocument element can be configured to allow the user to apply metadata to the following elements:

- Paragraphs
- Tables
- Table rows
- Table cells
- Inline elements

For table, row, cell, inline elements, when the user selects content and applies metadata, XML Author wraps the selection in a Word Content Control. The user sees the Control with its Titletab (Table, row, cell, inline).

For paragraph, XML Author does not use a Word Content Control because it has certain limitations. Therefore, XML Author handles paragraphs internally. No content control is shown on the Word canvas.

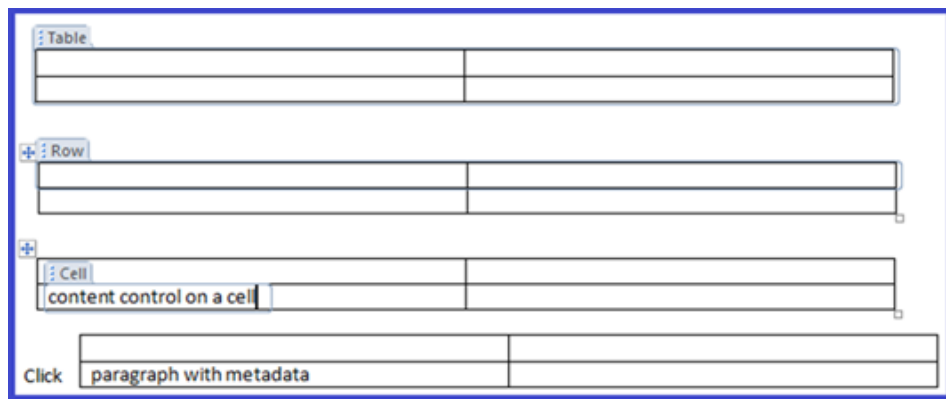


Figure 23-7: Metadata data applied to various element types in an embedded Word document

the table has metadata			
this row has metadata			
this para has metadata	this cell has metadata and some <u>TEXT FROM THE EI</u> emphasis	this paragraph has no metadata	

Figure 23-8: Metadata data applied to various element types in an embedded Word document

The OLEWordDocument element may contain the following child elements which are used for storing metadata: TableType, TableRowType, TableCellType, and ParaType. See “Table” for details on table elements.

These OLEWordDocument child elements are used for tracking metadata that has been applied by the user or the system. When the user right-clicks on something, the feature is invoked. When metadata is applied to a piece of content in the OLE Word document a corresponding element is created in the **Pseudo XOM** and a corresponding Word Content Control (paragraphs are handled differently) is created in the Word document. The metadata that is applied by the user is stored in the attributes of the Pseudo XOM element. What is the Psuedo XOM? It is a collection of nodes similar to the XML Author XOM, but much lighter. It does not track every content element in the embedded Word document. It only contains nodes for content elements to which metadata has been applied. Nodes/ Content Items are created on demand as a result of passing XomCurrentNode to an EI method. In the same way that a custom dialog requires XomCurrentNode to view and edit attributes for the current element. The **RemoveContentItem** delegate is used when the user has deleted all metadata for a content element. See “RemoveContentItem”.

The configuration of the OLEWordDocument child elements includes the definition of each attribute where the metadata will be stored. Each attribute is defined using an AttributeType element. The AttributeType references an AttributeDef already defined in the configuration.

Note that the following are reserved names in the Psuedo XOM: table, row, cell, p.

Each content type node, such as <ParaType>, contains a collection of <AttributeType> nodes that describes the attributes used for storing metadata.

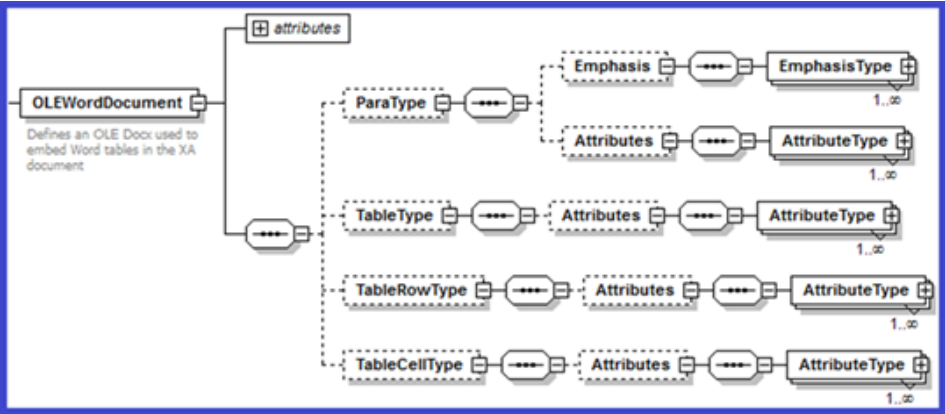


Figure 23-9: Configuration-XSD OLEWordDocument > AttributeType definition

The <AttributeType> node contains the following attribute.

Table 23-3: OLEWordDocument > ParaType> AttributeType Attributes

Attribute Name	Required	Definition
name	Yes	Contains the name of an AttributeDef defined earlier in the configuration. An AttributeDef that is defined for an XML Author document may be used/repurposed for this or a separate AttributeDef may be defined that is only used for OLEWordDocument.

The following example configuration shows the definition of table attributes for storing metadata.

```

<TableType>
  <Attributes>
    <AttributeType name="outputclass"/>
    <AttributeType name="audience"/>
    <AttributeType name="importance"/>
    <AttributeType name="translate"/>
    <AttributeType name="status"/>
    <AttributeType name="rev"/>
  </Attributes>
</TableType>

```

The following example configuration shows the definition of table row attributes for storing metadata.

```

<TableRowType>
  <Attributes>
    <AttributeType name="lang"/>
    <AttributeType name="platform"/>
    <AttributeType name="product"/>
    <AttributeType name="class"/>
    <AttributeType name="outputclass"/>
  </Attributes>
</TableRowType>

```

The following example configuration shows the definition of table cell attributes for storing metadata.

```

<TableCellType>
  <Attributes>
    <AttributeType name="lang"/>
    <AttributeType name="platform"/>
    <AttributeType name="product"/>
    <AttributeType name="class"/>
    <AttributeType name="outputclass"/>
    <AttributeType name="audience"/>
    <AttributeType name="importance"/>
    <AttributeType name="translate"/>
    <AttributeType name="status"/>
    <AttributeType name="rev"/>
  </Attributes>
</TableCellType>-->

```

The following example configuration shows the definition of paragraph attributes for storing metadata.

```
<ParaType>
  <Emphasis>...</Emphasis>
  <Attributes>
    <AttributeType name="id"/>
    <AttributeType name="lang"/>
    <AttributeType name="platform"/>
    <AttributeType name="otherprops"/>
    <AttributeType name="product"/>
    <AttributeType name="class"/>
    <AttributeType name="outputclass"/>
    <AttributeType name="audience"/>
    <AttributeType name="importance"/>
    <AttributeType name="status"/>
    <AttributeType name="rev"/>
  </Attributes>
</ParaType>
```

Metadata can also be applied to inline elements/emphasis.

Metadata default values

The default values specified for the referenced AttributeDefs are used as the default values for the metadata of “new” elements added to an embedded Word document.

For example,

AttributeDef name="lang" default="en"

This is known as *prepopulating*. In addition to prepopulating, the configuration defines what attributes are available for storing metadata and what attributes may be stored during the serialization process. Prepopulating is performed if the element is created by Core XML Author as opposed to an EI. If the element is created by an EI, the EI takes responsibility for which attributes are populated.

Metadata and use cases

Table 23-4: Metadata Use Cases in OLE Word Document

Action	Result
Copy and paste an element containing metadata.	New element is created that contains a copy of the original element's content and its metadata.
Paragraph 2 is merged into paragraph 1. Both paragraphs have metadata.	The content of paragraph 2 is appended to paragraph 1. Paragraph 1's metadata is unaffected. Paragraph 2's metadata is not preserved.

Action	Result
Enter is pressed in a paragraph causing a new paragraph to be created. The original paragraph has metadata.	Metadata is cloned if the cursor was positioned at the start of the paragraph when the user pressed Enter. Otherwise metadata is not cloned.

Smart Content

This section describes Smart Content serialization formats.

Overview

The Smart Content model is designed to allow the user to author content in a non-constrained authoring experience. The content created by the user is serialized to multiple formats.

```
<WordprocessingML>
<image>
<smart-content 2.0>
<smart-content 1.1>
```

WordprocessingML is the primary storage format for the content. This includes content metadata which is stored in a custom XML part within the WordProcessingML.

image is the primary storage format for the content. This includes content metadata which is stored in a custom XML part within the WordProcessingML.

smart content 1.0 & 2.0 are the storage formats that are more easily converted to HTML and other formats that can be consumed by reader applications and publishing engines. These versions of the content only exist in the serialized file and are not needed or used when the document is opened. They are considered disposable. These formats are not configurable. The element definitions are fixed. 1.0 tables are similar to CALS. 2.0 tables are Word's native XML. These table structures are fixed for table, row, cell, paragraph and will be whatever is appropriate for CALS or Word's native XML. However, the inline elements are custom and those tagnames come from the EmphasisDef. For example, "example", "cite".

Serialization

The serialization process is a two pass process for both smart content flavors (1.1 and 2.0).

First pass

Transform engine exports the basic structure of the content (without metadata). Into that structure, bread crumbs are inserted that specify the locations of content controls. In other words, these specify which content elements were being tracked because they have had metadata applied to them.

Second pass

The metadata that was being stored/tracked in the PseudoXOM is applied to the respective elements in the smart content structure (at the bread crumb locations). The bread crumbs are then removed from the smart content.

Metadata on a block element is serialized as attributes on the element in the Smart Content output. Metadata on an inline element is serialized as an actual element around the text.

Word Tables

The following example provides additional details on the serialization of an embedded Word table.

Each OLEWordDocument object found in the XML Author document is serialized using the following format:

```
<OLEWordDocument element="">
  <WordprocessingML> Base-64 encoded contents of the word binary
  representation of the OLE object.</WordprocessingML>
  <image dxaOriginal="width in twips" dyaOriginal="height in
  twips" defaultStyle="some styling characteristics ">
    base 64 encoded emf representation of the OLE object.
  </image>
</OLEWordDocument element>
```

Given an embedded Word Table in an XML Author document.

```
<MyOLEWordDocument>
  <WordprocessingML> Base-64 encoded contents of the word binary
  representation of the OLE object </WordprocessingML>
  <image dxaOriginal="width in twips" dyaOriginal="height in
  twips" defaultStyle="some styling characteristics ">
    base 64 encoded emf representation of the OLE object.
  </image>
  <smart-content xmlns="urn:schemas-quark-com:smart-content:2.0">
    <tbl>
      <tblPr/>
      <tblGrid/>
    </tbl>
  </smart-content>
  <smart-content xmlns="urn:schemas-quark-com:smart-content:1.1">
    <table>
      <tgroup/>
    </table>
  </smart-content>
</MyOLEWordDocument>
```

The two smart-content nodes are the resultant of calls to the TransformEngine. The content returned by the TransformEngine is of the smart content 2.0 format and has to be filtered by XML Author to create the required output.

```
<topic xsi:schemaLocation="urn:schemas-quark-com:smart-content:2.0 Smart-
Content%20(topic).xsd" xmlns="urn:schemas-quark-com:smart-content:2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <title/>
  <body>
    <tbl>
      <tblPr/>
      <tblGrid/>
    </tbl>
  </body>
</topic>
```

The same is true for the smart content 1.0 output format.

```
<topic xsi:schemaLocation="urn:schemas-quark-com:smart-
content:1.1 Smart-Content%20(simple%20topic).xsd"
xmlns="urn:schemas-quark-com:smart-content:1.1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <title/>
  <body>
    <table>
      <tgroup cols="">
        <colspec></colspec>
        <tbody>
          <row>
            <entry></entry>
          </row>
        </tbody>
      </tgroup>
    </table>
  </body>
</
topic
>
```

Why are there empty values in the smart content?

During serialization of regular XML Author content, attributes without values, empty strings, are intentionally not serialized (cleanup). This applies to attributes that XML Author manages, that are in the configuration. Unmanaged attributes are allowed and are serialized. XML Author does not modify them in any way. However, during serialization of that embedded Word documents, this “cleanup” of empty attributes is not performed.

Serializing Metadata

This section discusses the serialization of metadata for various table elements.

The metadata that is serialized is based on the attributes defined for each element type in the OLEWordDocument definition.

For example, the following example configuration shows the definition of table attributes for storing metadata.

```
<TableType>
  <Attributes>
    <AttributeType name="outputclass"/>
    <AttributeType name="audience"/>
    <AttributeType name="importance"/>
    <AttributeType name="translate"/>
    <AttributeType name="status"/>
    <AttributeType name="rev"/>
  </Attributes>
</TableType>
```

The following shows the serialization of table-level metadata.

```
<smart-content xmlns="urn:schemas-quark-com:smart-content:1.1">  
  <table frame="all" colsep="1" rowsep="1" outputclass=""  
    audience="my audience" importance="Recommended" incrementer="2">  
    <tgroup cols="4">
```

Images

The Word canvas

If an image is greater than available canvas real estate, then the image is resized to fit within the available canvas.

Image sizing logic

Multiple changes have been made with regard to the logic used to obtain and serialize image dimensions.

- Implemented logic to normalize DPI metrics as soon as an image is inserted into an XML Author document. Since DPI is serialized as an integer value, all DPI will be managed internally as integers thus eliminating the need to round DPI values on demand.
- Implemented new method to normalize point values to a granularity of 1/20 point. This is used following resize to limit rounding errors thus preventing dimensional drift.
- Resize logic was modified to use normalized point size of inline shapes following resize. This helps to limit rounding problems and to stabilize dimensional drift.
- Internal references and definition for DPI variables, properties, etc. were modified to change the data type from floating point to integer for the purpose of limiting rounding errors which cause image dimensions to drift upon save, close, reopen cycles.
- The point / pixel conversion logic was refactored to properly round float values during conversion. This is used during operations where images are inserted, refreshed, resized or replaced to limit rounding errors thus preventing dimensional drift.

Rounding Error Issue

Current image size behavior in XML Author is governed by actual image metrics that are extracted from the image binary itself wherever possible. All supported graphics files that contain image metrics use the same data types.

Height and width (pixels) are stored as an integer data type.

Horizontal and vertical resolution (DPI) are stored as a float data type.

However, meta-files (EMF, EMZ, WMF, WMZ) can store horizontal and vertical resolution as non-integer values. For example, 599.501. All other graphic formats store resolution as whole number values. For example, 600.000.

Facts and conversion factors:

- $\text{DPI} = \text{dpi} = \text{dots per inch} = \text{dots} / \text{inch}$
- 1 inch = 72 points

Microsoft measures the physical size of images in points.

Persistence and certain calculations require pixels and dpi, therefore we must convert back and forth between Microsoft points and pixels-dpi.

Point values may change when images are inserted, resized, refreshed. At the time of these events, point values are rounded to the nearest 1/20 or 0.05 so that point values never deviate from the necessary degree of precision required.

Precision requires that we manage values as integers, this requires that we round resolutions that are not whole numbers to be whole numbers. This only applies to meta-files (EMF, EMZ, WMF, WMZ).

The image metrics must be serialized as integer values using pixels and dpi as the unit of measure for consumption by other downstream processes.

Some precision is lost due to rounding, but rounding is required to maximize the consistency of calculations across all graphic types.

To inspect the image metrics of a given image file use Microsoft Office Picture Manager that is installed in the Office Tools folder. Do not use Microsoft Paint for this purpose. It is unreliable.

Implementing a Custom XML Resolver

Implementation Overview

Although Quark XML Author uses an internal XML Resolver to resolve such items as DTD declarations and Schema references, it may be necessary to implement a custom resolver.

A custom resolver is implemented via an external DLL that is called by an Extensibility Interface method. The custom resolver must implement the `IXmlResolverProvider` interface, shown below.

```
public interface IXmlResolverProvider
{
    XmlResolver GetResolver(XmlAuthorOperationType opType, string
input);
}
```

The `XmlAuthorOperationType` is an enumeration that denotes what operation is being performed, allowing the `IXmlResolverProvider.GetResolver` method to determine the appropriate `XmlResolver` to return:

```
public enum XmlAuthorOperationType
{
    Transform,        // import/export operations, such as when a
document is first loaded.
    Internal,         // All internal operations.
    XSLTTransform     // Used for loading XSLT stylesheets
}
```

Transform: This is specified when an initial document is loaded and a stylesheet is specified in the AppConfig's namespace section. It is also used when a document is exported either via the Save command, or Create/Display Renditions.

Internal: This is used for all internal processing. The interpretation of `XmlAuthorOperationType` is entirely dependent on the custom use case for which this Resolver is designed. It can choose to totally ignore the type of operation for which the resolver is requested and return a single resolver instance.

XSLTTransform: This is used when an XSLT transform is loaded:

```
XslCompiledTransform xsl = new XslCompiledTransform();

xsl.Load(xslPath, new XsltSettings(true, true),
resolverProvider.GetResolver(XmlAuthorOperationTypes.XSLTTransform));
```

A custom resolver DLL consists of at least two parts: the implementation of the `IXmlResolverProvider` interface and the class and method that will be called from the Quark XML Author Extensibility Interface. An example is shown below.

```
Public Class MyResolver
{
    public void GetACustomResolver(Delegate[] delegates)
    {
        //call the delegate here with a new instance of the
        IXmlResolverProvider implementation.
        SetXmlResolver sxr = delegates[0] as SetXmlResolver;
        Sxr(new MyXmlResolverProvider());
    }
}

//Sample custom resolver class that returns a XmlUrlResolver for
all Quark XML Author internal operations and null otherwise..
Public class MyXmlResolverProvider : IXmlResolverProvider
{
    Public XmlResolver GetResolver(XmlAuthorOperationType opType,
string input)
    {
        //return new XmlUrlResolver();
        If(opType == XmlAuthorOperationType.Internal)
            Return null; //No resolver for Quark XML Author
internal operations.
        Else
            Return new XmlUrlResolver(); //return URL
resolver for all other cases
    }
}
```

Finally, to implement the custom resolver, the EI method must be defined to call it. Typically, this method would be referenced in the `<Connect>` event in `AppConfig.xml` where this resolver will be used to set a global Resolver for all documents opened in Quark XML Author. An example is shown below.

```
<Method id="GetResolver" assembly="CustomResolver" class="MyResolver"
method="GetACustomResolver">

    <Argument type="Delegates">

        <Delegate>SetXmlResolver</Delegate>

    </Argument>

</Method>
```


How Microsoft and Quark Support Languages and Cultures

The Microsoft .Net Framework organizes language support in the following way. Each language has a main language resource assembly. For example, English (en). Within each language, there may be culture / locale specific versions of a language resource assembly. For example, United States English (en-us) or British English (en-gb). Quark XML Author provides the same support for main language and locale specific language resources.

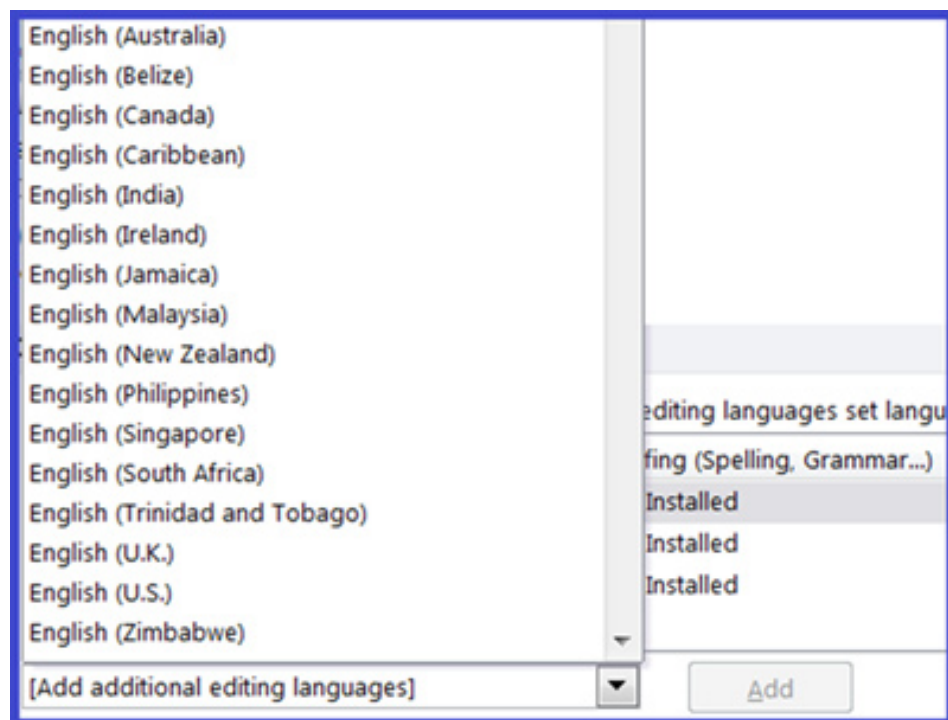


Figure 27-1: Locale specific versions of English supported by .Net

In addition, we need to differentiate between the Display Language and Editing Language within the context of Microsoft Office applications. The Display Language is the language displayed in the user interface (UI) of the Microsoft Office application. This includes menu items, toolbar buttons, ribbons, ribbon buttons, backstage views and Help. The Display Language only supports main language resources such as English (en), but not locale specific language resources such as United States English (en-us). The Editing Language is the language of the content that you are authoring on the Word canvas. This includes language-specific features such as dictionaries and grammar checking. The Editing Language supports locale specific language resources, "Figure".

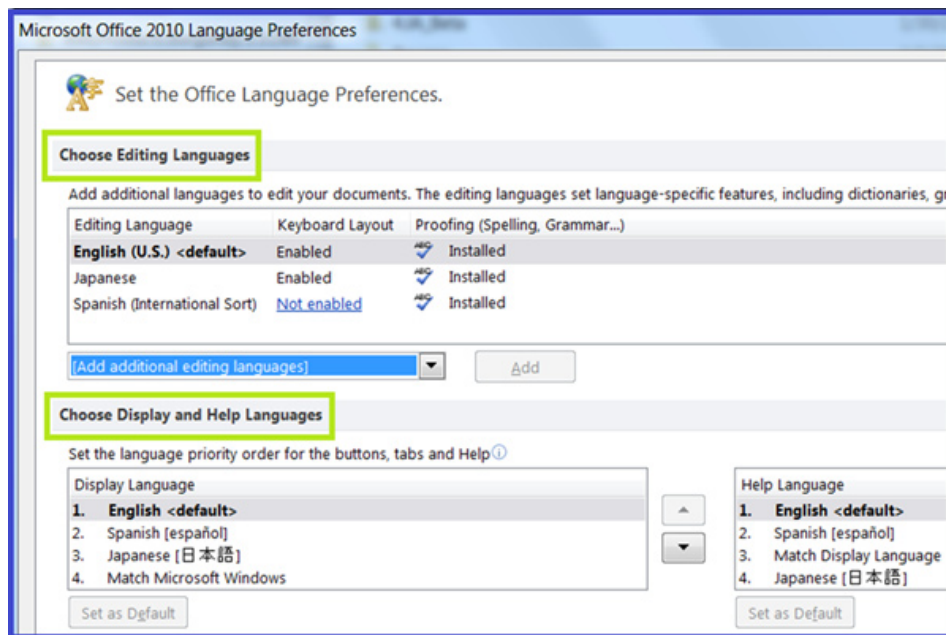


Figure 27-2: The user can specify both an editing language and a display language

Microsoft and Quark XML Author support using one language for the Display Language and a different language for the Editing Language. However, Quark XML Author does not support the authoring of multiple languages within a single Quark XML Author document.

Presently, Quark XML Author is distributed with the following Display language resources:

- English main language
- Spanish main language
- Japanese main language
- French main language

Quark XML Author supports the following languages:

Non-IME:

- English
- Spanish
- French
- Japanese
- Chinese (Traditional)
- Chinese (Simplified)
- Korean
- Vietnamese
- Arabic

- Hebrew

Support for these languages must be enabled in Microsoft Word. East Asian languages will also require IME to input characters.

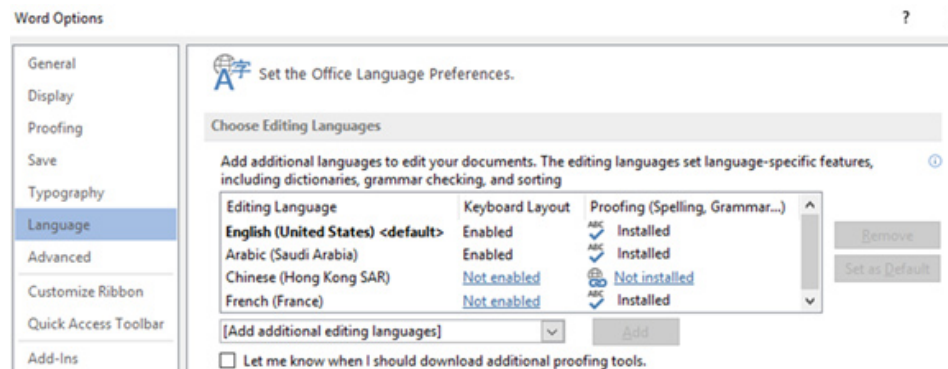


Figure 27.3: Choosing an editing language

Bidirectional language support

Bidirectional languages (right-to-left languages) refers to any writing system that is written from right to left and includes languages that require contextual shaping, such as Arabic, and languages that do not. Right-to-Left languages are written and read mainly from right to left, but some portions of the text, such as numbers and embedded Latin languages (e.g. English) are written and read left to right.

Bidirectional language support in Microsoft Office

Microsoft Office supports right-to-left functionality and features for languages that work in a right-to-left (or a combined right-to-left, left-to-right) environment for entering, editing, and displaying text.

Bidirectional language support in Quark XML Author

XML Author supports authoring in LTR and RTL layouts depending on the default editing language set in MS Word.

To support Bidirectional Text in XML Author, the following option must be disabled in Microsoft Word:

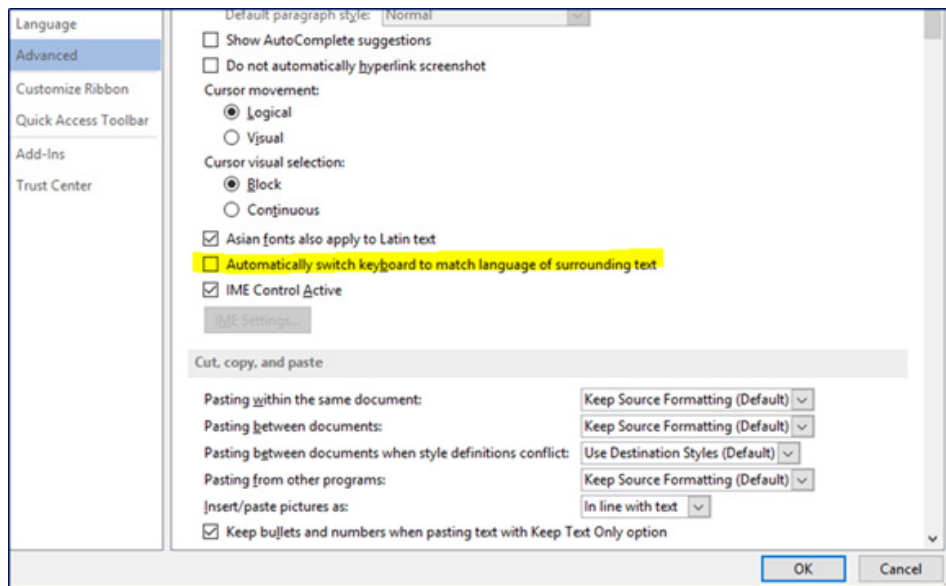


Figure 27.4: The option to automatically switch keyboard to match language of surrounding text must be disabled.

Configuring the XAS to allow support of direction attributes for elements

The Quark XML Author structure must be configured to allow support for direction attributes of elements. Use the `directionAttribute` datatype to enable Quark XML Author to support bidirectional text.

An example is shown below:

```
<Attributes>
...
<AttributeDef name="dir" datatype="directionAttribute"
visible="false"/>
<AttributeDef name="tablestyle" datatype="tableStyle" friendly=""
visible="false"/>
...
</Attributes>
The following corresponding changes are required in the
XpressSchema.xsd file:
<xs:attributeGroup name="fieldAttributes">
...
  <xs:enumeration value="directionAttribute"/>
...
</xs:attributeGroup>
<xs:attributeGroup name="tableStyle">
...
  <xs:attribute name="tableDirection" type="xs:string"
use="optional" />
...
</xs:attributeGroup>
```

Configuring the user interface for bidirectional language support

The `dir` attribute sets the text direction for elements regardless of the text direction of the document. Once this attribute is set, UI controls can be used to set text direction.

For example, the **Text Direction** buttons can be configured in config.xml to facilitate the insertion of elements with a right-to-left or left-to-right text direction, as in Microsoft Word.

```
<buttonGroup id="TextDirection">
  <toggleButton id
="TextDirectionLeftToRight"imageMso="TextDirectionLeftToRight">
    <InternalClass name="ToggleReadingDirectionControl"
directionAttribute="ltr" />
  </toggleButton>
  <toggleButton id="TextDirectionRightToLeft"
imageMso="TextDirectionRightToLeft">
    <InternalClass name="ToggleReadingDirectionControl"
directionAttribute="rtl" />
  </toggleButton>
</buttonGroup>
```

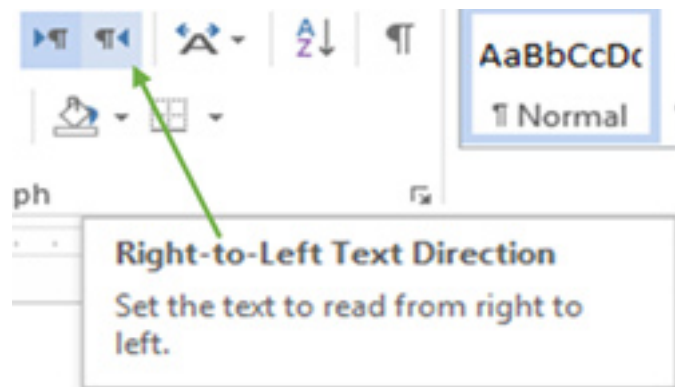


Figure 27-5: text direction buttons.

Configuring bidirectional language support at the content level

Use the **TextDirection** formatting type to format attributes that will allow different text runs within a single paragraph.

Example:

```
<Formatting>
...
<TextDirection name="ph" friendly="direction"
TextDirectionAttributeName="dir"/>
...
</Formatting>
```

The following changes must be made to the XpressSchema.xsd file:

```
<xs:element name="XpressSchema">
  <xs:complexType>
    <xs:sequence>
      ...
      ...
      <xs:element name="TextDirection" minOccurs="0">
        <xs:complexType>
          <xs:attribute name="name" type="xs:string"
use="required"/>
          <xs:attribute name="friendly" type="xs:string"
use="optional"/>
          <xs:attribute name="TextDirectionAttributeName"
```

```

type="xs:string" use="required"/>
    ...
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

Configuring bidirectional language support at the element level

Text runs will be created for blocks of text whose text direction is different as compared to the element.

Root snippet:

```

<topic id="" xmlns:inv="urn:xpressauthor:xpressdocument"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xsi:noNamespaceSchemaLocation="BUSDOCS.xsd" xml:lang="ar"
dir="rtl">
<title/><body/></topic>

```

The XML snippet generated for left-to-right text in a paragraph:

```

<body>
  <p widow-orphan="true" xml:id="IDd3ab0288-d634-4e82-90b1-
c1ecd1b3ee5f">شحنة شمس <ph dir="ltr"> sample text </ph>
</p>
</body>

```

Add the following to the element definition to support multi-directional text runs:

```

<Formatting>
  -----
  <FormattingType name="ph"/>
</Formatting>

```

Configuring bidirectional language support at the document level

To configure bidirectional language support at the document level, see [dir](#).

IME based:Supporting New Languages

Quark XML Author supports localization of the software to any required language that is supported by Microsoft Word. This section describes the steps that must be performed for Quark XML Author to support a new Display Language and who is responsible for each step.

What Must Be Translated

The strings that are displayed in Word UI controls are stored in language specific resource files. These strings can be translated to new languages. These UI controls include:

- Menu items
- Buttons
- Ribbons
- Forms
- Dialogs
- Error Messages
- Context Menus

Translation Process Overview

The following swim lane diagram provides an overview of the translation process.

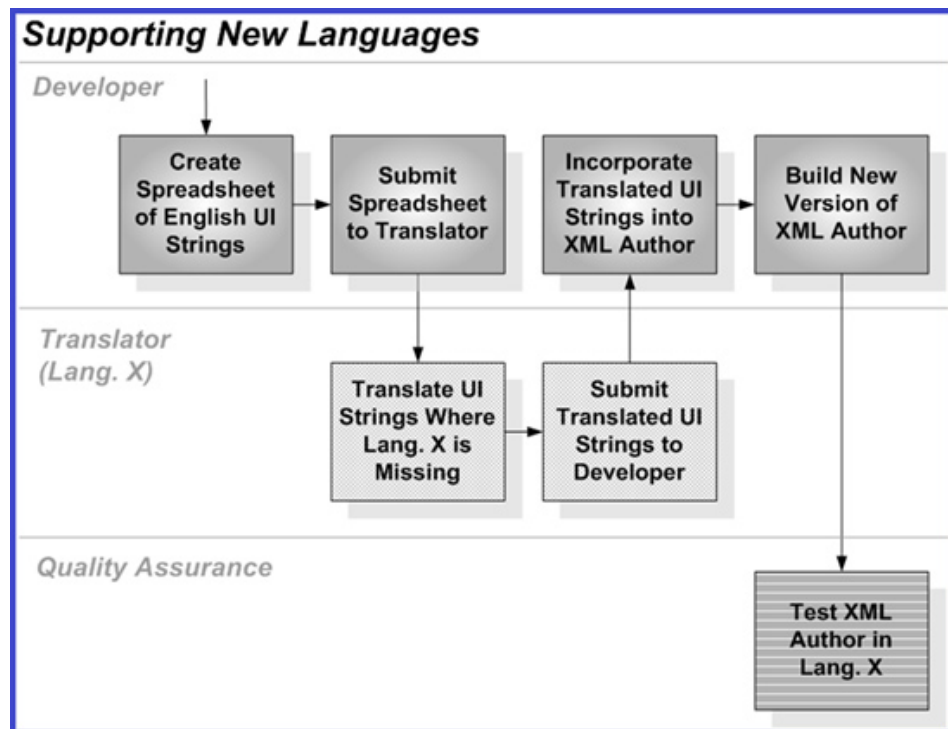


Figure 28-1: Translation process overview

(External)

The role of Translator may be performed by Quark's current contracted translation service or by a Quark partner. A Quark Developer or Translation Coordinator provides a spreadsheet which requires translation to the Translator. From the Translator's perspective, the process is straightforward in that they receive a spreadsheet essentially containing the following columns:

- Id
- English
- Translation
- Comments

The Translator is responsible for completing the Translation column. Because some of the UI strings are acquired directly from Microsoft Word in the new language, these strings are already completed and stored in the Translation column. For example the UI strings for the Word features, Spell Checker and Find & Replace. These completed strings do not need to be verified by the Translator. The Translator is only responsible for the providing the translated string where a translated string is not present; in other words, where cells in the Translation column are empty.

Icons (Internal)

See "Icons" and "Custom Icons".

Shortcut keys (Internal)

Quark Development acquires the shortcut keys for the new language from Microsoft Word and its documentation. Based on this information, Quark Development updates the shortcut keys in the Quark XML Author global shortcut keys configuration file and DITA for Business Documents configuration for the new language.

Professional services should refer to those efforts when creating other Quark XML Author solutions in the new language. And if needed, Development can provide to Professional Services a listing of all Microsoft Word shortcut keys in the new language.

Friendlies (Internal)

In configurations, strings for friendlies must be translated.

Localization: A step-by-step example

Localizing Quark XML Author for any supported language is a straightforward process. This section contains an example that describes the localization procedure for Quark XML Author.

Pre-requisites

Localizing Quark XML Author requires the following components and software:

- An installed copy of Quark XML Author.
- A decompilation tool for the Microsoft .Net library.
- A resource generation tool and an assembly linker tool. Please visit the following URLs for more information about these tools:
- Assembly Linker: <https://docs.microsoft.com/en-us/dotnet/framework/tools/al-exe-assembly-linker>
- Resource Generator: <https://docs.microsoft.com/en-us/dotnet/framework/tools/resgen-exe-resource-file-generator>
- A tool to modify the strings contained in the resource files. You may want to try the .NET Resourcer by Lutz Roeder for this purpose or any other tool that you are comfortable with. The .NET Resourcer can be downloaded from the following URL: <https://www.lutzroeder.com/dotnet/>

The localization procedure

Additional language support needs to be added to the existing installation of Quark XML Author to add resources for a new language.

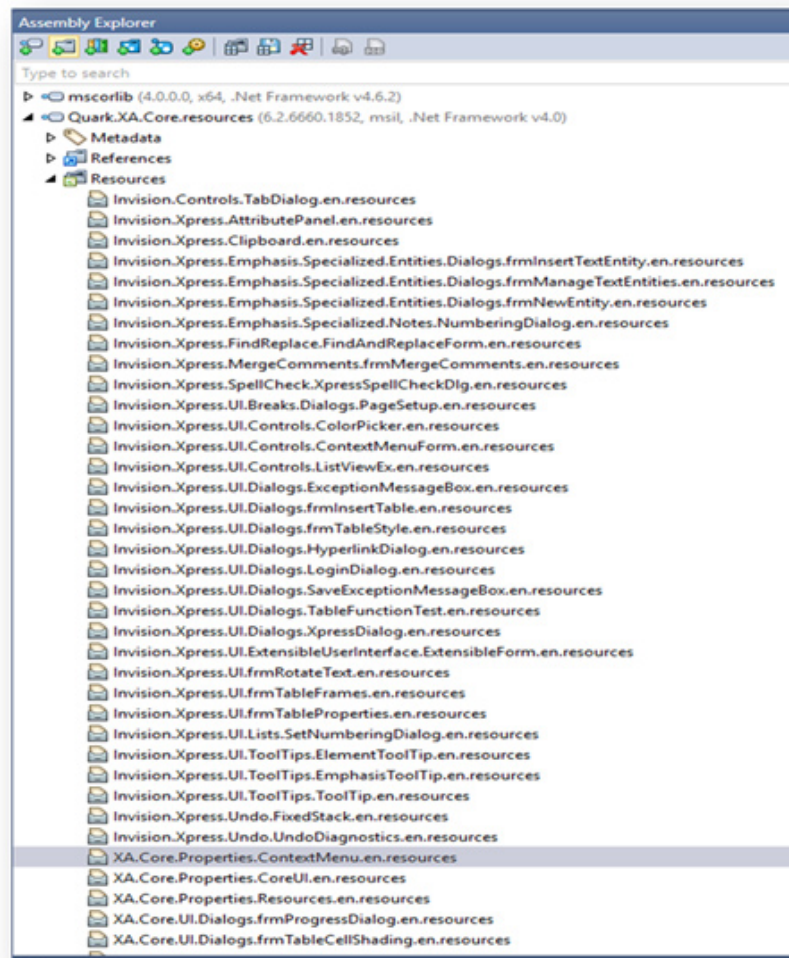
To add support for a new language:

1. Open the Quark XML Author installation folder.

2. Duplicate the **en** folder at this location and rename it to the desired locality. For example, if you wish to add support for Chinese, rename the duplicated **en** folder to **zh**. The **zh** folder will now contain the same resource files as the **en** folder.

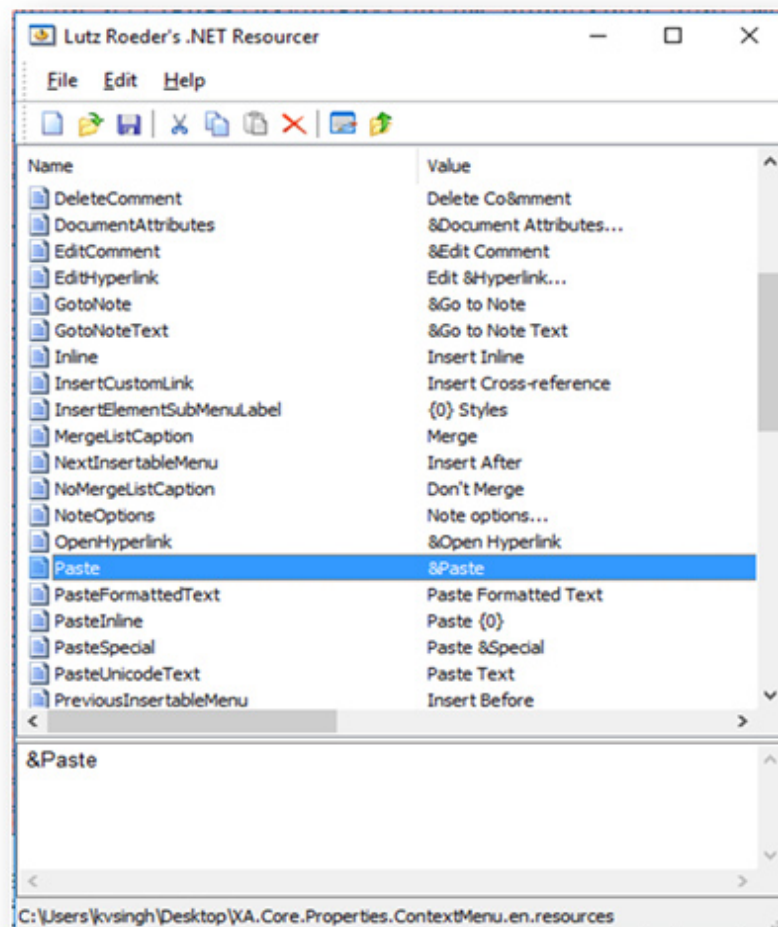
The zh folder will contain many resource files. The translations for all resources contained in these files need to be added for each file. For this example, see the localization procedure for **Quark.XA.Core.Resources.dll**.

3. Using the **Decompiler** tool, decompile **Quark.XA.Core.Resources.dll**. Once this file is decompiled, all the resources available in this file will be displayed:



4. Localize all of the resources available in the **zh** folder.
 - a) Extract the resource file and save as either a **.resources** file or a **.resx** file. (Files with the **.resources** extension can be localized using the **.NET Resourcer** or any other tool of your choice. Files with the **.resx** extension can be opened using any XML editing tool of your choice.) The available resources can then be

localized.



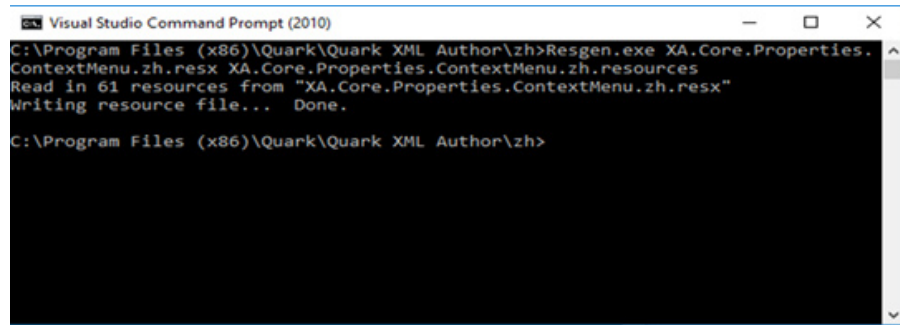
b) Save the file with a culture-specific language prefix in the name. For example the `XA.Core.Properties.ContextMenu.en.resources` file for Chinese will be renamed to `XA.Core.Properties.ContextMenu.zh.resources` or `XA.Core.Properties.ContextMenu.zh.resx`.

5. All files that have been extracted and saved as `.resx` must be converted to the `.resources` format. The conversion will be accomplished using the **resgen** utility. Repeat the following steps for all resource files for each dll.

a) Type the following command in the command prompt to convert `XA.Core.Properties.ContextMenu.zh.resx` to `XA.Core.Properties.ContextMenu.zh.resources`:

*Resgen.exe <Resource File to be converted> <Target Resource File type with extension>.*For example:

```
Resgen.exe XA.Core.Properties.ContextMenu.zh.resx
XA.Core.Properties.ContextMenu.zh.resources
```



```

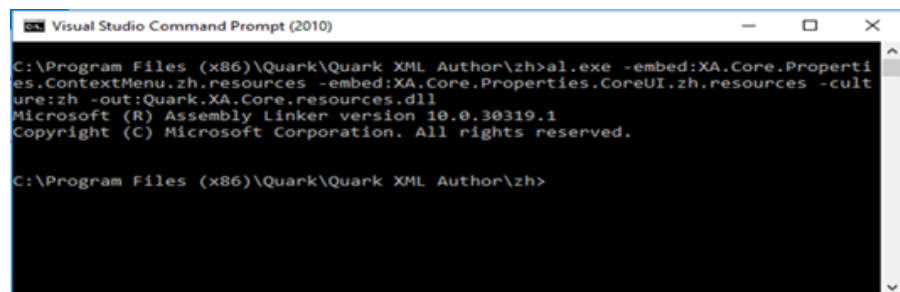
C:\Program Files (x86)\Quark\Quark XML Author\zh>Resgen.exe XA.Core.Properties.
ContextMenu.zh.resx XA.Core.Properties.ContextMenu.zh.resources
Read in 61 resources from "XA.Core.Properties.ContextMenu.zh.resx"
Writing resource file... Done.

C:\Program Files (x86)\Quark\Quark XML Author\zh>

```

b) Once all the resource files are available, recreate the resource dll using the assembly linker. To use the assembly linker, execute the following command in the Visual Studio Command Prompt: `al.exe -target:lib -embed:<ResourceFile1> <ResourceFile2> -culture:<cultureName> out:<outputdll>` For example :

`al.exe -target:lib -embed:XA.Core.Properties.ContextMenu.zh.resources -
embed:XA.Core.Properties.CoreUI.zh.resources -culture:zh -
out:Quark.XA.Core.resources.dll`



```

C:\Program Files (x86)\Quark\Quark XML Author\zh>al.exe -embed:XA.Core.Proper
ties.ContextMenu.zh.resources -embed:XA.Core.Properties.CoreUI.zh.resources -cult
ure:zh -out:Quark.XA.Core.resources.dll
Microsoft (R) Assembly Linker version 10.0.30319.1
Copyright (C) Microsoft Corporation. All rights reserved.

C:\Program Files (x86)\Quark\Quark XML Author\zh>

```

6. Relaunch Quark XML Author.

Variable Reference Support

Reference Type and Section Type block level elements in Quark XML Author can now refer to variable content references.

Section Type Elements

For section type elements such as Lists, Tables, Paras and Container elements, the variable content reference attribute has been kept configurable.

The element defines the attribute that can hold variable content references. For DITA elements the attribute will be “@conkeyref”.

Example:

Configuration of the Block level element Para to be able to refer variable content references in addition to direct content references:

```
<ElementDef name="p" xmlns="p" friendly="Paragraph"
referenceAttribute="conref"
variableReferenceAttribute="conkeyref" style="Normal (XA
Heading)" calloutAttribute="outputclass">
  <Section>
    <Para defaultHyperlink="xref@format='html'">
      ...
    </Para>
  </Section>
<Attributes>
  <AttributeType name="conref"/>
  <AttributeType name="conkeyref"/>
  <AttributeType name="outputclass"/>
  .....
</Attributes>
</ElementDef>
```

Reference Type Elements

For Reference Type Elements, such as DITA Topic Refs, **keyref** has been assumed as the variable reference attribute and “href” for direct referencing. The serialized name, however, is still configurable.

Example:

```
<ElementDef name="refLabel" xmlns="refLabel">
  <Reference>
    <ComponentType name="reftitle"
style="XA_Body Text Indent"

excludeFromComponentContextMenu="true" xpath="//title[1]"/>
  </Reference>
```

```

        <Attributes>
            <AttributeType name="format"
fixed="dita"/>
            <AttributeType name="navtitle"
friendly="Title"/>
            <AttributeType
name="href" visible="false"/>
            <AttributeType
name="keyref" visible="false"/>
        </Attributes>
</ElementDef>

```

Additional delegates

The following new delegates have been exposed to allow EIs to add and manage the reference variables for a Quark XML Author document.

- AssignRefVariable
- GetRefVariable
- GetAllRefVariables
- GetExportedXMLForRefVariables
- RemoveRefVariable

The Quark XML Author Reference Resolution Engine now honors these variables while resolving variable content references. The reference resolution gives preference to Variable references (if the variables are defined) over direct references.

Installation of Quark XML Author

This section provides details on how to install Quark XML Author.

Installation files are provided on a distribution media such as a CD or network folder.

Automated installation on a new machine:

1. Navigate to the appropriate folder.
2. Execute **Setup.exe**.

Or for manual installation on a new machine:

Install in the following order:

1. If Quark XML Author version 4.0 or higher has been installed on the machine, proceed to Step 5.
2. Navigate to the **WindowsInstaller4_5** folder. For a give operating system, select the specified installation item.
 - Windows Server 2003 Service Pack 1, Windows Server 2003 Service Pack 2 and Windows XP (64-bit)
 - x86 Platform: **WindowsServer2003-KB942288-v4-x86.exe**
 - x64 Platform: **WindowsServer2003-KB942288-v4-x64.exe**
 - Windows XP Service Pack 2 and Windows XP Service Pack 3 (32-bit)
 - x86 Platform: **WindowsXP-KB942288-v3-x86.exe**
 - Windows Vista, Windows Vista Service Pack 1 and Windows Server 2008
 - x86 Platform: **Windows6.0-KB942288-v2-x86.msu**
 - Windows 7 and Windows Vista Service Pack 2
 - No update required
3. Install Visual C++ Runtime Libraries (x86) Redistributable:
 1. Navigate to the **vc redistrib_x86** folder.
 2. Execute **vc redistrib_x86.exe**.
4. Install Microsoft Knowledge Base Article 908002 patches:

1. Navigate to the KB908002 folder.
2. Execute `extensibilityMSM.msi`.
3. Execute `lockbackRegKey.msi`.
5. Install Quark XML Author:
 1. Navigate to the appropriate folder.
 2. Execute `XML_Author.msi`.

Appendix A – Color Names

This appendix lists all valid color names for use with the foreColor and backColor Emphasis definition attributes.

AliceBlue
AntiqueWhite
Aqua
Aquamarine
Azure
Beige
Bisque
Black
BlanchedAlmond
Blue
BlueViolet
Brown
BurlyWood
CadetBlue
Chartreuse
Chocolate
Coral
CornflowerBlue
Cornsilk
Crimson
Cyan
DarkBlue
DarkCyan
DarkGoldenrod
DarkGray
DarkGreen

DarkKhaki
DarkMagenta
DarkOliveGreen
DarkOrange
DarkOrchid
DarkRed
DarkSalmon
DarkSeaGreen
DarkSlateBlue
DarkSlateGray
DarkTurquoise
DarkViolet
DeepPink
DeepSkyBlue
DimGray
DodgerBlue
Firebrick
FloralWhite
ForestGreen
Fuchsia
Gainsboro
GhostWhite
Gold
Goldenrod
Gray
Green
GreenYellow
Honeydew
HotPink
HotTrack
IndianRed
Indigo
Ivory
Khaki

APPENDIX A – COLOR NAMES

Lavender
LavenderBlush
LawnGreen
LemonChiffon
LightBlue
LightCoral
LightCyan
LightGoldenrodYellow
LightGray
LightGreen
LightPink
LightSalmon
LightSeaGreen
LightSkyBlue
LightSlateGray
LightSteelBlue
LightYellow
Lime
LimeGreen
Linen
Magenta
Maroon
MediumAquamarine
MediumBlue
MediumOrchid
MediumPurple
MediumSeaGreen
MediumSlateBlue
MediumSpringGreen
MediumTurquoise
MediumVioletRed
MidnightBlue
MintCream
MistyRose

Moccasin
NavajoWhite
Navy
OldLace
Olive
OliveDrab
Orange
OrangeRed
Orchid
PaleGoldenrod
PaleGreen
PaleTurquoise
PaleVioletRed
PapayaWhip
PeachPuff
Peru
Pink
Plum
PowderBlue
Purple
Red
RosyBrown
RoyalBlue
SaddleBrown
Salmon
SandyBrown
SeaGreen
SeaShell
Sienna
Silver
SkyBlue

Appendix B – Change Log

Changes in version 6.3

Changes in version 6.3 included the following:

AdditionsNew Delegates

- GetnodeDirtyType

Changes in version 6.2

Changes in version 6.2 included the following:

AdditionsNew Delegates

- SetReferenceAttributes
- RemoveReferenceAttributes
- SetListRestartContinueNumbering
- RemoveRefVariable

Changes in version 6.1

Changes in version 6.1 included the following:

AdditionsNew Delegates

- GetExportedXMLForRefVariables

Changes in version 6.0

Changes in version 6.0 included the following:

AdditionsThe following additions were made in version 6.0.

Variable reference support

New Delegates

- AssignRefVariable
- GetRefVariable
- GetAllRefVariables
- ResolveReference

- GetInsertableElements
- GetNodeDefProperties

New Argument type

- StyleListParents

Changes

The following APIs were changed in version 6.0

Changed Delegates

- GetExtensibleUserInterface
- InsertNode
- DeleteNode
- InsertEmphasis
- InvokeMethodID

Legal notices

Quark XML Author™ for Microsoft® Word System Administration Guide

Copyright © 2022. All rights reserved. No part of this document may be reproduced in any form, including photocopying or translation to another language, without prior written consent of Quark Software Inc.

Copyright protection claimed includes all forms and matters now and hereinafter granted protection by statutory or judicial law, including unlimited protection for screen images generated from the software programs. Such images include but are not limited to windows, icons, and report displays.

This manual is provided without any warranty of any kind, either expressed or implied. Quark Software Inc. shall not be liable to any person or entity with respect to any liability, loss, or damage to be caused directly or indirectly by this manual. Information in this document is subject to change without notice, and does not represent a commitment on the part of the vendor. The software described in this manual is furnished under a license agreement and may be used or copied only in accordance with the terms of the agreement. No part of this document may be reproduced or transmitted without permission from Quark Software Inc.

Trademark Acknowledgments

Any or all products or brand names mentioned in the document are trade names, service marks, trademarks, or registered trademarks of their respective owners.